conference
proceedings

26th Large Installation System Administration Conference (LISA '12)

San Diego, CA, USA
December 9–14, 2012

Sponsored by
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
in cooperation with
ACM SIGOPS

Proceedings of the 26th Large Installation System Administration Conference
San Diego, CA, USA December 9–14, 2012

usenix ASSOCIATION

ISBN 978-931971-97-3

USENIX Association

Proceedings of LISA '12: 26th Large Installation System Administration Conference

December 9–14, 2012
San Diego, CA

Conference Organizers

Program Chair
Carolyn Rowland, *National Institute of Standards and Technology (NIST)*

Program Committee
Patrick Cable, *MIT Lincoln Laboratory*
Brent Chapman, *Google, Inc.*
Marc Chiarini, *Harvard University*
Mike Ciavarella, *Coffee Bean Software Pty Ltd*
Rudi van Drunen, *Competa IT and Xlexit Technology, The Netherlands*
Andrew Hume, *AT&T Labs—Research*
Tim Nelson, *Worcester Polytechnic Institute*
Mario Obejas, *Raytheon*
Mark D. Roth, *Google, Inc.*
Andrew Seely, *SAIC*
Steve VanDevender, *University of Oregon*
Nicole Forsgren Velasquez, *Utah State University*

Invited Talks Coordinators
Narayan Desai, *Argonne National Laboratories*
Rikki Endsley, *USENIX Association*
Cory Lueninghoener, *Los Alamos National Laboratory*
Kent Skaar, *VMware, Inc.*

Lightning Talks Coordinator
Lee Damon, *University of Washington*

Workshops Coordinator
Kyrre Begnum, *Norwegian System Architects*

Guru Is In Coordinator
Chris St. Pierre, *Wireless Generation*

Poster Session Coordinator
Marc Chiarini, *Harvard University*

USENIX Board Liaison
David N. Blank-Edelman, *Northeastern University*

Steering Committee
Paul Anderson, *University of Edinburgh*
David N. Blank-Edelman, *Northeastern University*
Mark Burgess, *CFEngine*
Alva Couch, *Tufts University*
Rudi van Drunen, *Competa IT and Xlexit Technology, The Netherlands*
Æleen Frisch, *Exponential Consulting*
Xev Gittler, *Morgan Stanley*
William LeFebvre, *Digital Valence, LLC*
Adam Moskowitz
Mario Obejas, *Raytheon*
Elizabeth Zwicky, *Consultant*

Education Director
Daniel V. Klein, *USENIX Association*

External Reviewers

Paul Anderson	David Byers	Tom Limoncelli	Chris Schanzle
John Arrasjid	Alva Couch	Cory Lueninghoener	Kent Skaar
Lori Barfield	Narayan Desai	Adam Moskowitz	Benoit “Tsuna” Sigoure
Kyrre Begnum	Matt Disney	Andrew Mundy	Josh Simon
Tim Bell	Chris Hooper	Mario Obejas	Aleksey Tsalolikhin
Gavin Brennan	Paul Krizak	Federico Sacerdoti	Avleen Vig
Mark Burgess	William LeFebvre	Chris St. Pierre	Gong Zhang

LISA '12:
26th Large Installation System Administration Conference
December 9–14, 2012
San Diego, CA

Wednesday, December 12

Storage and Data

Security and Systems Management

Thursday, December 13

Tools You Can Use

(Thursday, December 13 continues on p. iv)

Community and Teaching

You Can't Monitor It You Can't Manage It

Friday, December 14

Content, Communication, and Collecting

DNSSEC

If You Build It They Will Come

Welcome to the LISA '12 Proceedings!

If you're reading these before the end of the conference, now is the time to make a list of actionable things as take-aways from LISA:

- Met and exchanged info with at least 3 new people in the profession
- Found at least one new idea or solution that I can use now
- Found at least two ideas or solutions that make me think I should
- Learn more or change something I'm doing

These take-aways can come from tutorials, the technical conference, Birds-of-a-Feather sessions (BoFs), or the "hallway track." Don't discount the hallway track as a valuable source of information and a great networking opportunity. When else will you find 1000+ people, all in your profession, milling around the conference between tracks?

As for these proceedings, consider them a take-away. These proceedings include the 22 accepted papers and practice and experience reports out of the 50 submitted to the conference. All paper submissions were peer reviewed, including reviews by the LISA '12 Program Committee and a list of hand-selected external reviewers with specialized expertise in the specific domains covered by our varied submissions.

You'll find a broad range of topics including the continually popular configuration management, HPC, teaching (which has become a hot topic recently), monitoring, setting up a wireless network, managing IOS devices, and more.

Out of the 22 papers you'll find here, eight of them are refereed papers, 13 are practice and experience reports, and one was accepted as an invited talk. Refereed papers are more traditional with a focus on research and bleeding edge topics. Practice and experience reports are about real experiences from real sysadmins along with the lessons they learned along the way. Both tracks provide valuable insight into the profession and together they deliver a strong practical and theoretical core for the LISA technical program.

Read a few and see what you think. Maybe you will come away from your reading thinking that you too could submit a paper to LISA.

Thanks for attending!

Carolyn Rowland
Program Chair

HSS: A simple file storage system for web applications

Abstract

AOL Technologies has created a scalable object store for web applications. The goal of the object store was to eliminate the creation of a separate storage system for every application we produce while avoiding sending data to external storage services. AOL developers had been devoting a significant amount of time to creating backend storage systems to enable functionality in their applications. These storage systems were all very similar and many relied on difficult-to-scale technologies like network attached file systems. This paper describes our implementation and the operating experience with the storage system. The paper also presents a feature roadmap and our release of an open source version.

1. Introduction

All web applications require some type of durable storage system to hold the content assets used to present the look and feel of the application and in some cases the code that runs the application as well. Web scale applications require that the content be published from numerous sources and be delivered from numerous sources as well. There have been quite a few solutions to this problem over time but many of these past solutions have presented scalability problems, availability challenges, or both. We have implemented storage systems based on traditional network file systems like NFS[1] and we have also implemented storage systems based on more scalable cluster file systems like IBRIX Fusion[2], Lustre[3], and OrangeFS/PVFS[4,5]. These solutions have drawbacks related to the tight coupling of the server and client, the restricted ability to add and remove capacity from the system, the recovery behaviors of the system, the ability of the system to allow non-disruptive upgrades of software, single points of failure built into critical parts of the system, and limited if any multi-tenant capabilities.

The server and client dependencies in clustered file systems create a difficult-to-manage issue around updates of the software. In general, mismatched client and server versions do not work together, so the entire system must be shut down to perform upgrades. The multi-tenant requirement to use the system as a service makes isolation within the clustered file systems difficult other than through traditional file system layout structures which have limited flexibility. The interfaces that clustered file systems provide tend to look like traditional POSIX interfaces but almost all of them have unusual behaviors that many applications cannot understand or workaround without changes. The requirement to make application changes to accommodate the file system specifics makes clustered file systems unattractive to legacy applications.

The storage system we have created seeks to improve on the availability and operational characteristics of the storage systems we have used in the past while providing easily consumable front end semantics for the storage and retrieval of files by applications. We provide a minimal set of operations and rely on external components for any additional functionality that an application may need. We have built several mechanisms into the system that provide data durability and recovery. We have also added a number of load management features through internal and external mechanisms to reduce hotspots and allow for data migration within the system. The system is aware of its physical makeup to provide resilience to local failures. There are also autonomic behaviors in the system related to failure recovery. All of these

features are built and implemented using well-understood open source software and industry standard servers.

We illustrate here the design criteria, explain why the criteria were chosen, and the current implementation in the system. We follow up with a discussion of the observed behaviors of the system under production load for the past 24 months. We then present the feature roadmap and our invitation to add on to the system by open sourcing the project as it is today.

2. Design

These are the design criteria that we have selected to implement in the system as required features. They are in our opinion of general use to all web applications. They are all fully implemented in the system today.

2.1 Awareness of physical system configuration

In order to be resilient to physical component failure in the system as well as failures of the hosting site infrastructure the system has awareness of its physical makeup. This awareness is fairly granular with known components consisting of storage devices, servers which are containers of storage devices, and groups of servers referred to as a site.

Storage device awareness prevents storage of a file and its replicas on the same storage device which would be unsafe in the event of a failure of that device. Servers and the storage devices they contain have the same merit in the system for data protection. None of the copies of a file should reside on the same disk or server.

The grouping of servers into a site allows for the creation of a disaster recovery option where an application that chooses to store files in multiple sites can achieve higher availability in the event of a failure of an entire site. There is no requirement that the sites be physically separate. The preferred configuration does include multiple sites that do not share physical infrastructure.

2.2 Data protection through file replication

Data protection can be accomplished through hardware, software or a combination of both. The greatest flexibility comes from the software data protection. Software data protection improves the availability of the system and site failure resilience. File replication is the chosen method in the system. While space-inefficient compared to traditional RAID algorithms, the availability considerations outweigh the cost of additional storage.

Software control of the replication allows tuning based on a particular application's availability requirements independent of the hardware used and of other applications. In addition to tuning considerations, the software allows us the flexibility to specify how file writes are acknowledged. We can create as many copies as necessary before the write is acknowledged to the application. In general, the number of copies created synchronously is two, and then any additional copies, typically one more, are created by the system asynchronously. These additional asynchronous copies include any copies made to another site in the system.

2.3 Background data checking and recovery

Data corruption[6] is of concern for a system designed to store large numbers of objects for an indefinite amount of time. With the corruption problem in mind we have implemented several processes within the system that check and repair data both proactively and at access time.

The system runs a background process that runs constantly and compares file locations and checksums to the metadata. The background scrubber continuously reads files from the system and confirms that the correct number of copies

of a file are available and that the files are not corrupt by using a checksum. If the number of copies is incorrect, the number of file copies is adjusted by creating any missing copies or removing any additional copies. If a checksum fails on any file replica then the corrupt copy is deleted and the correct number of replicas is recreated by scheduling a copy process from one of the known good copies of the file.

In addition to the background process that constantly checks data, file problems can be repaired upon an access attempt of a missing or corrupt file. If the retrieval of a file fails due to a missing file or checksum error the request is served by one of the other file replicas that is available and checksums properly. The correct number of replicas is then created by scheduling a copy process from one of the known good copies of the file to a new location. Metadata reliability is handled by the metadata system.

2.4 Internal file system layout

Each HSS server is comprised of multiple file systems, one for each physical drive; file systems do not span multiple drives. A single top level directory will be the location of all the subdirectories that contain files. The directory structure of the file system will be identical on all hosts but file replica locations will vary by host. This is necessitated because each host *may* have different capacities, but the desire is to balance utilization across all hosts and spindles, precluding placement in the same directory on all systems.

Each node will have some number of physical disks. Each disk will be mounted on the top level directory as a subdirectory. Below the mount point of the directory are 4096 directories with names aaa to 999 using the hexadecimal notation. This structure is used to prevent potential problems with having a very large number of files in a directory. In the pathological case that all files on a given disk are 2KB with 1TB drives, we will not go over 125000 files in any directory with even file distribution before filling any individual drive. In fact, the majority of files stored have sizes between 10KB and 100KB, which reduces the possible number of files on a disk before it fills to somewhere between 2400 and 24000 files. This should be manageable for the system even as drives get larger. In the futures section there is a discussion of how the large numbers of files can be managed.

2.5 High performance database for system metadata

Metadata scalability and availability have direct affects on the system throughput and is ability to serve data. A number of options exist for systems that we believe are reliable enough to serve as the metadata engine in the system. Depending on the performance requirements, a well-scaled RDBMS system has been found to meet the needs of the system for a large variety of workloads. An additional benefit of using an RDBMS system is that they are widely available and well understood.

We have implemented the system with MySQL as the primary metadata engine. The data structure within the system is rather simple with almost no relation between the tables. Even though there are few relational aspects to the data, the use of a well understood RDBMS simplifies the metadata management aspects of the system and the development effort required to create the system. Most of the data is rarely changed and most of the tables are used for configuration of the system. This simplicity in the database can help maintain high throughput. There are only six tables in the database as shown below in figure 1.

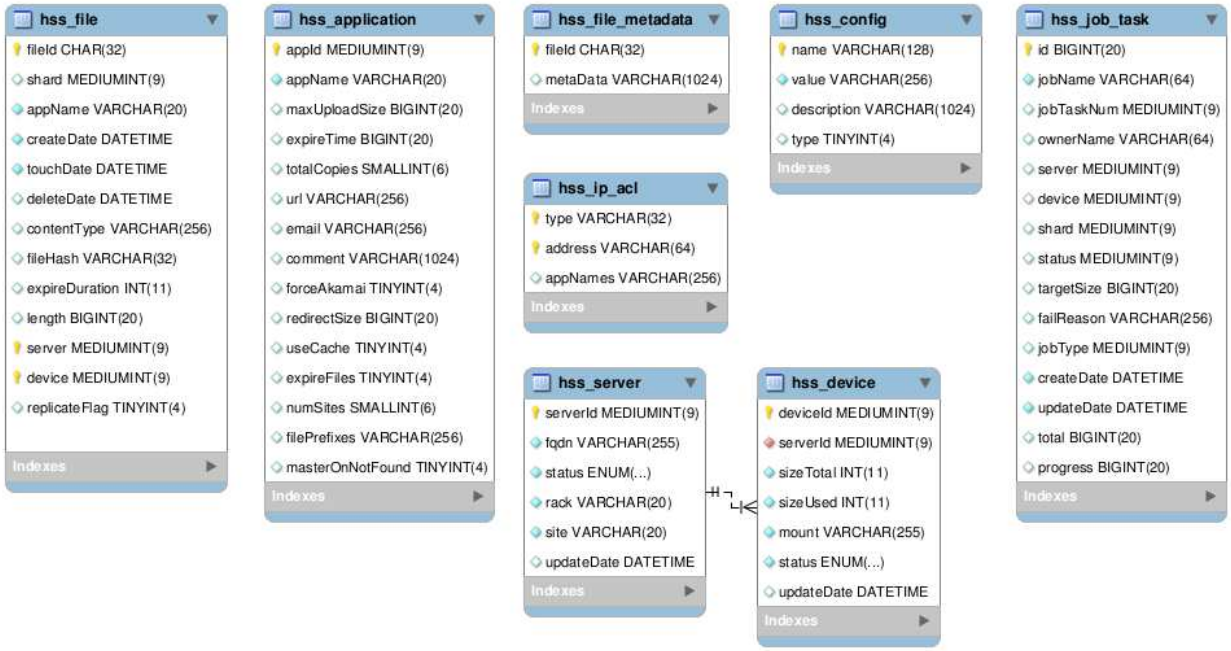


Figure 1

MySQL can be run in a partitioned or sharded fashion as the database grows to address potential performance concerns due to database growth. Replicas of the database are also maintained to create resilience to failures in the metadata system. A number of additional technologies exist from various sources to improve MySQL performance and availability while maintaining compatibility with the MySQL system. We have implemented several of those to improve the overall system.

We have also developed a version of the metadata system based on the VoltDB[7] database. VoltDB provides clustering for availability, an in-memory database engine for performance, and snapshots to durable storage for recovery and durability. The schema is unchanged when using VoltDB and the metadata engine is a configuration option for the storage nodes within the system.

2.6 Off the shelf open source components

The system is built from readily available open source software systems. The system is implemented on Linux servers running Apache Tomcat for the storage nodes and MySQL or VoltDB for the metadata system. These components were selected for their widespread adoption, functionality, transparency, and the deep knowledge about these systems prevalent in the technology community.

2.7 System scalability

The system has been designed to allow the addition or removal of capacity to increase both storage available and performance of the system. All of the changes to the system can be accomplished while the system remains online and completely available.

When additional storage nodes are added to the system, the new storage nodes are immediately available to take load from requesting applications. Additional write throughput from the new nodes is available immediately and read throughput of the new nodes comes online as files are stored on the new system through new writes or redistribution of data. Redistribution of stored data in the entire system or a subset

of the system is accomplished in the background to ensure that capacity utilization is roughly equal on all storage nodes in the system. It is anticipated that many files in the system are not commonly accessed and therefore those files can be moved without impact to the applications that own them.

Nodes can be removed from the system to scale down or migrate data. The system can be configured to disable writes to nodes that are being removed and then files are copied in an even distribution to all other remaining storage nodes. Once additional copies of the files have been made on the rest of the system, the files on the node being removed are deleted. This process continues until the node being removed has no files stored.

Metadata scalability tends to be more problematic but can still be accomplished within the constraints of the RDBMS being used for the metadata. In general, it is difficult to scale down the metadata engine. It is the expectation that the system will only increase metadata capacity and if the system is scaled down then the overhead in the metadata system will have to be tolerated.

2.8 Simple interface

The system is accessed though a very small number of web calls that, while somewhat restrictive, cover a large number of the use cases for web application storage systems. The system can write a file, read a file, delete a file, and replace a file. File object IDs are provided by the system in most cases but there is also a capability to allow applications to define their own object IDs. The system does not allow the updating of files. Examples of the system operations are discussed in section 3.

The external operations for the system are listed in Table

Operation	Example
Read	GET /hss/storage/appname/fileID
Write	POST /hss/storage/appname
Delete	DELETE /hss/storage/appname/fileId

Table 1

All of the external operations are atomic and return only success or failure. The client application is not expected to have to perform any operation other than perhaps a retry if appropriate. While the capabilities of the system may appear restrictive, we have found that adoption has been quite good. There are a number of internal operations that are used to manage the system behaviors.

The internal operations listed in Table 2 are only available to the storage nodes themselves. Administrative operations are an additional class of internal operations that are used for system configuration and batched file operations. The administrative operations are listed in table 3.

Operation	Example	Description
Copy	POST /hss/storage/internal_copy	Store replica on another storage node
Move	GET /hss/storage/internal_move	Move a file from one storage node to another
Migrate	GET /hss/storage/internal_migrate	Migrate a file from one server to another during healing process. The file is not removed from the old storage node

Table 2

Operation	Description
/hss/admin/addServer	add a server to a site configuration
/hss/admin/addDrive	add a drive to a server configuration
/hss/admin/changeServerStatus	set the server status to one of read-write, read-only, maintenance, or offline
/hss/admin/changeDriveStatus	set a drive status to read-write, read-only, or offline
/hss/admin/addApplication	add an application name to the configuration for files to be stored
/hss/admin/updateApplication	change application specific parameters about the way files are managed
/hss/admin/addJob	add a job to copy files to or from a device

Table 3

2.9 Network load balancer

The storage nodes in the system are accessed through a network load balancer to provide even distribution of incoming requests. Access to the metadata system is also provided by a network load balancer. This provides several benefits in addition to providing uniform request rates to all of the system components.

The load balancer implementation can remove unresponsive system nodes in a very short amount of time. The failure interval is also configurable providing a flexible tolerance for component failures. Unavailable system components do not cause requests to slow down or fail in the system except locally to a storage node for a very few requests that are in-flight during the actual failure.

Load balancers insulate client applications from the particulars of the system and allow changes to be made without having to update any external part of the system. The same is true of access to the metadata engine. The metadata is accessed through load balancers that provide access to the metadata system for reads, writes, or administrative background jobs independently and in some cases making use of replica copies for workload where data may not need to be up to date.

2.10 ACLs via IP address

The system has very little in the way of access controls. The system is designed to provide service to applications and the applications are expected to provide fine grained access controls and file security required. There are some basic controls available to restrict the system and assist applications with access control. The ACLs are provided not to prevent access to specific data or a subset of the data but to protect the system from bad external behaviors.

Read and write ACLs are defined for the system as a whole and on a per application basis. ACLs are specified by IP address. The system will reject write requests that do not come from systems that are explicitly defined within the ACL. The system will deny read request for systems that are explicitly defined within the ACL. The write ACL behavior is to only allow authorized systems to store files within the system. It is worth noting that only objects with file prefixes can be overwritten, so in the common use case for the system a file cannot be replaced and keep the same object ID. It is also possible for any allowed system to write files with any valid

application name. This can potentially become an accounting issue within the system but it doesn't place data at risk for overwrite or corruption in general. The read ACL behavior is to restrict reads from any clients that are found to be using the system improperly. The system is susceptible to bypassing application security to request an object if the object ID can be discovered.

2.11 Files served by proxy or redirect based on size

Any node may receive a request for any file in the system. Not all files will reside locally on every system node. The system serves file reads based on GET requests and those are treated like standard HTTP GET requests. If the file is not stored locally, a node will either retrieve the file from the node where it is stored and return the file itself or send a HTTP REDIRECT to the requester identifying a node that stores a copy of the file. The system implements a tunable size for the server redirect case. Files below the size threshold will be returned from the initial requesting node. Files above the size threshold will be redirected. This behavior allows the system to be tuned based on load and typical file size within and application.

2.12 Commonly served files cached

Typically we have seen that a small number of popular files are commonly requested. In order to avoid constant proxy or redirect requests a cache of commonly accessed files is implemented on the nodes. A tunable number of files can be cached on every node. The additional space consumed by the cache is more than offset by the read performance trade off of avoiding additional communication about file locations. The cached files also provide some insight into what files are popular and may be candidates for placement in an external CDN.

2.13 File expiration

The system has the ability to automatically delete files after a tunable interval has expired. The interval can be defined per application in the system. This enables the temporary storage of files for applications that make use of transient data. Deletion of files based on expiration offloads the need to keep track of file creation and lifetime from all applications that need this feature and implements it in the storage system consistently for all applications.

2.14 Application metadata stored with the file

The system provides a simple mechanism to store any additional metadata about a file along with the file record in the metadata system. This data is opaque to the storage system and simply exists as an assistive feature for applications. An application may store some metadata, perhaps a description, with the file and that metadata will be returned when the file is retrieved. The expectation is that this data is used within the storing and requesting application to provide some additional functionality. At some point it is expected that the system would be enhanced to make use of the application metadata stored to make the system more responsive and useful to application consumers.

2.15 Applications are system consumers with local configurations

The system assumes that files will be stored and retrieved by applications. Applications that make use of the system are expected to implement any necessary controls on files they store. The nature of the system keeps file storage as simple as possible and moves responsibilities for much of traditional content management and fine access control to external entities that make up the applications.

2.16 File accounting

While the system does have the concept of application to file relationships for multi-

tenancy concerns, it has no hierarchical namespace, so files stored are kept track of external to the storage system by the owning applications. This has the affect of allowing any file to be stored anywhere in the system for improved scalability and availability without concern for relationships between files. Multiple applications can store unique copies of the same file. While that duplication of content is space-inefficient, it prevents dependencies between applications within the storage system. The space overhead is preferred to potential conflicts between applications. It may be possible in a future update to the system to provide object level de-duplication to eliminate excessive copies of the same file.

The relationship between application owners and files allows the system to provide utilization reporting for capacity planning and usage metering. The system provides sufficient detail about files to report on capacity consumed per application, how much data is transferred per application, file specific errors, and several other usage based metrics. These metrics are used to predict when the system will require expansion or when access to files has become unbalanced for any reason. The metrics also enable the examination of usage trends within applications so that customers can be notified of observed changes to confirm that those changes are as expected.

2.17 Monitoring integrated with existing systems

There are several external mechanisms that are responsible for monitoring the system and its components. Based on events that occur in the system a number of external systems can be used to manage and adjust the system using the administrative operations. External systems are used to monitor throughput, software errors, and hardware errors. The external systems used are widely used industry standard systems.

Nagios[8], for example, is used to monitor the hardware components of the system. In the event of a disk failure, a Nagios monitor detects the occurrence. When that happens, an alert is sent to notify the system administrators, and the drive status is changed from read-write to offline automatically. This prevents any new requests for writes and any new read requests from using the failed disk drive and causes any at risk data to be re-protected by background processes.

3. Implementation

A high level diagram of the system and how the components interact is shown in Figure 2. A description of the external operations is provided to get a better feel for how the system functions from an application perspective. IP address access control checks are included in all operations.

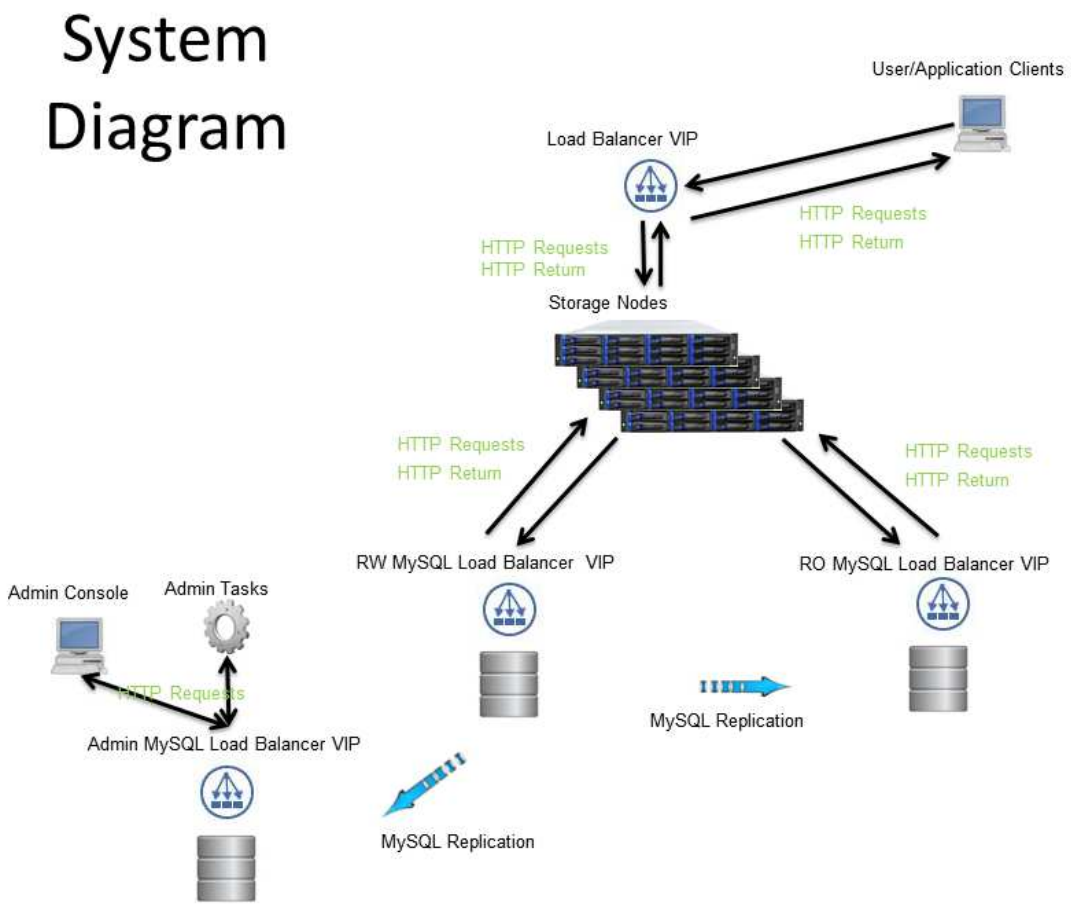


Figure 2

3.1 Write a file

When a POST request is sent to the system to write a file, the storage node that receives the request calculates the object ID and picks a local location to write the file. The storage node then inserts into the database the server, disk, and object ID of the file as well as some data about the file such as mime type and creation time. It then picks another node within the same rack if possible based on server availability and space free to write a second replica of the file. When the replica creation is successful the database is updated with the details that pertain to the replica. The write is then acknowledged as successful to the client once the minimum number of copies required is created. The default minimum number of copies is two but it is configurable per application. Based on the application configuration any additional replicas of the file are created at this time using the POST /hss/storage/internal_copy method from storage node that handled the original request. Once the minimum number of replicas is written a file is considered to be safe.

In the event of a failure before the first file write completes, the system retries the write and the process restarts. If the retries are unsuccessful after a limit is reached, the system returns a failure and the application is expected to retry the write. If any additional writes of file replicas fail, the system will retry the writes until the minimum replicas requirement is achieved and then any additional replicas will be created in the background.

If background replica creations fail, either by making too few or too many replicas, then the expectation is that the system background file check process will detect the improper number of replicas and take the appropriate action by making any additional replicas required or removing any additional replicas beyond what is required by the application configuration.

3.2 Read a file

A file read GET request specifies the application and objectID as part of the path sent to the system. The receiving storage node looks up the object ID and finds the location of all replicas. A replica is selected from the list of locations by preferring the replicas that are local if there are any, then within the same rack and then the same site. If the file is local it is simply served. If the file is not local, it is retrieved and served by the requesting storage node if it is smaller in size than the threshold specified for redirect by the application. If the file is larger than the redirect size, a redirect is served to the client that then requests the file for a server that has the file locally. In the case of a storage node serving a file it does not own, the file is placed in a local cache on the serving storage node with the expectation that commonly accessed files will be requested continuously from nodes that do not have a local copy due to the load balanced access to the system. This cache will avoid future lookups and file retrievals from other nodes for heavily accessed files. It also has the effect of making many copies of the file throughout the system to enable better concurrency when the file is requested.

In the case of a lookup failure from the database, an http 404 error is served to the requesting process. In order to avoid transient lookup failures a number of retry options are available. In the simple case a number of retries is attempted before returning the 404 error. In the case where a database replica is used to service some database lookups for read requests, a number of retries are attempted and then the lookup can be attempted on the master copy of the database if the application is configured to do so. The reasoning behind this second set of lookups is that replication lag between the master and the replica database may account for the lookup failure. If that is the case, the lookup in the master database should succeed if the replica database lookup fails. In general, the replication lag is less than 10 seconds within the current system, but some applications exhibit a usage pattern that causes an immediate read after a write that makes the lookup on the primary necessary. The metadata replication we use is asynchronous so it is not possible to require metadata replication to complete before a write is acknowledged to the client. Alternative metadata storage systems that have synchronous replication would also solve the problem of the replica metadata database being slightly behind the primary metadata database.

Occasionally a replica copy that is selected by the system will fail to be available either because the file is not found on the owning server or the file will fail the checksum when it is retrieved. If either of these cases occurs, the file will be served from one of the other replicas in the list returned by the lookup. The internal GET /hss/storage/internal_move method is then used to replace the missing or bad file replica.

3.3 Delete a file

The deletion of a file is the simplest of the external cases. The request to delete marks the files for deletion in the database and then deletes the replicas of the file and the records from the database. Any failures during the delete process are expected to be handled by the background checking processes of the system. If files are deleted

but the database entries remain for some reason, the records marked for deletion are removed by the background processes of the system. It is possible for the delete process to orphan files on storage nodes that are not referenced by the database. The process to recover from this occurrence requires looking up all files found on the storage nodes and comparing them to existing database entries. If the entries do not exist, the files can be safely deleted as there is no reference to them and they cannot be served.

3.4 Recovery of errors

A number of errors both well-understood and unknown are expected to occur within the system. These errors may include things like file system or storage node failures destroying data and causing the number of replicas to fall below the desired number, database lookup failures, storage nodes unavailable for extended periods due to hardware maintenance, network failures, and many others. The system is designed to be as resilient as possible to errors. The system has implemented recovery processes for the errors that are known to occur and updates are made to the system as new errors are detected and the root causes are discovered to prevent new errors from recurring.

3.5 Configuration

Application specific settings are managed from the administrative interface for the system. Application settings control the way files are stored in a number of ways. The specific parameters that can be set for an application are:

Expire Time – number of seconds a file will be kept before automatic deletion. Automatic deletion requires Expire Files to be set true.

Max Upload Size (MB) – The size limit of a file that may be written. There is no built in limit by default.

Copies – The number of replicas of a file required.

Sites – The number of sites that file replicas must be stored in.

Use Cache – Enables caching of often requested files on nodes that do not normally have local copies.

Expire Files – Enables or disables automatic deletion of files.

Use Master on 404 – Enables or disables queries for files that fail on read replicas of the metadata database to be retried on the master metadata database before returning a failure to the application.

Redir Size (MB) – The threshold size for a file when the request will be redirected instead of proxied by the storage node receiving the request.

File Prefixes – ObjectID prefixes that are specified by an application. An application may specify the ObjectID of a file and the system requires a consistent prefix for all of the files in that case. The prefix is usually the application name. This is useful for applications where the filename can be mapped directly to the file content such as a geographic location used in maps.

4. Results and observation

The system as described above has been in production since August of 2010. In that time it has served billions of files for dozens of applications. The system has good scalability and has serviced over 210 million requests in a 24 hour period. As of this time the system manages just over 380 million files. The file count includes all replicas. An analysis of the usage data from the system reveals that about one third of the current files in the system are active. The feature set is in-line with the intended use case. The feature set of the system also compares well with ongoing work on open storage systems designed for the same use case. Table 4 shows a feature comparison between the Openstack object storage system[9] (Swift) current and future features and the current features of the system presented here.

Open Stack Swift Storage System Current and Roadmap Feature Comparison

Feature	Openstack Swift	HSS
Store and manage files programmatically via API	Y	Y
Create public or private containers	Y	Y
Commodity hardware	Y	Y
HDD/Node failure agnostic	Y	Y
Unlimited storage	Y	Y
Multi-dimensional scalability	Y	Y
Account/Container/Object structure	Y	Y
Built-in replication	Y	Y
Easily add capacity	Y	Y
No central database	Y	N
RAID not required	Y	Y
Built in management utilities	Y	Y
Drive auditing	Y	Y
CLI	Y	Y
Improved client IP logging	N	Y
Option for replication	N	Y
Multi cluster syncing	N	Y
Large single uploads	N	Y
Self-destructing files	N	Y

Table 4

The load on the system is highly variable and dependent on external applications but the behaviors of the system are highly consistent as request load varies. Figure 3 shows the load over a thirty day period. Figure 4 shows the service time for all requests.

Daily Report

Each unit (▬) represents 8,000,000 requests or part thereof.

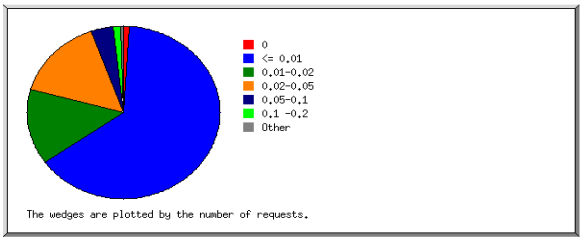
date	reqs	pages	Tbytes
14/Apr/12	34679957	117867	0.73
15/Apr/12	64603291	153938	1.36
16/Apr/12	82464950	180637	1.64
17/Apr/12	84348508	758508	1.64
18/Apr/12	85339078	356551	1.62
19/Apr/12	79331053	165346	1.58
20/Apr/12	89434038	227067	1.53
21/Apr/12	91277900	215410	1.28
22/Apr/12	71085800	209439	1.32
23/Apr/12	81083544	255295	1.62
24/Apr/12	77708581	300457	1.57
25/Apr/12	115020446	180783	1.50
26/Apr/12	102901308	130869	1.52
27/Apr/12	105428575	252088	1.50
28/Apr/12	92073035	185259	1.36
29/Apr/12	92852503	169668	1.25
30/Apr/12	111368504	194746	1.55
1/May/12	136847107	485617	1.54
2/May/12	158098801	508147	1.46
3/May/12	154312347	304603	1.48
4/May/12	141567232	268188	1.35
5/May/12	144097556	254431	1.21
6/May/12	152976182	232612	1.16
7/May/12	146743200	232870	1.58
8/May/12	119065960	378701	1.88
9/May/12	131036207	329123	2.15
10/May/12	109814165	187264	1.46
11/May/12	91329567	151346	1.34
12/May/12	148389620	115511	1.12
13/May/12	161159990	99295	1.14
14/May/12	212504364	121640	1.46
15/May/12	16494717	12697	0.09

Busiest day: 14/May/12 (212,504,364 requests).

Figure 3

In some cases days with fewer requests transfer more data than days with higher numbers of requests. This shows that not only is the request rate variable but file size is also quite variable.

Processing Time Report



seconds	reqs	%reqs
0	37514103	1.05%
<= 0.01	2296882027	64.00%
0.01-0.02	511234749	14.24%
0.02-0.05	547119494	15.24%
0.05-0.1	134610412	3.75%
0.1 -0.2	48795349	1.36%
0.2 -0.5	8474686	0.24%
0.5 -1	1974738	0.06%
1- 2	960778	0.03%
2- 5	599237	0.02%
5- 10	327617	0.01%
10- 20	299895	0.01%
20- 60	234591	0.01%
60-120	31508	
120-300	5914	
300-600	971	
> 600	388	

Figure 4

The overall observed response time of the system is quite acceptable with less than .5% of the requests taking more than 200ms and less than 2% of requests taking more than 100ms. This data includes all reads, writes and errors and we believe that the abnormally long response times are in actuality failures. The number of these abnormal events is very small and also quite acceptable.

Using only features of the system several infrastructures have been built and components have also been added and removed in existing infrastructures while the system has been under load and in production. The addition or removal of hardware components is simple and well-understood. The ability to update the system software components is more complicated but can also be done without taking the system offline. The existing systems have seen software updates, metadata database changes, and metadata database migrations without interruption in service.

5. Lessons learned

Many details of the system are implemented as conceived in the original design. There are still quite a few behaviors and functions of the system that can only be discovered though the use and maintenance of the system. The need to modify the way the system functions becomes apparent through its use by applications and administrators.

5.1 Data protection

There are several aspects of data protection that include both durability of the data and the availability of the data. Data replication within the system protects availability by ensuring that a file in the system is stored on multiple redundant components and it protects durability through that redundancy. There are some areas that require additional focus with file replication in practice.

The ability to rapidly recover large numbers of small files is limited due to several issues. Large numbers of small files in Linux file systems are difficult to access efficiently. Significant thought was given to creating a file system directory layout on the storage servers that would limit the impact of this problem but it still exists when very high numbers of small files are present even if they are efficiently stored in the file system. Additionally, small file transfers when files are replicated between storage servers are limited in the amount of bandwidth that can be utilized.

Another issue that occurs with the data protection availability aspects of the system is the I/O behaviors of the Tomcat servers when accessing a failed device of file system during the initial failure. There are a number of abnormal behaviors that prevent the system from returning the correct file to an application. Load spikes, crashes, and other oddities can occur when retrieving data. One response to the issue of failed physical devices can be to add local RAID data protection on a storage node to remove the possibility of a single device failure making a file system unavailable. This has the effect of increasing cost and complexity which are both to be avoided.

5.2 Metadata replication

Metadata replication is a requirement for the system in disaster scenarios as well as maintenance scenarios. It is temping to make use of the replicated metadata subsystem to assume some of the workload of the system. The benefits of this approach are that the replicated metadata subsystem is exercised to validate its proper functionality and that some load can be reduced on the primary metadata subsystem.

It is attractive to attempt to increase the total read capacity of the system by performing any reads from both the primary and the replicated metadata. The system is built with this capability in mind but the initial implementation suffered from the lag between the two systems. Applications that write a file and then immediately attempt a

read of that data initially received data access errors if the read was requested from the replicated metadata prior to the completion of replication. A retry case was added to always check the primary before returning a failure to the requesting client.

It is preferable to run administrative and maintenance jobs on the data from the replicated metadata subsystem. This prevents internal housekeeping from impacting the system. Many of the administrative jobs are long running and can be strenuous and having a secondary copy of the data isolates these jobs very effectively. Ad hoc jobs are also run against the replicated metadata subsystem.

5.3 Configuration tuning

The default configuration for the software components of the system are acceptable for integrating and bringing up the system. These configuration variables require adjusting to better manage the load and improve the overall experience with the system. There are many limits within the system based on the initial configuration that can safely be modified to increase the capabilities of the system.

Variables control memory utilization, thread limits, network throughput, and dozens of other functions within the system. These functions can be improved to the overall benefit of the system by examining the system and observing the limits. Many of the limits are imposed by the configuration and not by any physical or functional aspect of the system. Making informed changes to the configuration yields significant benefits.

6. Conclusion

To store and manage the large numbers of objects associated with web applications, the system is an extremely good fit. It offloads the need for application developers to manage where files are stored and how they will be protected. The system also provides a consistent interface so that application developers can avoid having to relearn how to store files for each new application that is developed.

From an operational point of view the system requires very little immediate maintenance due to its significant automation and redundancy. Repairs can be effected to subcomponents of the system in a delayed fashion without impacting the applications that use the system. Modifications to the system can be made while maintaining availability and durability of the data. These aspects make the system very lightweight for operators to manage.

The work done to develop this system and its usage in production applications for an extended period of time show that it is possible to build a system that is simple, reliable, scalable and extensible. The use of common industry standard hardware and software components makes the system economical to implement and easy to understand. There are certainly use cases where the system is not a good fit and should not be used but it is an excellent choice for the use around which it was designed.

There is room for improvement in the feature set and efficiency of the system. With that in mind there are some improvements that definitely should be investigated for integration into future versions of the system. A list of future planned improvements includes container files to address file management and performance concerns, lazy deletes to disconnect housekeeping operations from online operations, improved geographic awareness to improve access latency, and a policy engine to manage file placement and priority in the system.

References

[1] Network File system
http://tools.ietf.org/html/rfc1094
http://tools.ietf.org/html/rfc1813
http://tools.ietf.org/html/rfc3530
[2] IBRIX Fusion
http://en.wikipedia.org/wiki/IBRIX_Fusion
[3] Lustre
http://wiki.lustre.org/
[4] Orange File System Project
http://www.orangefs.org/
[5] PVFS
http://www.pvfs.org/
[6] Panzer-Steindel, Bernd Data Integrity April 2007 CERN/IT
[7] VoltDB
http://community.voltdb.com/
[8] Nagios
http://www.nagios.org/
[9] Openstack Object Storage (Swift)
http://openstack.org/projects/storage/

IDO: Intelligent Data Outsourcing with Improved RAID Reconstruction Performance in Large-Scale Data Centers

Suzhen Wu[12]*, Hong Jiang*[2]*, Bo Mao*[2]
[1]*Computer Science Department, Xiamen University*
[2]*Department of Computer Science & Engineering, University of Nebraska-Lincoln*
suzhen@xmu.edu.cn, {jiang, bmao}@cse.unl.edu

Abstract

Dealing with disk failures has become an increasingly common task for system administrators in the face of high disk failure rates in large-scale data centers consisting of hundreds of thousands of disks. Thus, achieving fast recovery from disk failures in general and high on-line RAID-reconstruction performance in particular has become crucial. To address the problem, this paper proposes *IDO* (*Intelligent Data Outsourcing*), a proactive and zone-based optimization, to significantly improve on-line RAID-reconstruction performance. IDO moves popular data zones that are proactively identified in the normal state to a surrogate set at the onset of reconstruction. Thus, IDO enables most, if not all, user I/O requests to be serviced by the surrogate set instead of the degraded set during reconstruction.

Extensive trace-driven experiments on our lightweight prototype implementation of IDO demonstrate that, compared with the existing state-of-the-art reconstruction approaches WorkOut and VDF, IDO simultaneously speeds up the reconstruction time and the average user response time. Moreover, IDO can be extended to improving the performance of other background RAID support tasks, such as re-synchronization, RAID reshape and disk scrubbing.

1 Introduction

RAID [16] has been widely deployed in large-scale data centers owing to its high reliability and availability. For the purpose of data integrity and reliability, RAID can recover the lost data in case of disk failures, a process also known as *RAID reconstruction*. With the growing number and capacity of disks in data centers, the slow performance improvement of the disks and the increasing disk failure rate in such environments [18, 20], the RAID reconstruction is poised to become the norm rather than the exception in large-scale data centers [2,5,6]. Moreover, it was also pointed out that the probability of a second disk failure in a RAID system during reconstruction increases with the reconstruction time: approximately 0.5%, 1.0% and 1.4% for one hour, 3 hours and 6 hours of reconstruction time, respectively [6]. If another disk failure or latent sector errors [3] occur during RAID5 reconstruction, data will be lost, which is unacceptable for end users and makes the use of RAID6 more necessary and urgent. Therefore, the performance of on-line RAID reconstruction is of great importance to the reliability and availability of large-scale RAID-structured storage systems.

A number of optimizations have been proposed to improve the on-line RAID-reconstruction performance [8, 12, 21–24, 26, 28–31]. However, all of them are failure-induced or *reactive* optimizations and thus passive. In other words, they are triggered *after* a disk failure has been detected and focus on either improving the reconstruction workflow [23, 26, 30] or alleviating the user I/O intensity during RAID reconstruction [24, 28, 29] but *not both*. In fact, our workload analysis reveals that the reactive optimization is far from being adequate (see Section 2.3 for details).

On the other hand, from extensive evaluations and analysis, our previous studies and research by others have found that user I/O intensity during reconstruction has a significant impact on the on-line RAID-reconstruction performance because there are mutually adversary impacts between reconstruction I/O requests and user I/O requests [23, 24, 28]. This is why the time spent on the on-line RAID reconstruction is much longer than that on its off-line counterpart [7]. However, existing on-line RAID reconstruction approaches, such as WorkOut and VDF, only exploit the temporal locality of workloads to reduce the user I/O requests during reconstruction, which results in very poor reconstruction performance under workloads with poor temporal locality, as clearly evidenced in the results under the Microsoft Project trace that lacks temporal locality (see Section 4

for details). Therefore, we strongly believe that both the temporal locality and spatial locality of user I/O requests must be simultaneously exploited to further improve the on-line RAID-reconstruction performance.

Based on these observations, we propose a novel reconstruction scheme, called *IDO* (Intelligent Data Outsourcing), to significantly improve the on-line RAID-reconstruction performance in large-scale data centers by proactively exploiting data access patterns to judiciously outsource data. The main idea of IDO is to divide the entire RAID storage space into zones and identify the popularity of these zones in the normal operational state, in anticipation for data reconstruction and migration. Upon a disk failure, IDO reconstructs the lost data blocks belonging to the hot zones prior to those belonging to the cold zones and, at the same time, migrates these fetched hot data to a surrogate RAID set (*i.e.*, a set of spare disks or free space on another live RAID set [28]). After all data in the hot zones is migrated, most subsequent user I/O requests can be serviced directly by the surrogate RAID set instead of the much slower degraded RAID set. By simultaneously optimizing the reconstruction workflow and alleviating the user I/O intensity, the reconstruction speed of the degraded RAID set is accelerated and the user I/O requests are more effectively serviced, thus significantly reducing both the reconstruction time and the average user response time.

The technique of data migration has been well studied for performance improvement [1, 10, 11] and energy efficiency [17, 25] of storage systems, IDO adopts this technique in a unique way to significantly optimize the increasingly critical RAID reconstruction process in large-scale data centers. Even though IDO works for all RAID levels, we have implemented the IDO prototype by embedding it into the Linux software RAID5/6 module as a representative case study to assess IDO's performance and effectiveness. The extensive trace-driven evaluations show that IDO speeds up WorkOut [28] by a factor of up to 2.6 with an average of 2.0 in terms of the reconstruction time, and by a factor of up to 1.7 with an average of 1.3 in terms of the average user response time. IDO speeds up VDF [24] by a factor of up to 4.1 with an average of 3.0 in terms of the reconstruction time, and by a factor of up to 3.7 with an average of 2.3 in terms of the average user response time.

More specifically, IDO has the following salient features:

- IDO is a *proactive* optimization that dynamically captures the data popularity in a RAID system during the normal operational state.
- IDO exploits both *the temporal locality and spatial locality* of workloads on all disks to improve the on-line RAID-reconstruction performance.
- IDO optimizes both the reconstruction workflow and user I/O intensity to improve the RAID-reconstruction performance.
- IDO is simple and independent of the existing RAID tasks, thus it can be easily extended to improve the performance of other background tasks, such as re-synchronization, RAID reshape and disk scrubbing.

The rest of this paper is organized as follows. Background and motivation are presented in Section 2. We describe the design of IDO in Section 3. Methodology and results of a prototype evaluation of IDO are presented in Section 4. The main contributions of this paper and directions for the future research are summarized in Section 5.

2 Background and Motivation

In this section, we provide the necessary background about RAID reconstruction and key observations that motivate our work and facilitate our presentation of IDO in the later sections.

2.1 RAID reconstruction

Recent studies of field data on partial or complete disk failures in large-scale data centers indicate that disk failures happen at a higher rate than expected [3, 18, 20]. Schroeder & Gibson [20] found that annual disk replacement rates in the real world exceed 1%, with 2%-4% on average and up to 13% in some systems, much higher than 0.88%, the annual failure rates (AFR) specified by the manufacturer's datasheet. Bairavasundaram et al. [3] observed that the probability of latent sector errors, which can lead to disk replacement, is 3.45% in their study. The high disk failure rates, combined with the continuously increasing number and capacity of drives in large-scale data centers, are poised to render the reconstruction mode the common mode, instead of the exceptional mode, of operation in large-scale data centers [5, 6].

Figure 1 shows an overview of on-line reconstruction for a RAID5/6 disk array that continues to serve the user I/O requests in a degraded mode. The RAID-reconstruction thread issues reconstruction I/O requests to all the surviving disks and rebuilds the data blocks of the failed disk to the new, replacement disk. In the meantime, the degraded RAID5/6 set must service the user I/O requests that are evenly distributed to all the surviving disks. Thus, during on-line RAID reconstruction, reconstruction requests and user I/O requests will compete for

the bandwidth of the surviving disks and adversely affect each other. User I/O requests delay the reconstruction process while the reconstruction process increases the user response time. Previous studies [23, 24, 28] have demonstrated that reducing the amount of user I/O traffic directed to the degraded RAID set is an effective approach to simultaneously reducing the reconstruction time and alleviating the user performance degradation, thus improving both reliability and availability of storage systems.

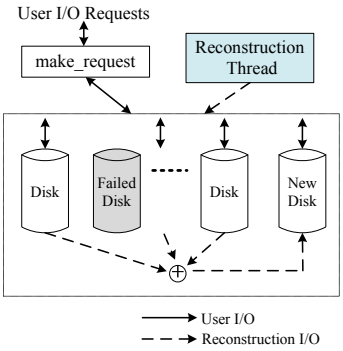


Figure 1: An overview of on-line reconstruction process for a RAID5/6 disk array.

2.2 Existing reconstruction approaches

Since RAID [16] was proposed, a rich body of research on the on-line RAID reconstruction optimization has been reported in the literature. Generally speaking, these approaches can be categorized into two types: optimizing the reconstruction workflow and optimizing the user I/O requests.

The first type of reconstruction optimization methods improve performance by adjusting the reconstruction workflow. Examples of this include SOR [9], DOR [8], PR [12], Live-block recovery [21], PRO [23], and JOR [27]. DOR [8] assigns one reconstruction thread for each disk, unlike SOR [9] that assigns one reconstruction thread for each stripe, allowing DOR to efficiently exploit the disk bandwidth to improve the RAID reconstruction performance. Live-block recovery [21] and JOR [27] exploit the data liveness semantics to reduce the RAID reconstruction time by ignoring the "dead" (no longer used) data blocks on the failed disk. PRO [23] exploits the user access locality by first reconstructing the hot data blocks on the failed disk. When the hot data blocks have been recovered, the reconstruction process for read requests to the failed disk can be significantly reduced, thus reducing both the reconstruction time and user I/O response time. Although the above reconstruction-workflow-optimized schemes can also improve the user I/O performance, the improvement is limited because the user I/O requests still must be serviced by the degraded RAID set. Therefore, the contention between user I/O requests and reconstruction I/O requests still persists.

The second type of reconstruction optimization methods improve performance by optimizing the user I/O requests. Examples of this include MICRO [30], WorkOut [28], Shaper [29], and VDF [24]. These optimization approaches directly improve the user I/O performance during reconstruction while simultaneously improving the reconstruction performance by allocating much more disk resources to the reconstruction I/O requests. MICRO [30] is proposed to collaboratively utilize the storage cache and the RAID controller cache to reduce the number of physical disk accesses caused by RAID reconstruction. VDF [24] improves the reconstruction performance by keeping the user requests belonging to the failed disk longer in the cache. However, if the requested data belonging to the failed disk have already been reconstructed to the replacement disk, the access delays of these user requests will not be further improved because they behave exactly the same as those of the data blocks belonging to the surviving disks [13]. Therefore, when VDF is incorporated into PRO [23], the improvement achieved by VDF will be significantly reduced. Both Shaper [29] and VDF [24] use the reconstruction-aware storage cache to selectively filter the user I/O requests, thus improving both the reconstruction performance and user I/O performance. Different from them, WorkOut [28] aims to alleviate the user I/O intensity on the entire degraded RAID set, not just the failed disk, during reconstruction by redirecting many user I/O requests to a surrogate RAID set.

While optimizing the user I/O requests can also reduce the on-line RAID reconstruction time, the performance improvement of the user-I/O-requests-optimized approaches above is limited. For example, both WorkOut and VDF only exploit the temporal locality of read requests, ignoring the beneficial spatial locality existing among read requests. Moreover, VDF gives higher priority to the user I/O requests addressed at the failed disk. However, the RAID reconstruction process involves all disks and the user I/O requests on the surviving disks also affect the reconstruction performance. And most importantly, the access locality tracking functions in these schemes are all initiated after a disk fails, which is passive and less effective.

2.3 Reactive vs. Proactive

The existing reconstruction optimizations initiate the reconstruction process only after a disk failure occurs, which we refer to as *failure-induced* or *reactive* optimizations, and thus are passive in nature. For example, PRO [23] and WorkOut [28] identify the popular data during reconstruction, which may result in insufficient

identification and exploitation of popular data. Compared with the normal operational state, the reconstruction period is too short to identify a sufficient amount of popular data by the reconstruction optimizations, as clearly evidenced by our experimental results in Section 2.5.

If the user I/O requests are monitored in the normal operational state, the popular data zones can be proactively identified before and in anticipation of a disk failure. Once a disk failure occurs, the optimization works immediately and efficiently by leveraging the data popularity information already identified. We call this process a *proactive* optimization, to contrast to its reactive counterpart. Figure 2 shows a comparison of user I/O performance between a reactive optimization and a proactive optimization, where the performance is degraded by the RAID reconstruction and returns to its normal level after completing the recovery. We can see that the proactive approach takes effect much faster than the reactive approach, shortening the reconstruction time. The reason is that the proactive approach with its popular data already accurately identified prior to the onset of the disk failure, can start reconstructing lost data immediately without the substantially extra amount of time required by the reactive approach to identify the popular data blocks.

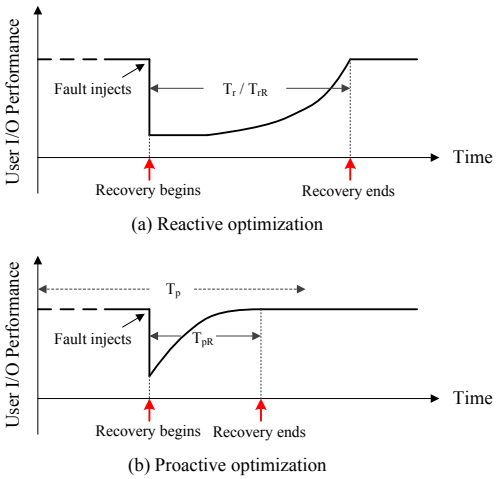


Figure 2: Comparisons of user I/O performance and reconstruction time between (a) reactive optimization and (b) proactive optimization. Note that T_r and T_p indicate the period of hot data identification, T_{rR} and T_{pR} denote the reconstruction time, and $T_r = T_{rR}$. In general, $T_r \ll T_p$ and $T_{rR} > T_{pR}$.

In large-scale data centers consisting of hundreds of thousands of disks, proactive optimization is very important because the disk-failure events are becoming the norm rather than the exception, for which RAID reconstruction is thus becoming a normal operation [5, 6].

2.4 Temporal locality vs. Spatial locality

In storage systems, access locality is reflected by the phenomenon of the same storage locations or closely nearby storage locations being frequently and repeatedly accessed. There are two dimensions of access locality. Temporal locality, on the time dimension, refers to the repeated accesses to specific data blocks within relatively small time durations. Spatial locality, on the space dimension, refers to the clustered accesses to data objects within small regions of storage locations within a short timeframe. These two access localities are the basic design motivations for storage-system optimizations.

Previous studies on RAID-reconstruction optimizations, such as VDF and WorkOut, use *request-based* optimization that only exploits the temporal locality of workloads, but not the spatial locality to reduce user I/O requests. PRO [23] and VDF [24] only focus on optimizing (*i.e.*, tracking or reducing) the user I/O requests to the failed disk, thus they ignore spatial locality and the impact of the user I/O requests on the surviving disks that also have notable performance impact on RAID reconstruction. Moreover, given the wide deployment of large-capacity DRAMs and flash-based SSDs as cache/buffer devices above HDDs to exploit temporal locality, the visible temporal locality at the HDD-based storage level is arguably very low. This is because of the filtering of the upper-level caches, while the visible spatial locality remains relatively high. The high cost of DRAMs and SSDs relative to that of HDDs makes good design sense for new system optimizations to put the large and sequential data blocks on HDDs for their high sequential performance, but cache the random and hot small data blocks in DRAMs and SSDs for their high random performance [4, 19]. As a result, these new system optimizations will likely render the existing temporal-locality-only RAID-reconstruction optimizations ineffective.

In order to capture both temporal locality and spatial locality to reduce the user I/O requests during reconstruction, we argue that *zone-based*, rather than *request-based*, data popularity identification and data migration schemes should be used. By migrating the "hot" and popular data zones to a surrogate RAID set immediately after a disk fails, it enables the subsequent user I/O requests to be serviced by the surrogate set during reconstruction. This improves the system performance by fully exploiting the spatial locality of the workload. In the meantime, the reconstruction process should rebuild the hot zones, rather than sequentially from the beginning to the end of the failed disk, to take the user I/O requests into consideration. By reconstructing the hot zones first, the data-migration overhead is reduced and most of the subsequent user I/O requests can be serviced by the surrogate set during reconstruction. In so doing,

the reconstruction workflow and the user I/O requests are simultaneously optimized to take the full advantages of both the temporal locality and spatial locality of workloads.

Figure 3 shows an example in which the request-based approach works in a reactive way by migrating the requested data on demand and thus fails to exploit the spatial locality. In contrast, the zone-based approach works in a proactive way by migrating the hot data zones, proactively identified during the normal operational state, to allow subsequent user read requests hit in the migrated data zones serviced by the surrogate set, thus further reducing the user I/O requests to the degraded RAID set during reconstruction.

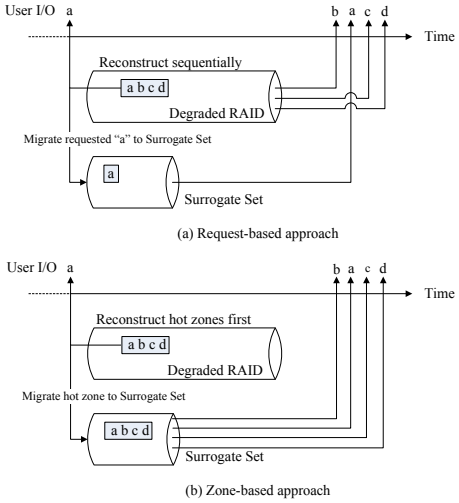


Figure 3: The I/O requests issued to the degraded RAID set with (a) a request-based approach and (b) a zone-based approach.

2.5 IDO motivation

Because user I/O intensity directly affects the RAID-reconstruction performance [28], we plot in Figure 4 the amount of user I/O traffic removed by a reactive request-based optimization (Reactive-request), a reactive zone-based optimization (Reactive-zone), a proactive request-based optimization (Proactive-request) and a proactive zone-based optimization (Proactive-zone), under the three representative traces, WebSearch2.spc, Financial2.spc and Microsoft Project. Each trace is divided into two disjoint parts, one runs in the normal operational state and the other runs in the reconstruction state. The reactive optimization, either request-based or zone-based, exploits the locality of user I/O requests in the reconstruction state and migrates the popular requests or zones to a surrogate set to allow the subsequent repeated read requests to be serviced by the surrogate set. The proactive scheme, either request-based or zone-based, exploits locality of user I/O requests by identifying the popular requests or hot zones in the normal operational state and migrating the hot requests or hot data zones to a surrogate set immediately after a disk fails, allowing any subsequent read requests that hit these migrated hot requests or zones to be serviced by the surrogate set during reconstruction.

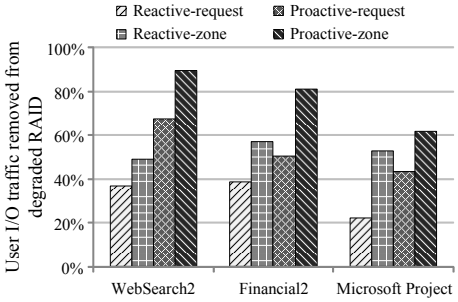


Figure 4: A comparison of the user I/O traffic removed from the degraded RAID set by a reactive request-based optimization (Reactive-request), a reactive zone-based optimization (Reactive-zone), a proactive request-based optimization (Proactive-request) and a proactive zone-based optimization (Proactive-zone), driven by three representative traces.

From Figure 4, we can see that the reactive-request scheme only exploits the data temporal locality in the reconstruction state, thus failing to remove a significant amount of user I/O traffic from the degraded RAID set. For traces with high spatial locality, such as the Microsoft Project trace, the reactive-zone scheme works better than the reactive-request scheme by removing an additional 30.5% of user I/O traffic from the degraded RAID set. For traces with high locality, be it temporal locality or spatial locality, the proactive approach removes much more user I/O traffic than the reactive approach. For example, the proactive-request scheme removes 41.4% and 21.2% more user I/O traffic than the reactive-request scheme for the WebSearch2.spc and Microsoft Project traces, respectively. By combining the proactive and zone approaches, the proactive-zone scheme removes the highest amount of the user I/O traffic from the degraded RAID set, with up to 89.8%, 81.2%, and 61.9% of the user I/O traffic being removed during reconstruction for the WebSearch2.spc, Financials.spc and Microsoft Project traces, respectively.

Clearly, the proactive optimization is much more efficient and effective than its reactive counterpart. Moreover, exploiting both the temporal locality and spatial locality (*i.e.*, zone-based) is better than exploiting only the temporal locality (*i.e.*, request-based), especially for the HDD-based RAIDs in the new HDD/SSD hybrid storage systems. If the hot data zones have been identified in the normal operational state (*i.e.*, without any disk failures)

and the data in these hot zones is migrated to a surrogate set at the beginning of the reconstruction period, the online RAID-reconstruction performance and user I/O performance can be simultaneously significantly improved.

Table 1 compares IDO with state-of-the-art reconstruction optimizations PRO, WorkOut and VDF based on several important RAID reconstruction characteristics. WorkOut [28] and VDF [24] only exploit the temporal locality of workloads to reduce the user I/O requests during reconstruction but ignore the spatial locality. PRO [23] and VDF [24] only focus on optimizing (*i.e.*, tracking or reducing) the user I/O requests to the failed disk but ignore the impact of the user I/O requests to the surviving disks that also have notable performance impact on RAID reconstruction. In contrast, IDO tracks all the user I/O requests addressed to the degraded RAID set in the normal operational state to obtain the data popularity information. Moreover, it exploits both the temporal locality and spatial locality of user I/O requests to both optimize the reconstruction workflow and alleviate the user I/O intensity to the degraded RAID set. Thus, both the RAID reconstruction performance and user I/O performance are simultaneously improved.

Table 1: Comparison of the reconstruction schemes.

Characteristics	PRO [23]	WorkOut [28]	VDF [24]	IDO
Proactive				✓
Temporal Locality	✓	✓	✓	✓
Spatial Locality	✓			✓
User I/O		✓	✓	✓
Reconstruction I/O	✓			✓

3 Intelligent Data Outsourcing

In this section, we first outline the main design objectives of IDO. Then we present its architecture overview and key data structures, followed by a description of the hot data identification, data reconstruction and migration processes. The data consistency issue in IDO is discussed at the end of this section.

3.1 Design objectives

The design of IDO aims to achieve the following three objectives.

- *Accelerating the RAID reconstruction performance* - By removing most of user I/O requests from the degraded RAID set, the RAID reconstruction process can be significantly accelerated.
- *Improving the user I/O performance* - By migrating the data belonging to the proactively identified hot zones to a surrogate RAID set, most subsequent user I/O requests can be serviced by the surrogate RAID set that is not affected by the RAID reconstruction process.
- *Providing high extendibility* - IDO is very simple and can be easily incorporated into the RAID functional module and extended into other background RAID tasks, such as re-synchronization, RAID reshape and disk scrubbing.

3.2 IDO architecture overview

IDO operates beneath the applications and above the RAID systems of a large data center consisting of hundreds or thousands of RAID sets, as shown in Figure 5. There are two types of RAID sets, *working RAID sets* and *surrogate RAID sets*. A working RAID set, upon a disk failure, becomes a *degraded RAID set* and is paired with a surrogate RAID set for the duration of reconstruction. A surrogate RAID set can be a dedicated RAID set that is shared by multiple RAID sets, or a RAID set that is capacity-shared with a lightly-loaded working RAID set. The dedicated surrogate RAID set improves the system performance but introduces extra device overhead, while the capacity-shared surrogate RAID set does not introduce extra device overhead but affects the performance of its own user applications. However, both are feasible and available for system administrators to choose from based on their characteristics and the system requirements. Moreover, the surrogate RAID set can be in the local storage node or a remote storage node connected by a network.

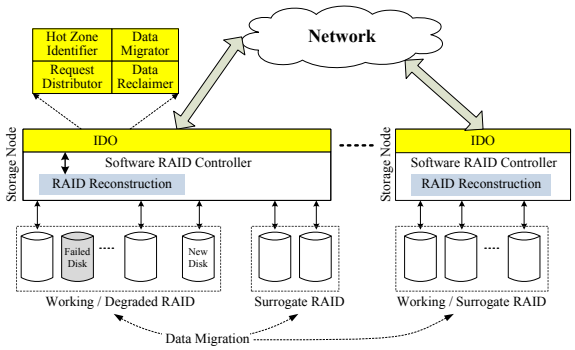


Figure 5: An architecture overview of IDO.

IDO consists of four key functional modules: Hot Zone Identifier, Request Distributor, Data Migrator and Data Reclaimer, as shown in Figure 5. *Hot Zone Identifier* is responsible for identifying the hot data zones in the RAID system based on the incoming user I/O requests. *Request Distributor* is responsible for directing the user I/O requests during reconstruction to the appropriate RAID set, *i.e.*, the degraded RAID set or the surrogate RAID set. *Data Migrator* is responsible for migrating all the data in the hot zones from the degraded RAID set to the surrogate RAID set, while *Data Reclaimer* is responsible for reclaiming all the redirected write data to

the newly recovered RAID set, *i.e.*, the previously degraded RAID set that has completed the reconstruction process. The detailed descriptions of these functional modules are presented in the following subsections.

IDO is an independent module added to an existing RAID system and interacts with its reconstruction module. In the normal operational state, only the Hot Zone Identifier module is active and tracks the popularity of each data zone. The other three modules of IDO remain inactive until the reconstruction module automatically activates them when the reconstruction thread initiates. They are deactivated when the reclaim process completes. The reclaim thread is triggered by the reconstruction module when the reconstruction process completes. IDO can also be incorporated into any RAID software to improve other background RAID tasks. In this paper, we mainly focus on the RAID reconstruction, but do include a short discusstion on how IDO works for some other background RAID tasks. This discusstion can be found in Section 4.4.

3.3 Key data structures

IDO relies on two key data structures to identify the hot data zones and record the redirected write data, namely, *Zone_Table* and *D_Map*, as shown in Figure 6. The Zone_Table contains the popularity information of all data zones, represented by three variables: *Num*, *Popularity* and *Flag*. *Num* indicates the sequence number of the data zone. Based on the Num value and the size of a data zone, IDO can calculate the start offset and end offset of the data zone to determine the target data zone for the incoming read request. *Popularity* indicates the popularity of the data zones. Its value is incremented when a read request hits the corresponding data zone. *Flag* indicates whether the corresponding data zone has been reconstructed and migrated. It is initialized to "00" and used by the Request Distributor module during the reconstruction period. The different values of Flag represent different states of the corresponding data zones, as shown in Figure 6.

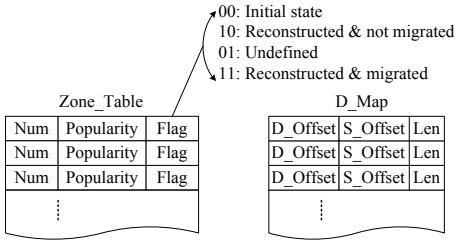


Figure 6: Main data structures of IDO.

The user read requests addressed to the data zones already migrated to the surrogate RAID set are redirected to it to reduce the user I/O traffic to the degraded RAID set. Besides that, IDO also redirects all user write requests to the surrogate RAID set to further alleviate the I/O intensity on the degraded RAID set. The D_Map records the information of all the redirected write data, including the following three variables. *D_Offset* and *S_Offset* indicate the offsets of the redirected write data on the degraded RAID set and the surrogate RAID set, respectively. *Len* indicates the length of the redirected write data. Similar to WorkOut [28], the redirected write data is sequentially stored on the surrogate RAID set to accelerate the write performance. Moreover, the redirected write data is only temporarily stored on the surrogate RAID set and thus should be reclaimed to the newly recovered RAID set after the reconstruction completes.

3.4 Hot data identification

IDO uses a dynamic hot data identification scheme, implemented in the Hot Zone Identifier module, to exploit both temporal locality and spatial locality of the user I/O requests. Figure 7 shows the hot data identification scheme in IDO. First, the entire RAID device space is split into multiple equal-size data zones of multiple-stripe size each. For example, in a 4-disk RAID5 set with a 64KB chunk size (*i.e.*, a stripe size of 3× 64KB = 192KB), the size of a data zone should be multiple times of 192KB. Therefore, a data zone is stripe aligned and can be fetched together to reconstruct the lost data blocks. Moreover, *spatial locality of requests* is also exploited since the tracking and migration unit of data is a data zone, thus IDO can capture the spatial locality of workloads by migrating data blocks prior to arrival of user I/O requests for those blocks. A detailed evaluation based on the selection of the different data zone sizes is presented in Section 4.3.

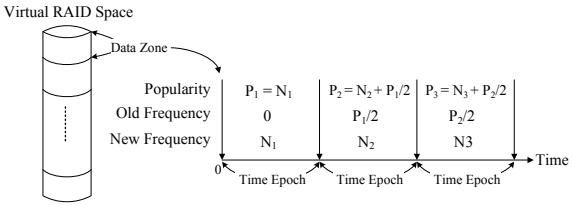


Figure 7: Hot data identification scheme in IDO. Note that the frequency of N_* is defined as the number of user I/O requests issued to the data zone in a time epoch.

Second, in order to effectively and accurately exploit *temporal locality of requests*, the hot data zones in IDO are aged by decreasing their popularity values with time. More specifically, the popularity of a data zone in the current time epoch is calculated by first halving its popularity value from the previous time epoch before adding any values to it as more requests hit the zone in the current epoch. For example, as shown in Figure 7, P_2 is

equal to N_2 plus half of P_1 that is the popularity of the same data zone in its former time epoch. When a time epoch ends, the popularities of all data zones are halved in value.

Third, in order to reduce the impact of the hot data identification scheme on the system performance in the normal operational state, IDO updates the Zone_Table in the main memory without any disk I/O operations. When a read request arrives, IDO first checks its offset to locate the data zone that it belongs to. Then, IDO increments the corresponding *Popularity* in the Zone_Table in the main memory. Since the delay of the memory processing is much smaller than that of the disk I/O operation, the identification process in IDO has little performance impact on the overall system performance. Moreover, with the increasing processing power embedded in the storage controller, some storage systems already implement intelligent modules to identify the data popularity. Consequently, IDO can also simply utilize these functionalities to identify the hot data zones.

3.5 Data reconstruction and migration

Figure 8 shows the data reconstruction and migration processes in IDO. When a disk fails, the RAID reconstruction process is triggered with a hot spare disk in the degraded RAID set. IDO first reconstructs data in the hot data zones on the failed disk according to the Zone_Table. When the data blocks in the hot zones are read, IDO reconstructs and concurrently migrates all the data in these hot zones to the surrogate RAID set. Because the data is sequentially written on the surrogate RAID set, the overhead of the data migration, *i.e.*, writing the hot data to the surrogate RAID set, is minimal in IDO. When a hot data zone has been reconstructed and its data migrated to the surrogate RAID set, the corresponding *Flag* in the Zone_Table is set to "11". After all the hot data zones have been reconstructed, IDO begins to reconstruct the remaining data zones. In order to reduce the space overhead on the surrogate RAID set, IDO does not migrate the data in the cold data zones to the surrogate RAID set. Moreover, migrating the cold data does little to improve the overall system performance since few subsequent user I/O requests will be issued to these cold data zones. After a cold data zone has been reconstructed, the corresponding *Flag* in the Zone_Table is set to "10".

When all the data zones have been reconstructed, that is, the RAID reconstruction process is completed, the reclaim process for the redirected write data is initiated. In IDO, the redirected write data on the surrogate RAID set is protected by a redundancy scheme, such as RAID1 or RAID5/6. The priority of the reclaim process is set to be lower than the user I/O requests, which will not affect

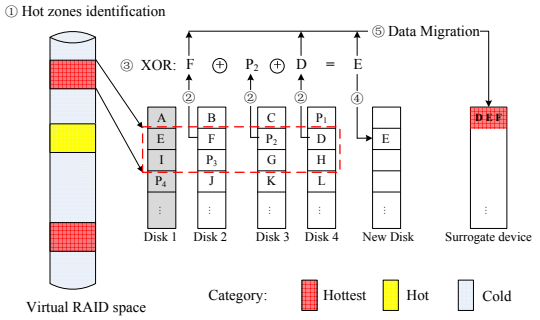


Figure 8: Data reconstruction and migration in IDO.

the reliability of the RAID system [28]. Therefore, the reclaim process for the redirected write data can also be scheduled in the system idle period. When a redirected write data block is reclaimed, its corresponding item in the D_Map is deleted. After all the items in the D_Map are deleted, the reclaim process completes.

During on-line RAID reconstruction, all incoming user I/O requests are carefully checked. Upon the arrival of a read request, IDO first determines its target data zone according to the Zone_Table and checks the second bit of the corresponding *Flag* to determine whether the data zone has been migrated or not ("1" indicates that the data zone has been migrated, while "0" indicates the opposite). If the data zone has not been migrated, the read request is issued to the degraded RAID set and the *Popularity* of the corresponding data zone is updated. Otherwise, the read request is issued to the surrogate RAID set. In order to obtain the accurate location on the surrogate RAID set, IDO checks the D_Map to determine whether the read request hits the previously redirected write data. If so, the read request is issued to the surrogate RAID set according to the *S_Offset* in the D_Map. Otherwise, the read request is issued to the surrogate RAID set according to the Zone_Table.

When processing a write request, the write data is sequentially written on the surrogate RAID set and IDO checks whether the write request hits the D_Map. If the write request hits the D_Map, the corresponding item in the D_Map is updated. Otherwise, a new item for the write request is added to the D_Map.

3.6 Data consistency

Data consistency in IDO includes two aspects: (1) The key data structures must be safely stored, (2) The redirected write data must be reliably stored on the surrogate set until the data reclaim process completes.

First, to prevent the loss of the key data structures in the event of a power supply failure or a system crash, IDO stores them in a non-volatile RAM (NVRAM). Since the size of Zone_Table and D_Map is generally very small, it will not incur significant extra hardware

cost. Moreover, in order to improve the write performance by using the write-back technique, the NVRAM is commonly deployed in the storage controllers. Thus, it is easy and reasonable to use the NVRAM to store the key data structures.

Second, the redirected write data must be safely stored on the surrogate set. To prevent data loss caused by a disk failure on the surrogate set, the surrogate set must be protected by a redundancy scheme, such as mirroring-based (RAID1) or parity-based (RAID5/6) disk arrays, a basic requirement for the surrogate RAID set. Our previous study [28] provides a detailed analysis on how to choose a surrogate set based on the requirements and characteristics of the applications. Moreover, since the up-to-date data for a read request can be stored on either the degraded RAID set or the surrogate set, each read request is first checked in the D_Map to determine whether it should be serviced by the degraded RAID set, the surrogate set or both (when the data is partially modified) to keep the fetched data always up-to-date, until all the redirected write data has been reclaimed.

4 Performance Evaluation

In this section, we present the performance evaluation of the IDO prototype through extensive trace-driven experiments.

4.1 Experimental setup and methodology

We have implemented an IDO prototype by embedding it into the Linux software RAID (MD) as a built-in module. IDO tracks the user I/O requests in the *make_request* function to identify the data popularity in the normal operational state. When a disk fails and the reconstruction thread is initiated by the *md_do_sync* function, the hot data zones are first reconstructed and migrated to the surrogate set. During reconstruction, the incoming user read requests are checked in the *make_request* function to determine by which device the requests are to be serviced, so as to avoid the degraded set whenever possible. All user write requests are issued to the surrogate set and marked as dirty for reclaim after the RAID reconstruction process completes.

The performance evaluation of IDO was conducted on a server-class hardware platform with an Intel Xeon X3440 processor and 8GB DDR memory. The HDDs are WDC WD1600AAJS SATA disks that were used to configure both the active RAID set and the surrogate RAID set. While the active set assumed a RAID5/6 organization, the surrogate set was configured as a RAID1 organization with 2 HDDs. Further, the surrogate set can be located either in the same storage node as the degraded RAID set or in a remote storage node in a data center. The rotational speed of these disks is 7200 RPM, with a sustained transfer rate of 60MB/s that is specified in the manufacture's datasheet. We used 10GB of the capacity of each disk for the experiments. A separate disk was used to house the operating system (Linux kernel version 2.6.35) and other software (MD and mdadm). In our prototype implementation, the main memory was used to substitute a battery-backed RAM for simplicity.

The traces used in our experiments were obtained from the UMass Trace Repository [15] and Microsoft [14]. The two financial traces (short for Fin1 and Fin2) were collected from the OLTP applications running at a large financial institution and the WebSearch2 trace (short for Web2) was collected from a machine running a web search engine. The Microsoft Project trace was collected in a volume storing the project directories (short for Proj). The four traces represent different access patterns in terms of read/write ratio, IOPS and average request size, with the main workload parameters summarized in Table 2.

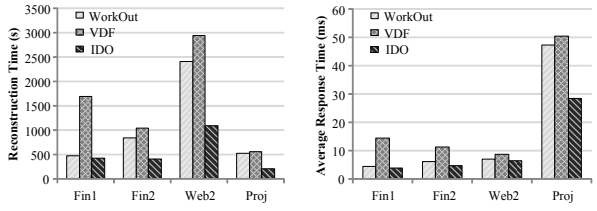
Table 2: The key evaluation workload parameters.

Trace	Trace Characteristic		
	Read Ratio	IOPS	Aver. Req. Size(KB)
Fin1	32.8%	69	6.2
Fin2	82.4%	125	2.2
Web2	100%	113	15.1
Proj	97.6%	29	57.8

To better examine the IDO performance under existing RAID reconstruction approaches, we incorporated IDO into MD's default reconstruction algorithm PR. We compared IDO with two state-of-the-art RAID reconstruction optimizations, WorkOut [28] and VDF [24], in terms of reconstruction performance and user I/O performance. WorkOut tracks the user access popularity and issues all write requests and popular read requests to the surrogate set during reconstruction. VDF exploits the fact that the user I/O requests addressed to the failed disk are expensive by keeping the requested data previously stored on the failed disk longer in the storage cache and choosing data blocks belonging to the surviving disks to evict first. Because VDF is a cache replacement algorithm, we applied it to the management of the surrogate set to make the comparison fair.

4.2 Performance results

We first conducted experiments on a 4-disk RAID5 set with a stripe unit size of 64KB while running WorkOut, VDF and IDO, respectively. Figure 9 shows the reconstruction time and average user response time under the minimum reconstruction bandwidth of 1MB/s, driven by the four traces. We configured a local 2-disk dedicated RAID1 set as the surrogate set to boost the reconstruction performance of the 4-disk degraded RAID5 set. For IDO, the data zone size was set to 12MB.



(a) Reconstruction Time (b) Average Response Time

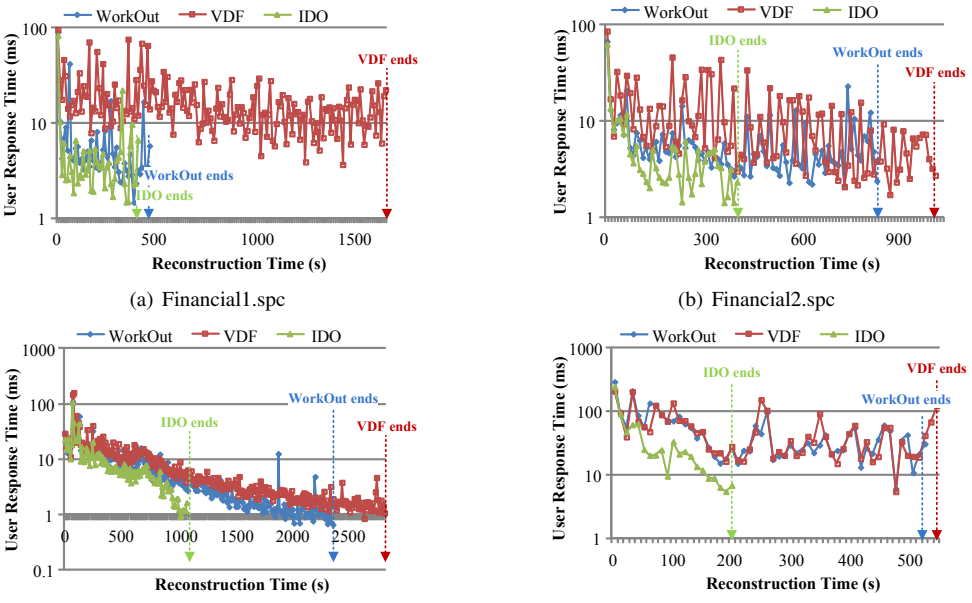
Figure 9: The performance results of WorkOut, VDF and IDO in a 4-disk RAID5 set with a stripe unit size of 64KB, 1MB/s minimum reconstruction bandwidth, and a local 2-disk dedicated RAID1 set as the surrogate RAID set, driven by the four traces.

From Figure 9(a), we can see that IDO speeds up WorkOut by a factor of 1.1, 2.1, 2.2 and 2.6, and speeds up VDF by a factor of 4.1, 2.5, 2.7 and 2.7 in terms of the reconstruction time for the Fin1, Fin2, Web2 and Proj traces, respectively. IDO's advantage stems from its ability to remove much more user I/O requests from the degraded RAID set than WorkOut and VDF, as indicated in Figure 11, which enables it to accelerate the RAID reconstruction process. However, since the Fin1 trace has much more write requests than read requests (as indicated in Table 2), IDO and WorkOut have similar abilities to remove the write requests from the degraded RAID set, reducing IDO's performance advantage over WorkOut. For the read-intensive traces, Fin2, Web2 and Proj, IDO removes much more read requests from the degraded RAID set than WorkOut. This is because IDO proactively identifies both the temporal locality and spatial locality in the normal operational state and migrates the hot data zones at the onset of reconstruction, while WorkOut only reactively identifies the temporal locality and migrates the user I/O requests after a disk fails. In this case, most subsequent read requests addressed to these hot data zones in IDO can be serviced directly by the surrogate RAID set instead of the much slower degraded RAID set. As a result, IDO's advantage margin over WorkOut is much wider under these three read-intensive traces than under the write-intensive Fin1 trace. For example, IDO reduces the reconstruction time much more significantly than WorkOut under the Proj trace because the Proj trace has poor temporal locality that leaves WorkOut much less room to improve than the spatial-locality-exploiting IDO.

On the other hand, from Figure 9(a), we can see that the performance of both WorkOut and IDO are better than that of VDF. The reason is that both WorkOut and IDO reduce not only the user I/O requests to the failed disk, but also the popular read requests and all write requests to the surviving disks. VDF keeps the data blocks belonging to the failed disk longer in the storage cache because servicing these data blocks is much more expensive than servicing the data blocks belonging to the surviving disks. However, if the data blocks belonging to the failed disk are already reconstructed, they will behave exactly the same way as the data blocks belonging to the surviving disks because the read redirection technique will fetch these data blocks directly from the replacement disk rather than reconstructing them from the surviving disks again. In the VDF evaluation reported in [24], the experimental kernel is Linux 2.6.32 without the read redirection function that has been implemented in the Linux MD software since Linux 2.6.35 [13]. The results also confirm the conclusion made in our previous WorkOut study that the user I/O intensity has a significant impact on the on-line RAID reconstruction performance.

Figure 9(b) shows that IDO speeds up WorkOut by a factor of 1.1, 1.3, 1.1 and 1.7, and speeds up VDF by a factor of 3.7, 2.4, 1.4 and 1.8 in terms of the average user response time (i.e., user I/O performance) during reconstruction for the Fin1, Fin2, Web2 and Proj traces, respectively. The reason why IDO only improves WorkOut slightly is that the minimum reconstruction bandwidth is set to be the default 1MB/s, which gives the user I/O requests a higher priority than the reconstruction requests. However, the performances of both IDO and WorkOut are better than that of VDF because IDO and WorkOut remove much more user I/O requests from the degraded RAID set than VDF, as revealed in Figure 11. Removing the user I/O requests from the degraded RAID set directly improves the user response time for the following two reasons. First, the response time of the redirected requests is no longer affected by the reconstruction process that competes for the available disk bandwidth with the user I/O requests on the degraded RAID set. Moreover, the redirected write data is sequentially laid out on the surrogate RAID set, thus further reducing the user response time. Second, many user I/O requests are removed from the degraded RAID set and the I/O queue on the degraded RAID set is accordingly shortened, thus reducing the response time of the remaining user I/O requests still serviced by the degraded RAID set.

Figure 10 compares in more details the reconstruction and use I/O performances of IDO, WorkOut and VDF during the reconstruction process, highlighting the significant advantage of IDO over WorkOut and VDF. Two aspects of the significant improvement in the user response time are demonstrated in these figures. First, the onset of the performance improvement of IDO is much earlier than that of WorkOut and VDF during reconstruction. The reason is that IDO has already captured the data popularity before a disk fails, thus enabling it to optimize the reconstruction process earlier and consequently outperform both WorkOut and VDF in terms of user response time during reconstruction. Second, IDO



(a) Financial1.spc

(b) Financial2.spc

(c) WebSearch2.spc

(d) MSR Project

Figure 10: A comparison of user response times of WorkOut, VDF and IDO during reconstruction in a 4-disk RAID5 set with a stripe unit size of 64KB, 1MB/s minimum reconstruction bandwidth, and a local 2-disk dedicated RAID1 set as the surrogate RAID set, driven by the four traces.

completes the reconstruction process much more quickly than WorkOut and VDF, which translates into improved reliability. The RAID reconstruction period is also called a "window of vulnerability" during which a subsequent disk failure (or a series of subsequent disk failures) will result in data loss [23]. Thus, shorter reconstruction time indicates higher reliability. Furthermore, user I/O requests in the IDO reconstruction experience a much shorter period of time in which they see increased response time than in the WorkOut and VDF reconstruction.

To gain a better understanding of the reasons behind the significant improvement achieved by IDO, we plotted the percentage of redirected requests for the three schemes under the minimum reconstruction bandwidth of 1MB/s. From Figure 11, we can see that IDO moves 88.1%, 78.7%, 72.4% and 42.0% of user I/O requests from the degraded RAID set to the surrogate RAID set for the four traces respectively, significantly more than either WorkOut or VDF does. This is because both WorkOut and VDF only exploit the temporal locality of user I/O requests and VDF only gives higher priority to the user I/O requests belonging to the failed disk. In contrast, IDO exploits both the temporal locality and spatial locality on all disks, reconstructs the hot data zones first and migrates them to the surrogate RAID set. And, most importantly, IDO uses a proactive optimization that is superior to the reactive optimizations, such as WorkOut and VDF. On the other hand, removing user I/O requests from the degraded RAID set directly reduces both the reconstruction time and user response time [28], something that IDO does much better than either WorkOut or VDF.

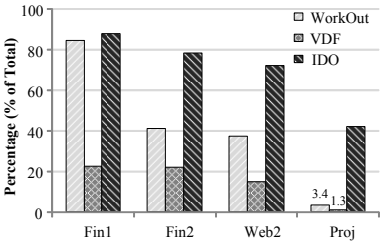
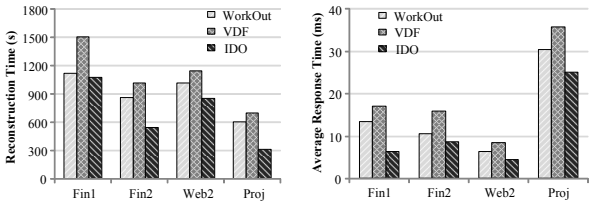


Figure 11: Percentage of redirected user I/O requests for WorkOut, VDF and IDO under the minimum reconstruction bandwidth of 1MB/s.

We also conducted experiments on a 6-disk RAID6 set (4 data + 2 parity) with a stripe unit size of 64KB under the minimum reconstruction bandwidth of 1MB/s. In the RAID6 experiments, we configured a 2-disk dedicated RAID1 set in the same storage node as the local surrogate set. In the experiments, we measured the reconstruction times and the average user response times when one disk fails.

From Figure 12, we can see that IDO improves both the reconstruction times and the average user response times over the WorkOut and VDF schemes. In particular, IDO speeds up WorkOut by a factor of up to 1.9 with an average of 1.4 in terms of the reconstruction time, and by a factor of up to 2.1 with an average of 1.5 in terms of the average user response time. IDO speeds up VDF

by a factor of up to 2.2 with an average of 1.7 in terms of the reconstruction time, and by a factor of up to 2.7 with an average of 2.0 in terms of the average user response time.



(a) Reconstruction Time (b) Average Response Time

Figure 12: Reconstruction times and average user response times of RAID6 reconstruction.

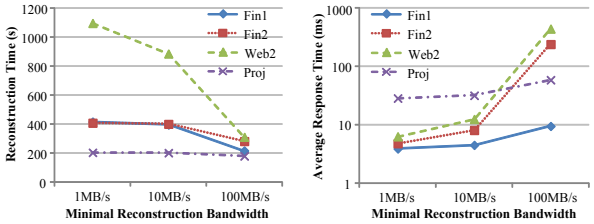
The reason behind the improvement on RAID6 is similar to that for RAID5. Upon a disk failure, all the disks in RAID6, as in RAID5, will be involved in servicing the reconstruction I/O requests. In the meantime, all the disks will service the user I/O requests. Thus, removing the user I/O requests can directly speed up the reconstruction process by allowing much more disk bandwidth to service the reconstruction I/O requests. Moreover, the average user response times also decrease because most of the user I/O requests are serviced on the surrogate RAID set without interfering with the reconstruction I/O requests. IDO works in a proactive way and exploits both the temporal locality and spatial locality of the user I/O requests, thus removing much more user I/O requests from the degraded RAID set to the surrogate set (as similarly indicated in the Figure 11) and performing better than Workout and VDF.

4.3 Sensitivity study

The IDO performance is likely influenced by several important factors, including the available reconstruction bandwidth, the data zone size, the stripe unit size, the number of disks, and the location of the surrogate set.

Reconstruction bandwidth. To evaluate how the minimum reconstruction bandwidth affects the reconstruction performance, we conducted experiments to measure reconstruction time and average user response time as a function of different minimum reconstruction bandwidths, 1MB/s, 10MB/s and 100MB/s, respectively. Figure 13 plotted the experimental results on a 4-disk RAID5 set with a stripe unit size of 64KB and a data zone size of 12MB.

From Figure 13(a), we can see that the reconstruction time generally decreases with the increasing minimum reconstruction bandwidth. However, for the Fin1, Fin2 and Proj traces, the reconstruction times remain almost unchanged when the minimum reconstruction bandwidth changes from 1MB/s to 10MB/s. The reason is that when the minimum reconstruction bandwidth is 1MB/s, the actual reconstruction speed is around 10MB/s due to the low user I/O intensity on the degraded RAID set. In contrast, From Figure 13(b), we can see that the user response time increases rapidly with the increasing minimum reconstruction bandwidth. When the minimum reconstruction bandwidth increases, much more reconstruction I/O requests are issued, thus lengthening the disk I/O queue and increasing the user I/O response time.



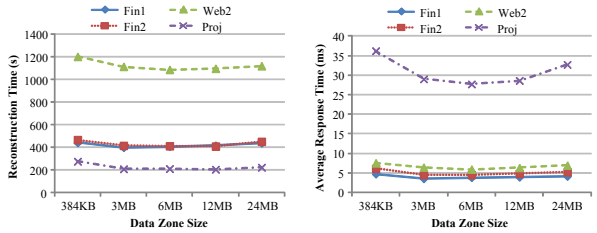
(a) Reconstruction Time (b) Average Response Time

Figure 13: Reconstruction times and average user response times of IDO as a function of different minimum reconstruction bandwidths (1MB/s, 10MB/s and 100MB/s) under the four traces.

Data zone size. In IDO, the data zone size is a key factor in identifying the data popularity. In order to evaluate the effect of the data zone size on the reconstruction performance and user I/O performance during reconstruction, we conducted experiments on a 4-disk RAID5 set with a stripe unit size of 64KB and different data zone sizes of 384KB, 3MB, 6MB, 12MB and 24MB, under the minimal reconstruction bandwidth of 1MB/s.

The results, shown in Figure 14, indicate that, with a very small data zone in one extreme, both the reconstruction time and the user response time are increased. The reason is that the reconstruction sequentiality is destroyed with a very small data zone. Moreover, the spatial locality is somewhat weakened with a very small zone size, although highly concentrated temporal locality is still captured. In the other extreme, with a very large data zone, the reconstruction performance and the user response time are again both increased. The reason is that with a very large data zone, IDO will likely migrate much more rarely-accessed cold data blocks to the surrogate set, thus wasting the disk bandwidth resources. Our experiments, driven by the four traces, seem to suggest that the data zone sizes between 3MB to 12MB are consistently the best.

Stripe unit size. To examine the impact of the stripe unit size on the RAID reconstruction, we conducted experiments on a 4-disk RAID5 set with stripe unit sizes of 4KB, 16KB and 64KB, respectively. The experimental results show that the reconstruction times and user response times are almost unchanged, suggesting that IDO

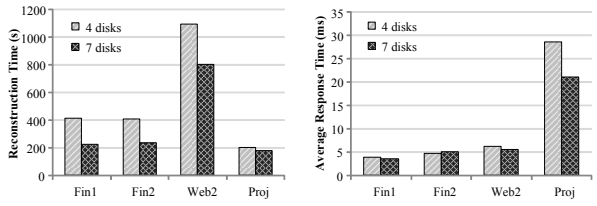


(a) Reconstruction Time (b) Average Response Time

Figure 14: Reconstruction times and average user response times of IDO as a function of different data zone size (384KB, 3MB, 6MB, 12MB and 24MB) under the four traces.

is not sensitive to the stripe unit size. The reason is that most user I/O requests are performed on the surrogate RAID set, thus the stripe unit size of the degraded RAID set has no impact on these redirected user I/O requests and very little impact overall. Accordantly, the RAID reconstruction speed is not affected. Due to space limits, these results are not shown here quantitatively.

Number of disks. To examine the sensitivity of IDO to the number of disks of the degraded RAID set, we conducted experiments on RAID5 sets consisting of different numbers of disks (4 and 7) with a stripe unit size of 64KB under the minimum reconstruction bandwidth of 1MB/s. From Figure 15, we can see that the reconstruction time decreases with the increasing number of disks. The reason is that the I/O intensity on individual disks will decrease when the RAID set has more disks, allowing for a shorter reconstruction time. However, the user I/O requests are not significantly affected since they are mostly serviced by the surrogate RAID set. The remaining user I/O requests still performed on the degraded RAID set only slightly affect the total user response time.

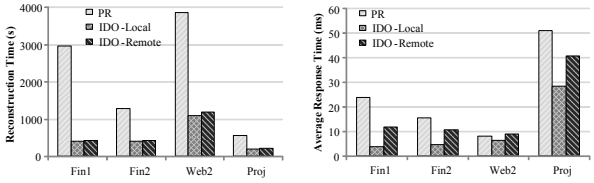


(a) Reconstruction Time (b) Average Response Time

Figure 15: Reconstruction times and average user response times of IDO as a function of different numbers of disks (4 and 7) under the four traces.

Location of the surrogate device. In a large-scale data center, the location of the surrogate set can also affect the RAID reconstruction performance. To examine the impact of the location of the surrogate set on the reconstruction performance, we conducted experiments to migrate the hot data to a different storage node connected with a gigabit Ethernet interface in a local area network.

Figure 16(a) shows that the reconstruction time is almost unchanged, which indicates that the reconstruction time is not sensitive to the location of the surrogate set. The reason is that no matter where the surrogate set is, the total numbers of redirected user I/O requests are the same, thus the reconstruction speed of the degraded RAID set is similar. On the other hand, the average user response time increases significantly with a remote surrogate set, as shown in Figure 16(b). The reason is that the response time of the redirected user I/O requests must now include the extra network delay with a remote surrogate set. However, compared with PR (the default reconstruction algorithm of Linux MD), IDO still significantly reduces the user response time and the reconstruction time with a remote surrogate set. The results suggest that in a large-scale data center, both the local and remote surrogate devices are helpful in improving the reconstruction performance, which further validates the effectiveness and adaptivity of IDO in large-scale data centers.

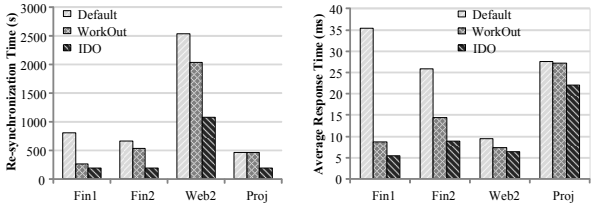


(a) Reconstruction Time (b) Average Response Time

Figure 16: Reconstruction times and average user response times with respect to different surrogate set locations under the four traces.

4.4 Extensibility

To demonstrate how IDO may be extended to optimize other background RAID tasks, we incorporated IDO into the RAID re-synchronization module. We conducted experiments on a 4-disk RAID5 set with a stripe unit size of 64KB under the minimum re-synchronization bandwidth of 1MB/s, driven by the four traces. We configured a dedicated 2-disk local RAID1 set as the surrogate set. The experimental results of the re-synchronization times and average user response times during re-synchronization are shown in Figure 17.



(a) Re-synchronization Time (b) Average Response Time

Figure 17: The re-synchronization results of the default re-synchronization function in the Linux MD software without any optimization, WorkOut and IDO.

Although the RAID re-synchronization process operates somewhat differently than the RAID reconstruction process, the re-synchronization requests compete for the disk resources with the user I/O requests during the on-line re-synchronization period in a way similar to the latter. By redirecting a significant amount of user I/O requests away from the RAID set undergoing re-synchronization, IDO can reduce both the re-synchronization time and user response time. The results are very similar to those in the above RAID reconstruction experiments, as are the reasons behind them.

4.5 Overhead analysis

Besides the device overhead for the surrogate set in the case of dedicated surrogate sets [28], we analyzed the following two overhead metrics in this paper: the performance overhead in the normal operational state and the memory overhead.

Performance overhead. Since IDO is a proactive optimization designed to improve the RAID reconstruction performance, it requires a hot data identification module that may affect the system performance in the normal operational state. In order to quantify how much the impact is, we conducted experiments to evaluate the user response times in the normal operational state with and without the hot data identification module.

From the experimental results, we find that the user response times in the two cases remain roughly unchanged under any of the four traces. In the worst case, the performance with the module activated degrades by less than 3% under the WebSearcch2.spc trace. The reason is that the hot data identification module only adds one extra operation (*i.e.*, incrementing the in-memory popularity value of the corresponding data zone by one) for each request. Thus the performance overhead is negligible in face of the high latency of the disk accesses.

Memory overhead. To prevent data loss, IDO uses non-volatile memory to store the Zone_Table and D_Map, thus incurring extra memory overhead. However, IDO uses less non-volatile memory capacity than WorkOut. The reason is that WorkOut migrates the user requested data to the surrogate set while IDO migrates the hot data zones. In WorkOut, each migrated write request and read request requires a corresponding entry in the mapping table. However, in IDO, only the migrated write requests and hot data zones require corresponding entries in the mapping information. Since the number of migrated hot data zones in IDO is generally much smaller than the number of hot read requests in WorkOut, the memory overhead of IDO is lower than that of WorkOut.

In the above experiments on the RAID5 set with individual disk capacity of 10GB, the maximum memory overheads are 0.11MB, 0.24MB, 0.11MB and 0.06MB for the Fin1, Fin2, Web2 and Proj traces, respectively. With the rapid increase in the size of memory and decrease in the cost of non-volatile memories, this memory overhead of IDO is arguably reasonable and acceptable to the end users.

5 Conclusion

In many data-intensive computing environments, especially data centers, large numbers of disks are organized into various RAID architectures. Because of the increased error rates for individual disk drives, the dramatically increasing size of drives, and the slow growth in transfer rates, the performance of RAID during its reconstruction phase (after a disk failure) has become increasingly important for system availability. We have shown that IDO can substantially improve this performance at low cost by using the free space available in these environments. IDO proactively exploits both the temporal locality and spatial locality of user I/O requests to identify the hot data zones in the normal operational state. When a disk fails, IDO first reconstructs the lost data blocks on the failed disk belonging to the hot data zones and concurrently migrates them to a surrogate RAID set. This enables most subsequent user I/O requests to be serviced by the surrogate RAID set, thus improving both the reconstruction performance and user I/O performance.

IDO is an ongoing research project and we are currently exploring several directions for future research. One possible direction is to design and conduct more experiments to evaluate the IDO prototype for other background tasks (such as RAID reshape and disk scrubbing) and other RAID levels (such as RAID10). Another is to build a power measurement module to evaluate the energy efficiency of IDO. By proactively migrating the hot data to the active RAID sets, the local RAID set can stay longer in the idle mode to save energy by redirecting read requests and write requests to the active RAID sets.

Acknowledgments

We thank our shepherd Andrew Hume, our mentor Nicole Forsgren Velasquez, and the anonymous reviewers for their helpful comments. This work is supported by the National Science Foundation of China under Grant No. 61100033, the US National Science Foundation under Grant No. NSF-CNS-1116606, NSF-CNS-1016609, NSF-IIS-0916859, and NSF-CCF-0937993.

References

[1] R. Arnan, E. Bachmat, T. Lam, and R. Michel. Dynamic Data Reallocation in Disk Arrays. *ACM-Transactions on Storage*, 3(1):2, 2007.

[2] Storage at Exascale: Some Thoughts from Panasas CTO Garth Gibson. Interview. http://www.hpcwire.com/hpcwire/2011-05-25/storage_at_exascale_some_thoughts_from_panasas_cto_garth_gibson.html. May. 2011.

[3] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *SIGMETRICS'07*, Jun. 2007.

[4] F. Chen, David A. Koufaty, and X. Zhang. Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems. In *ICS'11*, Jun. 2011.

[5] Veera Deenadhayalan. GPFS Native RAID for 100,000-Disk Petascale Systems. In *LISA'11*, Dec. 2011.

[6] G. Gibson. Reflections on Failure in Post-Terascale Parallel Computing. Keynote. In *ICPP'07*, Sep. 2007.

[7] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Fourth edition, 2006.

[8] M. Holland. *On-Line Data Reconstruction in Redundant Disk Arrays*. PhD thesis, Carnegie Mellon University, Apr. 1994.

[9] M. Holland, G. Gibson, and D. P. Siewiorek. Architectures and Algorithms for On-Line Failure Recovery in Redundant Disk Arrays. *Journal of Distributed and Parallel Databases*, 2(3):295–335, Jul. 1994.

[10] IBM Easy Tier. http://www.almaden.ibm.com/storagesystems/projects/easytier.

[11] S. Kang and A. Reddy. User-Centric Data Migration in Networked Storage Systems. In *IPDPS'08*, Apr. 2008.

[12] J. Lee and J. Lui. Automatic Recovery from Disk Failure in Continuous-Media Servers. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):499–515, May. 2002.

[13] [MD PATCH 09/16] md/raid5: preferentially read from replacement device if possible. http://www.spinics.net/lists/raid/msg36361.html.

[14] D. Narayanan, A. Donnelly, and A. Rowstron. Write Off-Loading: Practical Power Management for Enterprise Storage. In *FAST'08*, Feb. 2008.

[15] OLTP Application I/O and Search Engine I/O. http://traces.cs.umass.edu/index.php/Storage/Storage.

[16] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD'88*, Jun. 1988.

[17] E. Pinheiro and R. Bianchini. Energy Conservation Techniques for Disk Array-Based Servers. In *ICS'04*, Jun. 2004.

[18] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure Trends in a Large Disk Drive Population. In *FAST'07*, Feb. 2007.

[19] M. Saxena and Michael M. Swift. FlashVM: Virtual Memory Management on Flash. In *USENIX ATC'10*, Jun. 2010.

[20] B. Schroeder and G. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *FAST'07*, Feb. 2007.

[21] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving Storage System Availability with D-GRAID. In *FAST'04*, Mar. 2004.

[22] L. Tian, Q. Cao, H. Jiang, D. Feng, C. Xie, and Q. Xin. SPA: On-Line Availability Upgrades for Parity-based RAIDs through Supplementary Parity Augmentations. *ACM Transactions on Storage*, 6(4), 2011.

[23] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: A Popularity-based Multi-threaded Reconstruction Optimization for RAID-Structured Storage Systems. In *FAST'07*, Feb. 2007.

[24] S. Wan, Q. Cao, J. Huang, S. Li, X. Li, S. Zhan, L. Yu, C. Xie, and X. He. Victim Disk First: An Asymmetric Cache to Boost the Performance of Disk Arrays under Faulty Conditions. In *USENIX ATC'11*, Jun. 2011.

[25] C. Weddle, M. Oldham, J. Qian, A. Wang, P. Reiher, and G. Kuenning. PARAID: A Gear-Shifting Power-Aware RAID. In *FAST'07*, Feb. 2007.

[26] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable Performance of the Panasas Parallel File System. In *FAST'08*, Feb. 2008.

[27] S. Wu, D. Feng, H. Jiang, B. Mao, L. Zeng, and J. Chen. JOR: A Journal-guided Reconstruction Optimization for RAID-Structured Storage Systems. In *ICPADS'09*, Dec. 2009.

[28] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. WorkOut: I/O Workload Outsourcing for Boosting RAID Reconstruction Performance. In *FAST'09*, Feb. 2009.

[29] S. Wu, B. Mao, D. Feng, and J. Chen. Availability-Aware Cache Management with Improved RAID Reconstruction Performance. In *CSE'10*, Dec. 2010.

[30] T. Xie and H. Wang. MICRO: A Multilevel Caching-Based Reconstruction Optimization for Mobile Storage Systems. *IEEE Transactions on Computers*, 57(10):1386–1398, 2008.

[31] Q. Xin, E. L. Miller, and T. J. E. Schwarz. Evaluation of Distributed Recovery in Large-Scale Storage Systems. In *HPDC'04*, Jun. 2004.

Theia: Visual Signatures for Problem Diagnosis in Large Hadoop Clusters

Elmer Garduno, Soila P. Kavulya, Jiaqi Tan, Rajeev Gandhi, Priya Narasimhan
Carnegie Mellon University

Abstract

Diagnosing performance problems in large distributed systems can be daunting as the copious volume of monitoring information available can obscure the root-cause of the problem. Automated diagnosis tools help narrow down the possible root-causes—however, these tools are not perfect thereby motivating the need for visualization tools that allow users to explore their data and gain insight on the root-cause. In this paper we describe Theia, a visualization tool that analyzes application-level logs in a Hadoop cluster, and generates visual signatures of each job's performance. These visual signatures provide compact representations of task durations, task status, and data consumption by jobs. We demonstrate the utility of Theia on real incidents experienced by users on a production Hadoop cluster.

1 Introduction

Hadoop [29] is an open-source implementation of Google's MapReduce [9] framework. As of 2012, a typical Hadoop deployment consisted of tens to thousands of nodes [30, 31]. Manual diagnosis of performance problems in a Hadoop cluster requires users to comb through the logs on each node—a daunting task, even on clusters consisting of tens of nodes. Fortunately, there has been significant research on automated diagnosis in distributed systems; ranging from techniques that generate and analyze end-to-end causal traces [14, 25], to alarm-correlation techniques [13, 20, 16].

In recent years, there has also been an increased interest in developing diagnosis techniques that understand the application-specific semantics of MapReduce jobs, thereby providing users with insight into the behavior of their jobs [8, 28, 2]. However, the use of automated diagnosis techniques in isolation is not always sufficient to localize problems at the level of granularity desired by users. Visualization tools help bridge this gap by providing interactive interfaces that allow users to explore their data, and formulate their own hypothesis about the root-cause of problems. A number of visualization tools focus on visualizing time-series data [17, 18], request flows [26, 10], and the outputs of automated diagnosis algorithms [15].

Our tool, Theia[1], leverages application-specific semantics about the structure of the MapReduce programming model to generate high-density, interactive visualizations of job performance that scale to support current industry deployments. A recent study of users at a production Hadoop cluster [5] highlighted users' need to differentiate application-level problems (e.g., software bugs, workload imbalances) from infrastructural problems (e.g., contention problems, hardware problems). In Theia, we have developed *visual signatures* that allow users to easily spot performance problems due to application-level and infrastructural issues.

We developed three different visualizations: one at the cluster-level that represents the performance of jobs across nodes over time, and two others at the job-level that summarize task performance across nodes in terms of task duration, task status and volume of data processed. We evaluated these visualizations using real problems experienced by Hadoop users at a production cluster over a one-month period. Our visualizations correctly identified 192 out of 204 problems that we observed during that time period.

The rest of the paper is organized as follows. Section 2 describes the Hadoop production cluster, and the challenges of diagnosing problems at scale. Section 3 describes the design and implementation of our visualization tool, Theia. Section 4 describes our visualizations using case studies drawn from real problems experienced by Hadoop users in the cluster. Finally, Section 5 and 6 present related work and conclusions.

[1] Named after the Greek goddess of light.

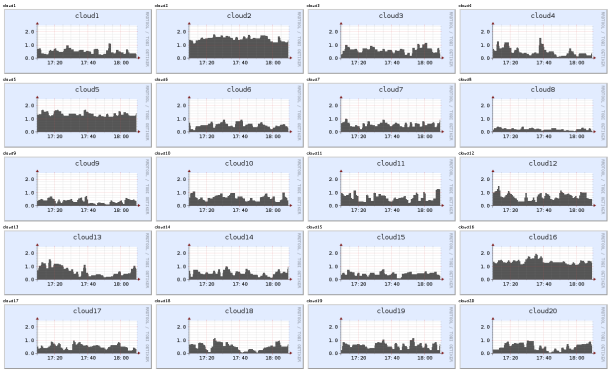


Figure 1: Screenshot from *Ganglia* where each chart represents load on a node during a one hour period.

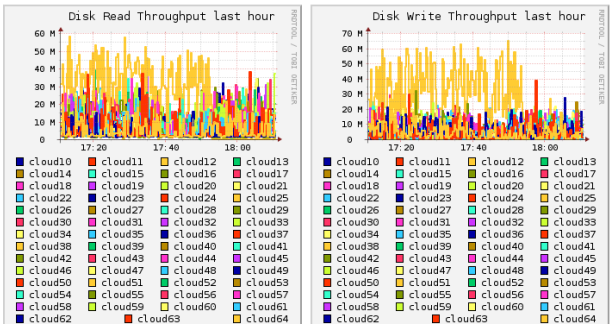
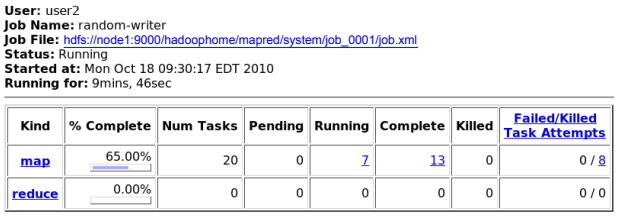


Figure 2: Screenshot of disk throughput where each line represents a node's throughput during a one hour period.



User: user2
Job Name: random-writer
Job File: hdfs://node1:9000/hadoophome/mapred/system/job_0001/job.xml
Status: Running
Started at: Mon Oct 18 09:30:17 EDT 2010
Running for: 9mins, 46sec

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	65.00%	20	0	7	13	0	0 / 8
reduce	0.00%	0	0	0	0	0	0 / 0

Figure 3: Screenshot of the Hadoop web interface showing progress of Map and Reduce tasks in a single job.

2 Hadoop cluster, Users, and Challenges

Hadoop production cluster. We analyzed the jobs and problems experienced by Hadoop users at the OpenCloud cluster for data-intensive research. The OpenCloud cluster currently has 64 worker nodes, each with 8 cores, 16 GB DRAM, 4 1TB disks and 10GbE connectivity between nodes [22].

Each Hadoop job consists of a group of Map and Reduce tasks performing some data-intensive computation. Hadoop automates job scheduling and allows multiple users to share the cluster. The master node in a Hadoop cluster typically runs two daemons: 1) the JobTracker that schedules and manages all of the tasks belonging to a running job; and 2) the NameNode that manages the Hadoop Distributed Filesystem (HDFS) namespace by providing a filename-to-block mapping, and regulating access to files by clients. Each slave node runs two daemons: 1) the TaskTracker that launches and tracks the progress of tasks on its local node; and 2) the DataNode that serves data blocks to HDFS clients.

We analyzed one-month's worth of logs generated by Hadoop's JobTracker on the OpenCloud cluster. These logs store information about the Map and Reduce tasks executed by each job. This information comprises of start times, durations, status, volume of data read and written, and user-generated performance counters for each task.

Users. The users of the cluster are researchers familiar with configuring Hadoop, writing and running MapReduce jobs, and analyzing the output generated by their jobs. These users run a diverse set of data-intensive workloads such as large-scale graph mining, text and web mining, natural language processing, machine translation problems, and data-intensive file system applications. The Hadoop users try to diagnose problems either on their own, by soliciting help from the cluster mailing list, or by escalating the problem to the system administrators. The users are primarily interested in distinguishing between two diagnostic scenarios [5]: 1) application-level problems (e.g., software bugs, workload imbalances) which they can fix on their own, and 2) infrastructural problems (e.g., contention problems, hardware problems) which they should escalate to the system administrators.

Visualization challenges. Analyzing logs from Hadoop clusters can be a daunting task due to the large amounts of information available [23, 27]. For example, in the OpenCloud cluster, we have observed jobs with over 120,000 tasks running on over 60 nodes. This information overload would be compounded in larger clusters supporting thousands of nodes. Widely-used tools such as Ganglia [17] and Nagios [19] which allow users to monitor the performance of nodes on their clusters, do not fully capture the semantics of MapReduce applications—which are important when trying to diagnose application-level problems.

In addition, the time-series representation of node-level performance generated by these tools is not scalable. The problem of visualizing information from a large number of nodes is exemplified in Figure 1 which displays Ganglia's visualization of load across multiple

Table 1: Heuristics for developing visual signatures of problems experienced in a Hadoop cluster.

Dimension	Visual Signatures		
	Application problem	**Workload imbalance**	**Infrastructural problem**
Time	*Single* user or job over time	*Single* user or job over time	*Multiple* users and jobs over time
Space	Span *multiple* nodes	Span *multiple* nodes	Typically affect *single* node, but correlated failures also occur
Value	Performance degradations and *task exceptions*	Performance degradation and *data skews*	Performance degradations and *task exceptions*

nodes on single screen. If the number of nodes is large, users will have a difficult time correlating the load across nodes. An attempt to ease correlation across nodes by displaying every node's time-series data in a single chart also leads to information overload, as shown in Figure 2 which displays disk throughput across multiple nodes.

Hadoop provides a web interface that allows users to browse application logs and monitor the progress of the Map and Reduce tasks that constitute their jobs. Hadoop also provides a separate interface where users can monitor the status of the distributed filesystem. However, these interface can become bloated for clusters with large numbers of jobs and tasks since users have to scroll through long lists of tasks, and click through multiple screens to troubleshoot a problem. The information displayed by the web interface, as shown in Figure 3, is a snapshot of the current state of a job—making it difficult to discern historical trends in performance.

3 Theia: Visual Signatures for MapReduce

We developed Theia—a visualization tool that characterizes the (mis-)behavior of large MapReduce clusters as a series of visual signatures, and facilitates troubleshooting of performance problems on the cluster. We targeted performance problems due to hardware failures or data skews, and failed jobs due to software bugs. Our current implementation does not address performance problems due to misconfigurations. The key requirements for Theia were: 1) an interactive interface that supports data exploration thereby enabling users to formulate a hypothesis on the root-cause of problems; and 2) compact representations that can support MapReduce clusters consisting of up to thousands of nodes.

We implemented Theia using a Perl script that gathered data about job execution from the job-history logs generated by Hadoop's JobTracker. We stored this information in a relational database, and generated visualizations in the web browser using the D3 framework [4].

We developed visual signatures that allow users to spot of patterns (or signatures) of misbehavior in job execution by identifying visual patterns across the time, space, and value domain. Table 1 summarizes the heuristics that we used to develop visual signatures that distinguish between application-level problems, workload imbalances between tasks from the same job, and infrastructural problems.

We developed the visualizations iteratively by manually identifying jobs which we knew had failed, and consulting with the system administrators to learn what incidents had occurred in the cluster. Next, we developed the visualizations using a subset of these incidents, and iterated through different designs to select the visualizations that best displayed the problem. We then manually verified that the visualizations matched up with the heuristics for distinguishing between different problems. These heuristics are explained below:

1. **Time dimension.** Different problems manifest in different ways over time. For example, application-level problems and workload imbalances are specific to an application; therefore, the manifestation of a problem is restricted to a single user or job over time. On the other hand, infrastructural problems, such as hardware failures, affect multiple users and jobs running on the affected nodes over time.

2. **Space dimension.** The space dimension captures the manifestation of the problem across multiple nodes. Application-level problems and workload imbalances associated with a single job manifest across multiple nodes running the buggy or misconfigured code. Infrastructural problems are typically limited to a single node in the cluster. However, a study of a globally distributed storage system [11] shows that correlated failures are not rare, and were responsible for approximately 37% of failures. Therefore, infrastructural problems can also span multiple nodes.

3. **Value dimension.** We quantify anomalies in the value domain by capturing the extent of performance degradation, data skew, and task exceptions experienced by a single job. Application-level and infrastructural problems manifest as either performance degradations or task exceptions. Workload imbalances in Hadoop clusters can stem from skewed data distributions that lead to performance degradations.

3.1 Quantifying Anomalies

To generate the visual signatures of problems, we quantify the anomalies experienced during job execution using a small number of metrics. We visualize *Map* and *Reduce* tasks separately because these tasks have very different semantics. We detect anomalies by first assuming that under fault-free conditions, the workload in a Hadoop cluster is relatively well-balanced across nodes executing the same job—therefore, these nodes are peers and should exhibit similar behavior [21]. Next, we identify nodes whose task executions differ markedly from their peers and flag them as anomalous. Aggregating task behavior on a per-node basis allows us to build compact signatures of job behavior because the number of nodes in the cluster can be several orders of magnitudes smaller than the maximum number of tasks in a job. We flag anomalous nodes based on the following metrics:

1. **Task duration.** Task duration refers to the span of a task execution for a given job on a single node. Performance degradations are detected by identifying nodes whose task durations significantly exceed those of its peers.

2. **Data volumes.** The volume of data processed is the total number of bytes read or written from the local filesystem, and the Hadoop Distributed Filesystem (HDFS). We flag anomalies when nodes process significantly more data than their peers (indicating a workload imbalance) or significantly less data (possibly indicating performance problem at the node).

3. **Failure ratios** The failure ratio is computed by aggregating tasks on a per-node basis, and calculating the ratio of failed to successful tasks. In our visualizations, we distinguish *failed* tasks from *killed* tasks which arise when speculatively-executed tasks are terminated by the task scheduler.

To compute the anomaly score we assume that metrics follow a normal distribution and use the z-score, a dimensionless quantity that indicates how much each value deviates from the mean in term of standard deviations, and is computed using the following formula: $z = \frac{x-\mu}{\sigma}$, where μ is the mean of the values, and σ is the corresponding standard deviation.

For the cluster-level visualization, we estimate the severity of problems by using a single anomaly score that flags nodes as anomalous if the geometric mean of the absolute value of the z-scores is high, *i.e.*, $AnomalyScore = (|z_{TaskDuration}| * |z_{DataVolume}| * |z_{FailureRatio}|)^{(1/3)}$.

Table 2: Metrics utilized by the visualizations.

Metric	Type
Duration	Scalar/Anomaly
Information volume	Scalar/Anomaly
Successful tasks	Count
Killed tasks	Count
Failed tasks	Count
Data skew	Percentage
Job name & job id	Text
Date & time	Text

3.2 Visualizing Anomalies

To generate visualizations that are meaningful in clusters with hundreds or thousands of nodes, we take advantage of the human brain's deduction and perception capabilities [12]. Human perception is determined by two kinds of processes: bottom-up, driven by the visual information in the pattern of light falling on the retina, and top-down, driven by the demands of attention, which in turn are determined by the needs of the tasks. [33]. In our case, the top-down task is to find those nodes, tasks, or jobs that are exhibiting anomalous behavior. We have generated three visualizations that represent different aspects of a job's execution at varying levels of granularity.

All of the visualizations in our system are designed with the following guidelines: 1) they display jobs and nodes in an order that preserves contextual information, e.g., sorting nodes by the amount of data they process; 2) they clearly distinguish between attributes that have different semantics, *e.g.*, distinguishing between Map or Reduce tasks, and failed and killed tasks; and 3) they preserve the structure of the information across the different visualizations. We use a square as the unit of representation for the different metrics of the tasks executing on a node. Table 2 shows the different metrics represented in our visualizations.

We also made the following design decisions: 1) present as much information as is understandable in a single viewport by using color and size to signal the relevant information—this allows the brain's visual query mechanism to process large chunks of information; 2) postpone the display of non-relevant attributes and take advantage of the interactive nature of web-browsers—all of our visualizations provide access to additional information by using the mouseover gesture, and allowing users to drill-down to a more detailed view of the data by clicking on the relevant interface elements.

3.3 Scalability

A Hadoop cluster might be composed of hundreds or thousands of nodes—leading to challenges in problem

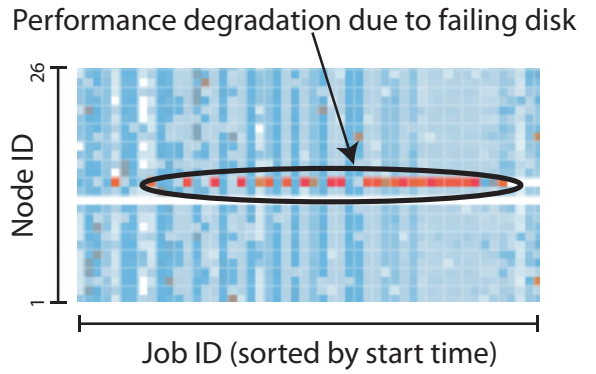


Figure 4: Visual signature of an *infrastructural problem* shows succession of anomalous jobs (darker color) due to a failing disk controller on a node.

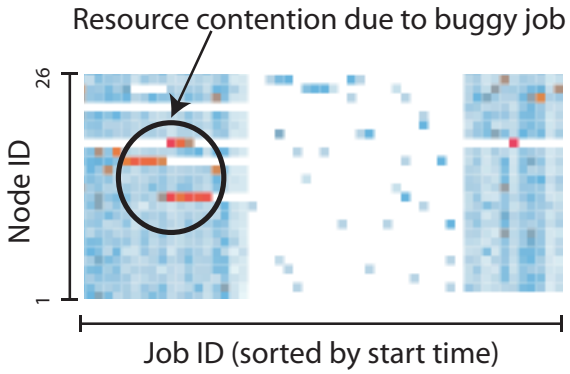


Figure 5: Visual signature of an *application-level problem*. In this case, the problem was caused by a single user submitting a resource-intensive job to the cluster repeatedly causing degraded performance on multiple nodes which were eventually blacklisted.

diagnosis brought about by the scale of the system. Research on human perception has shown that the brain can manage to distinguish features at a very high resolution, *e.g.*, differentiating up to 250 features per linear inch [32]. The following formula can be used to calculate the data density of a visualization:

$$\text{Data density} = \frac{\text{Number of data entries or features}}{\text{Area of data display}}$$

For scalability, we leveraged *high-resolution data graphics* to display the relevant information of each node in the cluster. Our heatmap visualization is capable of representing between 1,500 and 2,900 features per square inch, depending on the resolution of the display; this contrasts with the approximately 10 features per square inch shown in a typical publication [32].

4 Visualizations and Case Studies

We developed three different visualizations that facilitate problem diagnosis. We describe their design considerations, and use cases in this section. The first visualization is the anomaly heatmap which summarizes job behavior at the cluster-level; the other two visualizations are at the job-level. The first job-level visualization, referred to as the job-execution stream, allows users to scroll through jobs sequentially thus preserving the time context. The second job-level visualization, referred to as the job-execution detail, provides a more detailed view of task execution over time on each node in terms of task duration and amount of data processed.

4.1 Anomaly Heatmap

A heatmap is a high-density representation of a matrix, that we use to provide users with a high-level overview of jobs execution at the cluster-level. This visualization is formulated over a grid that shows nodes on the rows and jobs on the columns, as shown in Figure 4. The darkness of an intersection on the grid indicates a higher degree of anomaly on that node for that job. By using this visualization, anomalies due to application-level and infrastructural problems can be easily spotted as bursts of color that contrast with non-faulty nodes and jobs in the background.

Figure 4 displays the visual signature of an *infrastructural problem* identified by a succession of anomalous jobs (darker color) due to a failing disk controller on a node. Figure 5 illustrates the visual signature of an *application-level problem* caused by a single user repeatedly submitting a resource-intensive job to the cluster. The resource contention led to degraded performance across multiple nodes which were eventually blacklisted. It is easy to identify when a node has gone offline or has been blacklisted by spotting a sudden sequence of horizontal white-space; vertical white-space indicates periods of time when the cluster was idle.

The data density of the anomaly heatmap is around 2,900 features per square inch on a 109 ppi [2] display, using 2x2 pixels per job/node. This gives us a capacity to show approximately $54 \times 54 \approx 2900$ features per square inch [3]; which is equivalent to fit 1200 jobs x 700 nodes on a 27″ display.

[2] ppi: pixels per inch
[3] (109*109ppi)/(2*2pixels) = 2970.25 features per square inch

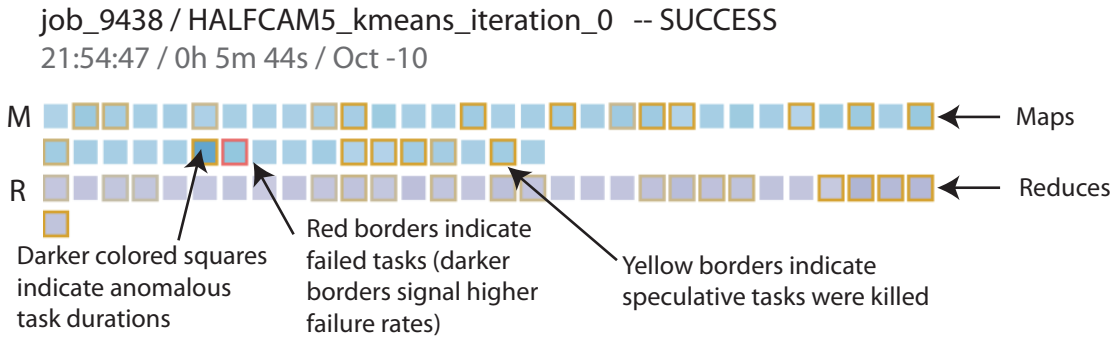


Figure 6: The *job execution stream* visualization compactly displays information about a job's execution. The header lists the job ID, name, status, time, duration and date. The visualization also highlights anomalies in task duration by using darker colors, and task status by using yellow borders for killed tasks and red borders for failed tasks. The nodes are sorted by decreasing amount of I/O processed.

Figure 7: Visual signature of *bugs in the Map phase*. Failures spread across all Map nodes (solid red border) typically signals a bug in the Map phase.

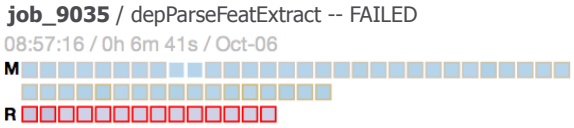


Figure 8: Visual signature of *bugs in the Reduce phase*. Failures spread across all Reduce nodes (solid red border) typically signals a bug in the Reduce phase.

Figure 9: Visual signature of *data skew*. A node with anomalous task durations (darker color) and high volume of I/O (first nodes in the list) can signal data skew.

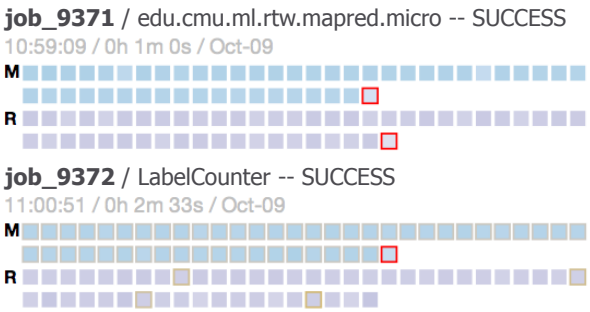


Figure 10: Visual signature of *infrastructural problem*. A node with failures (red border) spread across multiple jobs can signal problems with that node.

4.2 Job Execution Stream

The job execution stream, shown in Figure 6, provides a more detailed view of jobs while preserving information about the context by showing a scrollable stream of jobs sorted by start time. In addition to displaying general information about the job (job ID, job name, start date and time, job duration) in the header, this visualization has a more complex grammar that allows the representation of an extended set of signatures.

Since the application-semantics of Map and Reduce tasks are very different, we divided nodes into two sets: the Map set and the Reduce set. Each of these sets represent nodes that executed Map or Reduce tasks for a given job sorted by decreasing I/O. The intuition behind the sort order is that nodes that process significantly less data than their peers might be experiencing performance problems. We enhanced the representation of each node with a colored border that varies in intensity depending the ratio of failed tasks to successful tasks, or the ratio of killed tasks to successful tasks; killed tasks arise when the task scheduler terminates speculative tasks that are still running after the fastest copy of the task completed. Killed tasks are represented using a yellow border, which is overloaded by a red border if there are any failed tasks.

The job execution stream visualization allows us to

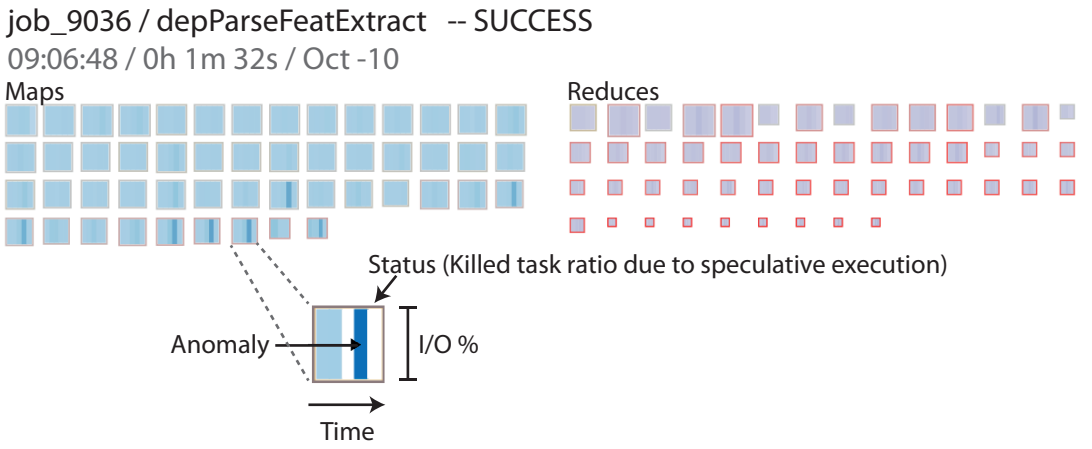


Figure 11: The *job execution detail* visualization highlights both the progress of tasks over time, and the volume of data processed. Each node is divided on five stripes that represent the degree of anomalies in tasks executing during the corresponding time slot; the size of the square represents the proportion of I/O processed by that node.

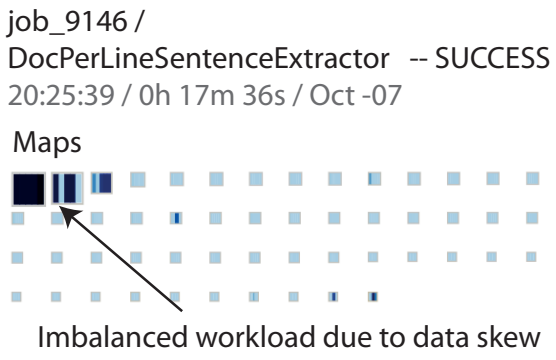


Figure 12: Visual signature of *data skew*. A single node with high duration anomaly (darker color) and high amount of I/O (first nodes in the list, large square size) can signal data skew.

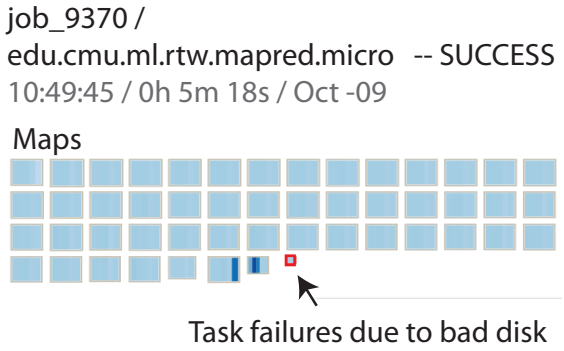


Figure 13: Visual signature of an *infrastructural problem*. A single node with high task durations (darker color) or failed tasks (red border), and a low volume of I/O (small square size) can indicate a hardware failure.

generate signatures for the following scenarios:

1. *Application-level problems.* Bugs in either the Map or Reduce phase manifest as a large number of failed tasks across all nodes in either the Map or Reduce set (see Figures 7 and 8).

2. *Workload imbalance or data skew.* A *data skew* can be identified by spotting nodes with anomalous task durations on the first positions of the node list. For example, see the dark left-most node in Figure 9.

3. *Infrastructural problems. Infrastructural problems* can be detected if a node recurrently fails across multiple jobs, as shown in Figure 10.

4.3 Job Execution Detail

The *job execution detail* visualization provides a more detailed view of task execution and is less compact than the *job execution stream*. The *job execution detail* visually highlights both the progress of tasks over time, and the volume of data processed as shown in Figure 11. Nodes are still represented as two sets of squares for Map and Reduce tasks; however, given that there is additional space available since we are only visualizing one job at a time, we use the available area on each of the squares to represent two additional variables: 1) anomalies in task durations over time by dividing the area of each node into five vertical stripes, each corresponding to a fifth of

Table 3: Problems diagnosed by cluster-level and job-level visualizations in Theia. The infrastructural problems consisted of 42 disk controller failures, 2 full hard drives, and 1 network interface controller (NIC) failure. The infrastructural problems diagnosed by the job execution stream were a subset of those identified by the heatmap.

Type	Total problems	Diagnosed by heatmap	Diagnosed by job execution stream
Application-level problem	157	0	157
Data-skew	2	2	2
Infrastructural problem	45	33	10

the total time spent executing tasks on that node—darker colors indicate the severity of the anomaly while white stripes represent slots of time where no information was processed; 2) percentage of total I/O processed by that node, *i.e.*, reads and writes to both the local filesystem and the Hadoop distributed filesystem (HDFS)—larger squares indicate higher volumes of data.

Figure 12 shows the visual signature of a data skew where a subset of nodes with anomalous task durations (darker color) and high amounts of I/O (first nodes in the list, large square size) indicate data skew. In this visualization, the data skew is more obvious to the user when compared to the same problem visualized using the job execution stream in Figure 9. Infrastructural problems as shown in Figure 13 can be identified as a single node with high task durations (darker color) or failed tasks (red border), coupled with a low volume of I/O (small square size) which might indicate a performance degradation. The data density of this visualization, using 24x24 squares and 7 features per square (5 time slots + I/O percentage + status ratio) on a 109 ppi screen, is about 112 numbers per square inch—this allows us to display up to 4800 nodes on a 27″ display (2400 nodes for Maps + 2400 nodes for Reduces).

4.4 Interactive User Interface

Theia is implemented as an interactive web interface supporting a top-down data exploration strategy that allows users to form, and confirm their hypotheses on the root-cause of the problems. Users can navigate from the cluster-level visualization to the job-level visualizations by clicking on the relevant interface elements. Theia takes 1–2 seconds to change between views. All of our visualizations provide access to additional information by using the mouseover gesture. Figure 14 uses the *job execution detail* visualization to show how the performance of a job was degraded by a failed NIC (Network Interface Controller) at a node highlighted using a dark blue square. By hovering over the failed node, a user can obtain additional information about the behavior of tasks executed on that node. For example, a user can observe that 50% of the tasks executed on this node failed.

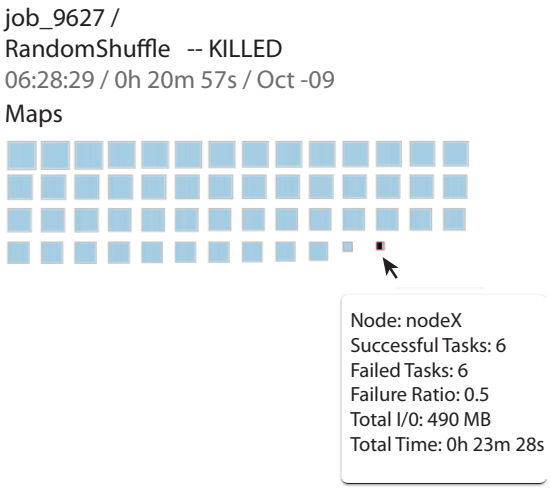


Figure 14: *Interactive User Interface*. This job execution detail visualization shows degraded job performance due to a NIC failure at a node. Hovering over the node provides the user with additional information about the behavior of tasks executed on that node.

4.5 Summary of Results

We generated our visualizations using one month's worth of logs generated by Hadoop's JobTracker on the OpenCloud cluster. During this period, 1,373 jobs where submitted, comprising of a total of approximately 1.85 million tasks. From these 1,373 jobs, we manually identified 157 failures due to application-level problems, and 2 incidents of data skew. We also identified infrastructural problems by analyzing a report of events generated by the Nagios tool installed on the cluster. During the evaluation period, Nagios reported 68 messages, that were associated with 45 different incidents namely: 42 disk controller failures, 2 full hard drives, and 1 network interface controller (NIC) failure.

We evaluated the performance of Theia by manually verifying that the visualizations generated matched up with the heuristics for distinguishing between different problems described in Table 1. Table 3 shows that we successfully identified all the application-level problems and data skews using the job execution stream (similar

results are obtained using the job execution detail). In addition, the anomaly heatmap was able to identify 33 of the 45 infrastructural problems (the problems identified by the job execution stream are a subset of those identified by the heatmap). We were unable to detect 4 of the infrastructural problems because the nodes had been blacklisted. We hypothesize that the heatmap was unable to detect the remaining 8 infrastructural problems because they occurred when the cluster was idle.

5 Related Work

The visualizations we generated are based on previous research on visual log-based performance debugging [27]. This paper presents a Host-State decomposition visualization from which we borrowed elements to build our Job-Node heatmap visualization. Bodik et al. [3] present a combined approach to failure localization using anomaly detection and visualization that leverages heatmaps. Theia also uses heatmaps to visualize anomalies, but we also present heuristics that allow users to distinguish between different types of problems.

HiTune [8], X-trace [10], and Dapper [26] are techniques for tracing causal request paths in distributed systems, but they also offer support for visualizing requests whose causal structure or duration are anomalous. While Theia does not currently support the visualization of the causal structure of Hadoop jobs, our visualizations provide a compact representation of multiple aspects of job behavior such as task duration, task status, and data volume within a single viewport.

Artemis [7] provides a pluggable framework for distributed log collection, data analysis, and visualization. LiveRAC [18] is tool for browsing and correlating time-series data in large-scale systems. Otus [24] collects and correlates performance metrics from distributed components and provides views of the data as a time-series. Both LiveRAC and Otus support visualization of generic time-series data. Artemis supports visualization plugins that display domain-specific histograms and time-series data. Theia, on the other hand, incoporates semantic information about job execution—allowing us to distinguish application-level problems from infrastructural problems.

Chau et al. [6] present an initiative for combining machine learning and visualizations for large network data exploration. Amershi et al. [1] use a similar approach for alarm triage on a network of more than 15,000 devices. We are investigating whether we can incorporate more sophisticated anomaly detectors into our tool.

6 Conclusion

Theia is a visualization tool that exploits application-specific semantics about the structure of MapReduce jobs to generate high-density, interactive visualizations of job performance that scale to support current industry deployments. Theia uses heuristics to identify visual signatures of problems that allow users to distinguish application-level problems (e.g., software bugs, workload imbalances) from infrastructural problems (e.g., contention problems, hardware problems). We have evaluated our visualizations using real problems experienced by Hadoop users at a production cluster over a one-month period. Our visualizations correctly identified 192 out of 204 problems that we observed.

In the future, we plan to integrate these visualizations as part of the standard stack of tools for Hadoop diagnosis. Detecting anomalies in heterogeneous clusters is an open issue as our assumption of peer-similarity might not hold in these systems. Other enhancements to the visual interfaces that we are considering are: 1) to display nodes hosted in the same rack together to increase our ability to spot failures at the rack-level; 2) to generate high-density per-task views and to incorporate resource-usage information such as CPU and memory usage; and 3) to develop algorithms that automatically classify application-level and infrastructural problems using the heuristics described in this paper.

7 Acknowledgments

We thank Mitch Franzos, Michael Stroucken, Zisimos Economou, and Kai Ren for access to the OpenCloud Hadoop logs. We thank our shepherd, Cory Lueninghoener, the conference reviewers, and our colleagues, Raja Sambasivan and Ilari Shafer, for their valuable feedback. We thank the members and companies of the PDL Consortium (including Actifio, APC, EMC, Emulex, Facebook, Fusion-io, Google, Hewlett-Packard Labs, Hitachi, Intel, Microsoft Research, NEC Labs, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, STEC, Symantec, VMware, and Western Digital) for their support. This research was sponsored in part by the project CMUPT/RNQ/0015/2009 (TRONE—Trustworthy and Resilient Operations in a Network Environment), and by Intel via the Intel Science and Technology Center for Cloud Computing (ISTC-CC).

References

[1] AMERSHI, S., LEE, B., KAPOOR, A., MAHAJAN, R., AND CHRISTIAN, B. CueT: human-guided fast and accurate network alarm triage. In *ACM Conference on Human Factors in Computing Systems (CHI)* (Vancouver, BC, Canada, May 2011).

[2] ANANTHANARAYANAN, G., KANDULA, S., GREENBERG, A., STOICA, I., LU, Y., SAHA, B., AND HARRIS, E. Reining in the

outliers in map-reduce clusters using Mantri. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Vancouver, BC, Canada, October 2010), pp. 1–16.

[3] BODIK, P., FRIEDMAN, G., BIEWALD, L., LEVINE, H., CANDEA, G., PATEL, K., TOLLE, G., HUI, J., FOX, A., JORDAN, M. I., AND ET AL. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *IEEE International Conference on Automatic Computing (ICAC)* (Seattle, WA, June 2005), pp. 89–100.

[4] BOSTOCK, M., OGIEVETSKY, V., AND HEER, J. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011).

[5] CAMPBELL, J. D., GANESAN, A. B., GOTOW, B., KAVULYA, S. P., MULHOLLAND, J., NARASIMHAN, P., RAMASUBRAMANIAN, S., SHUSTER, M., AND TAN, J. Understanding and improving the diagnostic workflow of mapreduce users. In *ACM Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT)* (Boston, Massachusetts, December 2011).

[6] CHAU, D. H., KITTUR, A., HONG, J. I., AND FALOUTSOS, C. Apolo: making sense of large network data by combining rich user interaction and machine learning. In *ACM Conference on Human Factors in Computing Systems (CHI)* (Vancouver, BC, Canada, May 2011).

[7] CRETU-CIOCARLIE, G. F., BUDIU, M., AND GOLDSZMIDT, M. Hunting for problems with artemis. In *USENIX Workshop on Analysis of System Logs* (San Diego, CA, December 2008).

[8] DAI, J., HUANG, J., HUANG, S., HUANG, B., AND LIU, Y. Hitune: dataflow-based performance analysis for big data cloud. In *USENIX Annual Technical Conference (ATC)* (Portland, OR, June 2011).

[9] DEAN, J., AND GHEMAWAT, S. MapReduce: simplified data processing on large clusters. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (San Francisco, CA, December 2004), pp. 137–150.

[10] FONSECA, R., PORTER, G., KATZ, R. H., SHENKER, S., AND STOICA, I. X-Trace: A pervasive network tracing framework. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (Cambridge, MA, Apr. 2007).

[11] FORD, D., LABELLE, F., POPOVICI, F. I., STOKELY, M., TRUONG, V.-A., BARROSO, L., GRIMES, C., AND QUINLAN, S. Availability in globally distributed storage systems. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Vancouver, CA, October 2010), pp. 61–74.

[12] KALAWSKY, R. S. Gaining greater insight through interactive visualization: A human factors perspective. In *Trends in Interactive Visualization*, R. Liere, T. Adriaansen, and E. Zudilova-Seinstra, Eds., Advanced Information and Knowledge Processing. Springer London, 2009.

[13] KANDULA, S., MAHAJAN, R., VERKAIK, P., AGARWAL, S., PADHYE, J., AND BAHL, P. Detailed diagnosis in enterprise networks. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* (Barcelona, Spain, August 2009), pp. 243–254.

[14] KAVULYA, S., DANIELS, S., JOSHI, K. R., HILTUNEN, M. A., GANDHI, R., AND NARASIMHAN, P. Practical experiences with chronics discovery in large telecommunications systems. In *IEEE Conference on Dependable Systems and Networks (DSN)* (Boston, MA, June 2012).

[15] LIU, Z., LEE, B., KANDULA, S., AND MAHAJAN, R. Netclinic: Interactive visualization to enhance automated fault diagnosis in enterprise networks. In *IEEE Conference on Visual Analytics Science and Technology* (Salt Lake City, UT, October 2010), pp. 131–138.

diagnosis in a large IPTV network. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* (Barcelona, Spain, August 2009), pp. 231–242.

[17] MASSIE, M. L., CHUN, B. N., AND CULLER, D. E. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing 30* (June 2004), 817–840.

[18] MCLACHLAN, P., MUNZNER, T., KOUTSOFIOS, E., AND NORTH, S. C. LiveRAC: interactive visual exploration of system management time-series data. In *ACM Conference on Human Factors in Computing Systems (CHI)* (Florence, Italy, April 2008), pp. 1483–1492.

[19] NAGIOS ENTERPRISES LLC. Nagios. http://www.nagios.org/, September 2012.

[20] OLINER, A. J., KULKARNI, A. V., AND AIKEN, A. Using correlated surprise to infer shared influence. In *IEEE Conference on Dependable Systems and Networks (DSN)* (Chicago, IL, July 2010), pp. 191–200.

[21] PAN, X., TAN, J., KAVULYA, S., GANDHI, R., AND NARASIMHAN, P. Ganesha: Blackbox diagnosis of MapReduce systems. *SIGMETRICS Perform. Eval. Rev. 37* (January 2010).

[22] PARALLEL DATA LAB. Apache Hadoop. `http://wiki.pdl.cmu.edu/opencloudwiki/`, September 2012.

[23] RABKIN, A., AND KATZ, R. Chukwa: a system for reliable large-scale log collection. In *USENIX Conference on Large Installation System Administration (LISA)* (San Jose, CA, November 2010), USENIX Association.

[24] REN, K., LÓPEZ, J., AND GIBSON, G. Otus: resource attribution in data-intensive clusters. In *Workshop on MapReduce and its applications (MapReduce)* (San Jose, CA, June 2011).

[25] SAMBASIVAN, R. R., ZHENG, A. X., ROSA, M. D., KREVAT, E., WHITMAN, S., STROUCKEN, M., WANG, W., XU, L., AND GANGER, G. R. Diagnosing performance changes by comparing request flows. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (Boston, MA, March 2011), pp. 43–56.

[26] SIGELMAN, B. H., BARROSO, L. A., BURROWS, M., STEPHENSON, P., PLAKAL, M., BEAVER, D., JASPAN, S., AND SHANBHAG, C. Dapper, a large-scale distributed systems tracing infrastructure. Tech. rep., Google, Inc., 2010.

[27] TAN, J., KAVULYA, S., GANDHI, R., AND NARASIMHAN, P. Visual, log-based causal tracing for performance debugging of mapreduce systems. In *IEEE International Conference on Distributed Computing Systems (ICDCS)* (Genova, Italy, June 2010), pp. 795–806.

[28] TAN, J., PAN, X., KAVULYA, S., GANDHI, R., AND NARASIMHAN, P. SALSA: Analyzing Logs as StAte Machines. In *USENIX Workshop on Analysis of system logs* (San Diego, California, December 2008).

[29] THE APACHE SOFTWARE FOUNDATION. Apache Hadoop. `http://hadoop.apache.org/`, September 2012.

[30] THE APACHE SOFTWARE FOUNDATION. PoweredBy Hadoop. `http://wiki.apache.org/hadoop/PoweredBy`, September 2012.

[31] THUSOO, A., SHAO, Z., ANTHONY, S., BORTHAKUR, D., JAIN, N., SEN SARMA, J., MURTHY, R., AND LIU, H. Data warehousing and analytics infrastructure at facebook. In *ACM Conference on Management of Data (SIGMOD)* (Indianapolis, IN, June 2010).

[32] TUFTE, E. R. *The Visual Display of Quantitative Information*, 2nd ed. Graphics Pr, May 2001.

[33] WARE, C. *Visual Thinking: for Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

Lessons in iOS device configuration management

Tim Bell <tbell@trinity.unimelb.edu.au>
Trinity College, the University of Melbourne

Abstract

After an initial trial, Trinity College has deployed iPads to its 600 international students. Supporting these devices on the Trinity network has presented challenges relating to configuration management, wireless network capacity, and server load.

We addressed the problem of configuration management with a web application written using the Django framework, which enables students to download and install a customized configuration profile to their iPads.

This paper describes the requirements and implementation of the configuration management solution, security issues, as well as the experiences and lessons learned from its use.

Tags: configuration management, iOS, wireless, web application framework

1 Introduction

Trinity College is a residential college affiliated with the University of Melbourne in Victoria, Australia. As well as providing accommodation for approximately 300 university students, for the past 20 years, Trinity has offered a Foundation Studies (FS) program to provide a pathway to university entrance for international students.

Starting in 2010, Trinity trialled the use of Apple iPads with a small number of FS students with a pilot program over a period of about 15 months. The pilot had two objectives:

1. To explore and evaluate the educational value of iPads.

2. To discover what IT infrastructure improvements were needed to support iPad use by students and staff.

The pilot involved approximately 20 teaching staff and 50 students.[1] The iPad had been launched just 5 months[2] before the pilot began, so for many of the participants, it was their first experience with the device.

The chief technical requirement was to enable the iPads to work smoothly on the wireless network (which uses 802.1x authentication), and to interact with the existing network services such as email and calendaring. We decided that for ease of operation, at least with this pilot group, we wanted the iPads to be distributed to the users fully pre-configured.

We used Apple's iPhone Configuration Utility[3] (iPCU) to create a configuration profile that contained the settings we needed, and loaded it onto a test device. (Like the iPhone, the iPad runs an operating system that Apple now calls "iOS"; the same configuration utility can be used for the iPhone, iPad and iPod touch.)

However, we discovered that iPCU did not provide the complete or ideal solution. First of all, we required each configuration profile to be customized to each user, to include their email address as well as authentication credentials for the wireless network and email client. iPCU required that these details be updated manually for each user. Second, iPCU required that each iPad be connected via USB to a computer for the configuration profile to be loaded.

These two problems were manageable for the pilot (and indeed, one staff member unboxed and configured all 70 iPads with iPCU via USB connection), but we realised that they would need proper solutions if the pilot were to be successful and iPads rolled out to the whole cohort of 600 students.

[1] http://ipadpilot.wordpress.com/2010/09/17/first-post/

[2] http://www.apple.com/pr/library/2010/03/05iPad-Available-in-US-on-April-3.html

[3] http://help.apple.com/iosdeployment-ipcu/

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>PayloadContent</key>
        <array>
                <dict>
                        <key>PayloadDescription</key>
                        <string>Configures security-related items.</string>
                        <key>PayloadDisplayName</key>
                        <string>Passcode</string>
                        <key>PayloadType</key>
                        <string>com.apple.mobiledevice.passwordpolicy</string>
                        <key>PayloadUUID</key>
                        <string>AFB6F3CF-303D-442F-8B38-0D7A1D554AA2</string>
                        <key>PayloadVersion</key>
                        <integer>1</integer>
                        <key>allowSimple</key>
                        <true/>
                        <key>forcePIN</key>
                        <true/>
                        <key>maxFailedAttempts</key>
                        <integer>5</integer>
                        <key>minLength</key>
                        <integer>4</integer>
                </dict>
        </array>
        <key>PayloadRemovalDisallowed</key>
        <false/>
        <key>PayloadType</key>
        <string>Configuration</string>
        <key>PayloadUUID</key>
        <string>85125A7C-2F36-4354-959C-18B1AB1CD928</string>
        <key>PayloadVersion</key>
        <integer>1</integer>
</dict>
</plist>
```

Figure 1: Sample mobileconfig configuration profile

2 Initial solution

While we accepted that manually configuring each iPad using iPCU would be workable (although not ideal), we knew that manually entering the configuration details for each user into iPCU would be extremely tedious and prone to errors.

Luckily, the configuration profile that iPCU creates to be loaded onto each device is a fairly simple XML file,[4] and was amenable to being "hacked" manually. (See Figure 1 for short sample profile that just configures the device's passcode; some details are omitted for brevity.) We created a working profile, and then manually edited it to insert template variables in place of the following to create a template file:

- username
- password
- email address
- PayloadUUID

For example, in Figure 1, this fragment:

```
<key>PayloadUUID</key>
<string>85125A7C-2F36-4354-959C-
    18B1AB1CD928</string>
```

would be edited to substitute the "$TCUUID2" template variable ("TC" is for Trinity College, and "2" since it's the second PayloadUUID in the profile), like this:

```
<key>PayloadUUID</key>
<string>$TCUUID2</string>
```

The PayloadUUID needs to be globally unique (according to the specification), and each configuration fragment (for example, relating to email server configuration) requires its own; we used the `uuidgen(1)`[5] command in Mac OS X to generate them.

[4] `http://tinyurl.com/iPhoneConfigurationProfile`

[5] `http://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/uuidgen.1.html`

We then wrote a Python script to iterate over a list of usernames and passwords and substitute them (along with the generated UUIDs) into the template file, creating one "username.mobileconfie" file per user. These files were then loaded back into iPCU for uploading to the iPads.

This initial solution proved satisfactory for the pilot, and we were tempted to stop there, since we'd already significantly improved the deployment process. On reflection, however, we realized that a process that was acceptable for 70 students and staff would quickly become a burden when it needed to be repeated for 100 staff members, and a new cohort of 600 students each year. We needed to improve the solution, and came up with these requirements:

- Configuration profiles must be automatically generated from the template.
- The users must be able to load the configuration profiles themselves.
- The profiles must be updatable (which requires users to be able to reload them).

Before we describe the solution we eventually created, we'll look at some of the alternatives that were considered.

3 Other solutions

Apple announced their Mobile Device Management[6] (MDM) platform in April 2010[7], although it wasn't supported by a released iOS version for several months. Furthermore, Apple did not release a service that implemented the platform, but left it to third-party vendors to create and market their own products.

MDM is most often used to provide complete device management, including installation of apps and configuration settings, and removes a certain amount of configurability and control from the user. That may be appropriate in a corporate setting, but the model of iPad deployment at Trinity was that students would pay for (via student fees) their own iPad, which they would then own and be responsible for. MDM's high level of control was considered to be incompatible with students' responsibility for their own iPads.

Another problem with MDM solutions is that they don't incorporate user passwords in the configurations that are deployed, so that users are prompted (sometimes repeatedly) to enter their password. While this is more secure, it makes the initial user experience poorer; this was considered a fairly important factor for us.

JAMF Software sells the Casper Suite[8] for Mac OS X and iOS client management, which provides an MDM solution for iOS. Casper was considered for use, but it was thought to be overkill for requirements we had, as well as requiring a per-device license fee (which, being in education, is always a turn-off). We have however since adopted it for managing staff iPads, since Casper allows profiles for individual users to be customised very simply, which has been very useful for staff devices.

Apple released their own partial MDM solution, "Apple Configurator"[9], in March 2012.[10] However, Apple Configurator is intended for use by IT staff to preconfigure iOS devices one-by-one, and doesn't meet our requirements due to the level of manual handling required. (Furthermore, it wasn't yet available when we were looking for a solution.)

4 iOS Configurator

We conceived our solution as a web-based adaption of our existing script. To ease transformation to make it web-based, we looked to use a web application framework; since we already had some experience with Django, that was the logical choice.

4.1 Coding

Django[11] is a Python-based web application framework, with its own built-in template engine, and an object-relational mapper that works with a number of different database backends.

Our application itself is very small: 167 lines of Python source (including whitespace and comments), with a further 229 lines in settings and configuration files. There are also HTML templates for the web pages that form the user interface.

The application is based around three models:

- A user group model. The template file that is used to generate a configuration profile is based on the user's group (e.g. staff or student); this model stores the template.
- A mobileconfig model; this model incorporates our original code to generate the configuration profile from the template and user information, and is not persisted in the database.

[6] http://www.apple.com/iphone/business/integration/mdm/

[7] http://www.apple.com/pr/library/2010/04/08Apple-Previews-iPhone-OS-4.html

[8] http://jamfsoftware.com/products/casper-suite

[9] http://help.apple.com/configurator/mac/1.0/

[10] http://www.informationweek.com/byte/reviews/personal-tech/mobile-apps/232602364

[11] http://www.djangoproject.com

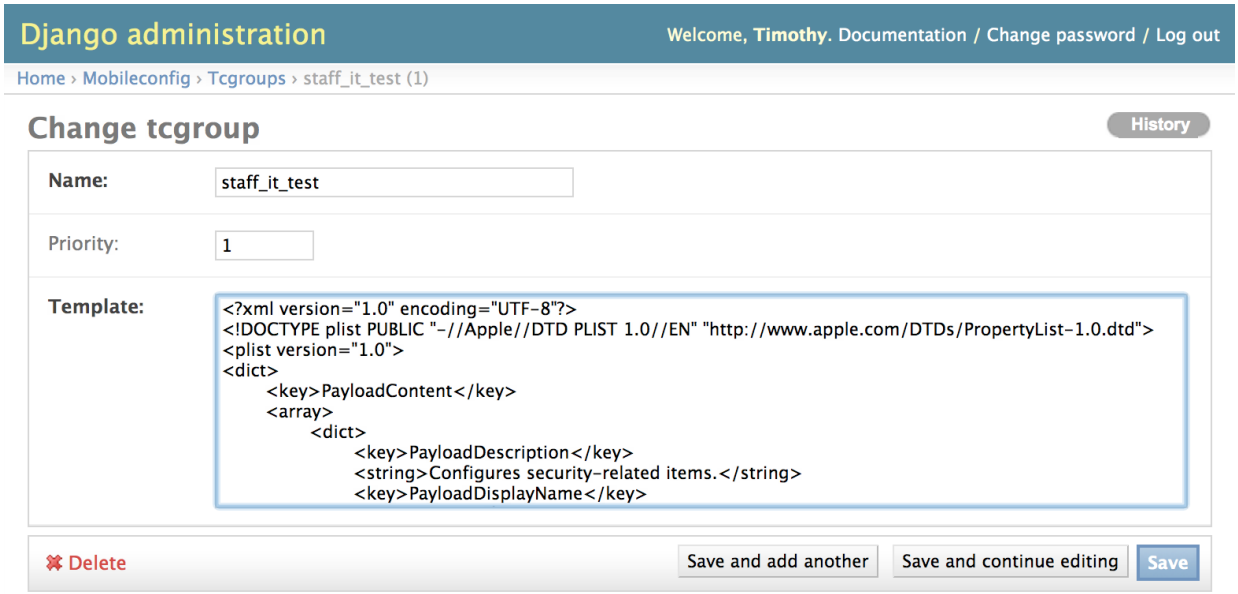


Figure 2: Django admin screen for template loading

- A download model, which records each download of a configuration profile for each user.

We use Django's built-in admin interface (see Figure 2) to manage the groups (including uploading template files) and view download records, which is more than adequate for our needs, and requires significantly less effort to set up than custom views.

Authentication is handled using Django's LDAP authentication backend, which works against our OpenLDAP server. As well as authenticating the users (see Figure 3), LDAP is used to retrieve the user's group (which is then used to select the correct template) and name (which is used in the email client configuration). During authentication, the user's password is captured in the login session, so it can be substituted into the configuration profile.

Once the user is authenticated, they are given a link to download their configuration profile. The download has MIME type "application/x-apple-aspen-config", which causes iOS devices to prompt to install it once it downloads. Once the profile is downloaded, iOS Configurator's job is done.

Note that as with any method of deployment that requires the user to accept the configuration profile on the iOS device, the user also has the power to remove the profile at a later stage if desired. They can also override any of the settings in the profile by setting them directly.

4.2 Deployment

We deployed iOS Configurator on a Debian Linux virtual machine, with Apache as the webserver and using mod_wsgi[12] to host the Django app. This is far from an ideal deployment setup, but it has the advantage of being relatively simple, and leveraging our existing familiarity with Apache. Best practice would have static resources (image and CSS files) served from one web server, while the Django app was served by a different server, or indeed multiple servers. However, the improved scalability this deployment model provides was not needed for our use case.

iOS Configurator was deployed on an HTTPS site, so that authentication would be secure, and the configuration profile (containing cleartext passwords) would be encrypted during transit over the wireless network. (See Section 6 for more details.)

4.3 Wireless

Since the configuration profile we want to install includes the configuration for connecting to the wireless network, how will the iPads connect to the iOS Configurator web app? The simple solution to this chicken-and-egg problem is to use an open wireless network that only permits access to iOS Configurator and related pages.

We already had such a wireless network, and used it to provide configuration information for other types of devices. In fact, the menu seen at the top of Figure 3 is from

[12] http://code.google.com/p/modwsgi/

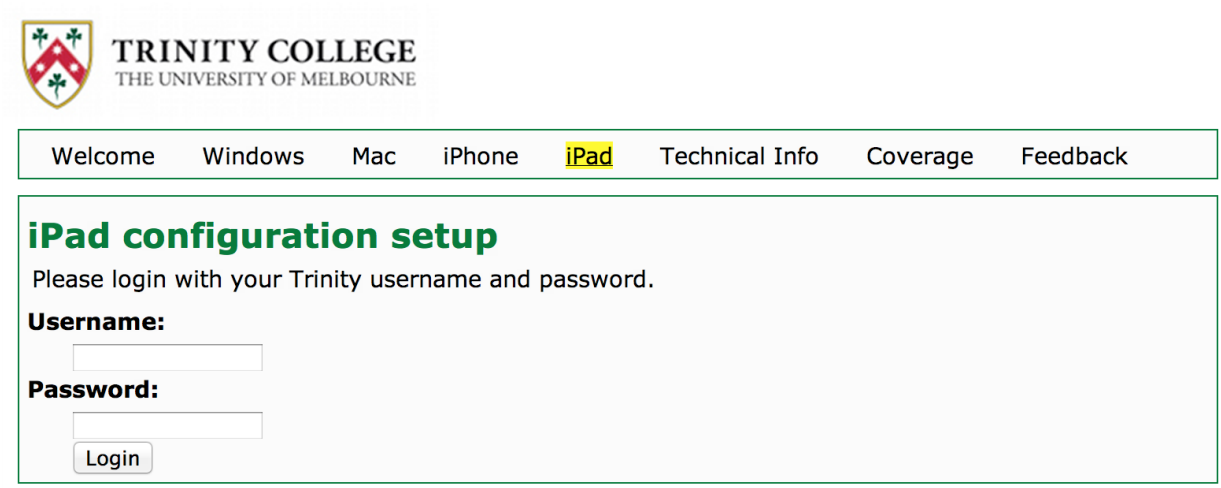


Figure 3: iOS Configuration login screen

the existing information website; when we deployed iOS Configurator, we added the "iPad" item to the menu.

One complication with blocking access to most web pages is that Apple's Captive Network Support[13] (CNS) detection causes the iOS Configurator web page to pop up automatically when joining the open wireless network. This seems to be an advantage, but we discovered that the prompt to install the downloaded configuration profile appears underneath the web page, where it is invisible to the user, and can't be accessed until the iOS Configurator web page (which is presented modally) is dismissed by tapping "Cancel". This is potentially very confusing to the user, and we therefore wanted to avoid the situation.

To overcome this problem, we adjusted the blocks that applied to the open wireless network to allow the requests used by CNS. While this fixed the problem, it meant that users had to be instructed to open the Safari web browser and to navigate to the iOS Configurator (although any web request would be redirected to the iOS Configurator).

5 Deployment issues

The iOS Configurator was first deployed in August 2011 for use by an intake of 50 students. (Although there are 600 FS students at Trinity, they are split into two main groups, starting in February and July, with a number of other intakes feeding into those groups. Thus we never have to deal with 600 new students at the same time; the largest intake is usually around 300 with some intakes as small as 20 students.) That deployment was marred by unrelated issues with the wireless network[14], although the iOS Configurator appeared to function well.

5.1 Webserver configuration

The next large intake of students to use the iOS Configurator was in February 2012. During the iPad configuration sessions, we discovered that enough students were accessing iOS Configurator simultaneously that it was overloading the webserver. Part of the cause of the problem was that the sessions where the iPads were handed out to the students were run in large groups, and in order to make sure that everyone completed all the steps, the presenter would wait for everyone to catch up before moving onto the next step. That meant that when it was time to configure the iPad, everyone tried to connect to iOS Configurator simultaneously.

The other part of the cause was that we had used a default configuration of the `MaxClients` Apache configuration parameter; we discovered that 30 Apache processes consumed the available memory, and the server started to swap. Tuning `MaxClients` down to 15 (which would mean about half the available memory was used) was the simple solution to this problem.

We realised that we hadn't encountered this overloading issue in the August 2011 deployment because the

[13] `https://developer.apple.com/library/IOs/documentation/SystemConfiguration/Reference/CaptiveNetworkRef/Reference/reference.html`

[14] We discovered that the wireless controller was underspecified for the recently expanded wireless network; upgrading to the larger model controller fixed the wireless network issues.

wireless issues had ensured that students were all accessing iOS Configurator at different times.

6 Security

In our use case, the configuration profile that is delivered to the iOS device contains the user's authentication credentials, that is, their username and password. This raises several security concerns.

6.1 Authentication and profile delivery

The first main security concern is how the configuration profile is delivered to the iPad. We use an open, insecure wireless network for delivery of the profile, so there is no security protection there. However, all of the configuration process happens over HTTPS, using a commercially-provided SSL certificate. HTTPS provides encryption of the authentication and profile delivery over the air. The use of a commercial SSL certificate, rather than a self-signed one, means that students are not prompted to accept a certificate signed by an unknown and untrusted certificate authority, and allows for the server's identity to be checked, to avoid the simplest man-in-the-middle attacks. Thus, any cleartext passwords and other sensitive information in the configuration profile can't be sniffed over the network.

Furthermore, the download link for the configuration profile is only available via the authenticated session, which times out after a short period, which means that downloading profiles without authenticating is not possible.

6.2 Cleartext passwords in profiles

The second security concern is the presence of cleartext passwords in the profile on the iPads. (Note that cleartext passwords are not stored on the server beyond the duration of the authenticated session.) This is a legitimate concern, but we believe that the risk it creates is acceptable for the benefits achieved in our situation.

iOS has a "Keychain" facility which is used to store passwords that the user enters; the passwords are protected by encryption, and require the screen lock code to retrieve in most cases. However, iOS Configurator is designed to include the user's plaintext password in the configuration profile directly, where it is not protected by encryption.

It is not a simple process to recover the plaintext passwords from an iPad, since the profile isn't directly viewable on the device; in fact, once the profile has been accepted and installed, the configuration details it contains are entered to the device's Keychain as if they had been entered directly.

However, there are methods of obtaining such details stored in the Keychain of a device once physical access to the device has been obtained.[15] The level of risk depends on the way the iOS app has been written, the version of iOS running (with iOS 5 significantly more secure than iOS 4), and the length and complexity of the user's screen lock code to unlock the device.[16] In particular, a four-digit lock code is not secure (taking only 9 minutes to attack by brute force), while a six-digit alphanumeric lock code is effectively secure (taking over a year to brute force).

Crucially for us with our use case, compromising of student authentication credentials is not a significant threat; nothing of monetary value is accessible with a student account (unless a student decides to store sensitive information such as credit card details), and student data such as files and emails are backed up and can be restored if compromised or lost. On the other hand, the benefit we get from having cleartext passwords in the profile is a much simpler configuration process for students, which is greatly desired. Furthermore, there appear to be no alternatives to achieve the same outcome.

A challenge resulting from making this choice is communicating clearly to the students the potential risks of storing sensitive information on their iPads; the vast majority of these students speak English as a second (or third) language, which makes it even trickier to try to communicate complex security issues. We are currently exploring ways to improve awareness of these issues.

Any organization considering the use of a configuration profile delivery system such as presented here must perform its own risk analysis to decide whether the risks of including passwords and other sensitive security information in a configuration profile outweigh the benefits.

7 Improved profile template loading

In Section 2 we briefly described how the configuration profile created with iPCU needed to be manually edited (a.k.a. hacked by hand) to create the template to be loaded into iOS Configurator. That editing process is time-consuming and error-prone; what is worse, any errors may not be realised until a user tries to download a profile, or load the profile on their iOS device, depending on the particular error that was made. Although we made do with this state of affairs for some time, we recognised that it needed improvement.

To automate the conversion of the iPCU-output profile into a template, we implemented a custom `save()`

[15] `http://sit.sit.fraunhofer.de/studies/en/sc-iphone-passwords.pdf`

[16] `http://sit.sit.fraunhofer.de/studies/en/sc-iphone-passwords-faq.pdf`

method on the user group model. The method uses a regular expression match to replace any UUIDs found with the appropriate template variable, while keeping track of the substitutions so that subsequent uses of a particular UUID are substituted with the same variable, to preserve cross-references that may exist in the profile. This single change provides the required functionality, without needing to touch the Django admin interface, or provide a new user interface for template uploading.

While this improvement solves the problem for UUIDs in the profile, it doesn't address the other template variables used: username, password and email address. The simplest solution is to enter `$TCUSER`, `$TCPASS` and `$TCUSER@trinity.unimelb.edu.au` in the appropriate places when creating the profile in iPCU. While the resulting profile can't be tested directly in iPCU, it can be entered unchanged into iOS Configurator, thereby removing the remaining manual template creation step.

8 Lessons learned

We have ended up with a successful configuration tool for iOS devices, but not without encountering a few bumps on the road. Here are some of the lessons we have learned.

- Always continue to improve services; don't be content to stop after the first improvement.
- As well as functional testing, always include load testing in any service you deploy.
- But don't forget the functional testing! (We were completely surprised by the CNS detection issue and its impact on functionality.)

There was a big positive lesson however, which was how easy it was to set up a small web application with Django. Most system administrators would consider that development of scripts and their integration into day-to-day processes was a regular part of their job, and familiarity with one or more scripting languages was part of their sysadmin toolkit. However, I expect that relatively fewer would consider a web application framework such as Django as another tool available to them (unless their work is predominantly in support of websites or web services).

9 Further work

At the moment, when a user changes their password, the configuration profile needs to be replaced to update the password in the wireless and email settings. This is currently a separate step that users need to do. Ideally changing the password and updating the profile would be part of a single process, for example, by adding password changing functionality to the iOS Configurator, which would then provide a new configuration profile for download; however the user would need to remove the old profile before the new one could be installed, which complicates the process. Note that it wouldn't be possible to improve this with a custom iOS app, since apps do not have access to change settings (i.e. for email accounts) other than their own.

10 Acknowledgments

My thanks go to my former colleague, Mark Dorset, who drove the iPad project from the IT side, and Trent Anderson, who took over that responsibility. From the educational side, Glenn Jennings and Jennifer Mitchell kept us focussed on the user experience. Thanks are also due to the many staff and students who helped make the pilot program a success, leading to Trinity's adoption of iPads across all FS students.

Thanks are particularly due to the anonymous reviewers, whose comments have led to significant improvements in this paper, and to the shepherd, who provided valuable feedback on subsequent revisions.

A Declarative Approach to Automated Configuration

John A. Hewson
School of Informatics
University of Edinburgh
john.hewson@ed.ac.uk

Paul Anderson
School of Informatics
University of Edinburgh
dcspaul@ed.ac.uk

Andrew D. Gordon
Microsoft Research &
School of Informatics
University of Edinburgh
adg@microsoft.com

Abstract

System administrators increasingly use declarative, object-oriented languages to configure their systems. Extending such systems with automated analysis and decision making is an area of active research. We introduce ConfSolve, an object-oriented declarative configuration language, in which logical constraints over a system can be specified. Verification, impact analysis or even the generation of valid configurations can then be performed, by translation to a Constraint Satisfaction Problem (CSP), which is solved with an off-the-shelf solver. We present a full definition of our language and its compilation process, and show that our implementation outperforms previous work utilising an SMT solver, while adding new features such as optimisation.

Keywords

configuration management, automation, constraints

1 Introduction

Configuration of large computing installations is increasingly performed by automated tools which make use of declarative, object-oriented languages [1, 2, 3]. These tools replace low-level scripts which describe the steps needed to achieve a given system state, with a high-level declarative model of the goal state of the system. Such tools can be used to configure workstations, servers, or network hardware, and have proven popular among administrators of large or complex sites.

However, such tools do not facilitate automated analyses, such as checking that a configuration is valid, or assigning values automatically given a set of constraints: instead it is left to the administrator to decide which configurations are possible. Indeed, this predominantly manual process is often subject to inefficiencies and errors [4].

There is a need for a general-purpose system configuration language which can be used to model a broad range of configuration problems involving complex declarative constraints, in an easy-to-use manner. Such a tool could be used to make or help make better decisions, in a number of possible scenarios:

1. Verification of a system configuration according to some model.
2. Impact-analysis of configuration changes.
3. Generating valid configurations from a model.
4. Optimising a configuration according to a model.

There has been recent interest in configuration tools which, given a model of a system, are able to perform analyses to aid the system administrator. Recent research into such systems has made use of off-the-shelf boolean satisfiability (SAT) solvers [5, 6] and constraint-logic-programming (CLP) systems [3]. These approaches are promising, but there still remains a need for a rigorously defined high-level modelling language which supports constraints, and provides sufficient expressivity to be of use in system configuration, while scaling to problems of a practical size.

We believe the constraint programming (CP) is a natural candidate for modelling and solving such problems. In this paper we present ConfSolve, a system configuration language, in which constraints over valid solutions may be specified, and valid concrete configurations may be either validated or generated via a constraint satisfaction problem (CSP) solver. ConfSolve aims to be a language general enough to describe a large range of configuration problems in a manner natural to a system administrator, without requiring expertise in a constraint modelling language, or expert help to construct models.

Contributions of the paper

1. Define a constraint-based object-oriented configuration language.
2. Define the translation of the language to a CSP (encoded in MiniZinc [7]).
3. Show that translated models can scale to problems of a useful size.
4. Demonstrate that the language can be used to model problems identified in previous work based on SMT, and scale to significantly larger problems.

Structure of the paper

The remainder of this paper is structured as follows: We introduce the ConfSolve language by means of example,

give an overview of the MiniZinc constraint language, present the abstract grammar of ConfSolve, describe its type system, and describe a method for its transformation into MiniZinc. Finally, we provide experimental results demonstrating that our method outperforms previous work, before discussing the implications and directions for future work.

2 Modelling with ConfSolve

ConfSolve provides the user with an object-oriented declarative language, with a Java-like syntax, which adheres to several key principles:

1. Order never matters. Declaration and usage can occur in any order with no difference in meaning.
2. Everything is an expression, except declarations.
3. All classes are equal: there are no built-in classes with special meanings such as Machine or File.

Variables and Classes: A ConfSolve model consists of a global scope in which strongly-typed variables, classes, and enumerations may be declared. For example, a simple machine may be defined as:

```
enum OperatingSystem { Windows, UNIX, OSX }

class Machine {
  var os as OperatingSystem;
  var cpus as 1..4;
  var memory as int;
}

var m0 as Machine;
```

In which `m0` is a Machine object in the global scope, with members `os`, an enumeration; `cpus`, an integer subrange; and `memory`, an unbounded integer.

Member variables may also declare objects, allowing the nesting of child objects within a parent object. For example, we could add a network interface to the machine definition:

```
class Machine {
  ...
  var en0 as NetworkInterface;
}

class NetworkInterface {
  var subnet as 0..3;
}
```

An instance of NetworkInterface will be created whenever a Machine is instantiated. The lifetime of the NetworkInterface instance is tied to that of its parent object, and is not shared between different instances of Machine.

Inheritance: Objects support classical single inheritance via abstract classes. For example, we declare a class model machine-roles, with specialised subclasses for web servers:

```
abstract class Role {
  var machine as ref Machine;
}

class WebServer extends Role {
  var port as 0..65535;
}
```

References: Associations between objects are modelled using reference types. References are handles to objects elsewhere in the model, which cannot be null. Consider an instance of the web server role:

```
var ws1 as WebServer;
```

In the previous declaration of the Role class, the variable `machine` was declared as a Machine reference. Thus `w1` contains a reference to a machine, in this case it will refer to `m0`, as it is the only machine we have so far declared. The solver will automatically assign the value of a reference to any instance of the appropriate type, so if we always wanted `ws1` to run on `m1` we would also need to write:

```
ws1.machine = m1;
```

Which is an example of an equality constraint.

Constraints: Constraints are expressions which must hold in any solution to the model. For example, introducing a database-server role which can be either a slave or master, and must be peered with another slave or master, as appropriate:

```
enum DatabaseRole { Master, Slave }

class DatabaseServer extends Role {
  var role as DatabaseRole;

  // slave or master
  var peer as ref DatabaseServer;

  // the peer cannot be itself
  peer != this;

  // a master's peer must be a slave,
  // and a slave's peer must be a master
  role != peer.role;
}
```

This allow us to define two database server roles:

```
var masterDB as DatabaseServer;
masterDB.role = DatabaseRole.Master;
masterDB.peer = slaveDB;

var slaveDB as DatabaseServer;
slaveDB.role = DatabaseRole.Slave;
slaveDB.peer = masterDB;
```

Likewise we may define logical boot-disks on a SAN for each physical machine, and assign logical boot-disks to the two roles:

```
var db_disk as LogicalDisk;
db_disk.capacityGB = 2048;

var web_disk as LogicalDisk;
web_disk.capacityGB = 10;
```

Udating the declarations of Machine, WebServer and DatabaseServer with:

```
class Machine {
  ...
  var bootDisk as ref LogicalDisk;
}

class WebServer extends Role {
  ...
  machine.bootDisk = web_disk;
}

class DatabaseServer extends Role {
  ...
  machine.bootDisk = db_disk;
}
```

Sets and Quantifiers: Sets of variables may be declared, for example 10 web servers:

```
var webServers as WebServer[10];
```

A quantified constraint over the members of `webServers` can ensure that each server's port is set to 80, as long as the role is not running on `m0`:

```
forall ws in webServers where ws.machine != m0 {
  ws.port = 80;
};
```

As the port of `m0` is not constrained, the solver is free to choose its value. Should we want to specify it ourselves, we could write:

```
m0.port = 443;
```

So far our model contains only one Machine, `m0`, let's declare a class to describe a rack of 24 machines:

```
class Rack {
  var machines as Machine[24];
}

var r1 as Rack;
var r2 as Rack;
```

Here `machines` is declared as a set of objects, 24 new instances of Machine will be created as children of each Rack instance, in this case `r1`.

Given the following constraints, which place the master and slave databases in different racks:

```
masterDB.machine in r1;
slaveDB.machine in r2;
```

If rack `r2` fails, is there a valid solution? The answer is clearly no, and we can perform a quick impact analysis of such a failure by simply commenting out `r2`. Alternatively we could modify the definition of a Role to be:

```
abstract class Role {
  var machine as ref Machine;
  machine in r2.machines = false;
}
```

In either case ConfSolve will report that there is no valid solution to the model.

Optimisation Minimisation and maximisation constraints may be used for any solver-populated variable. The solver will find not just a valid value, but an optimal value, given some expression to be maximised or minimised. For example, if we prefer database masters and slaves to be in different racks, but this is not a hard constraint, then we can remove the constraints:

```
masterDB.machine in r1;
slaveDB.machine in r2;
```

Replacing them with a constraint maximising the number of machines with peers on different racks:

```
var databases as ref DatabaseServer[2];
var racks as ref Rack[2];

maximize sum r in racks {
  count (db in databases
    where db.machine in r.machines
      != db.peer.machine in r.machines);
};
```

Output The final output of the ConfSolve compiler, once solving is complete, is an object-tree in a format similar to the popular JSON (JavaScript Object Notation) format, which we call CSON (ConfSolve Output Notation). We describe CSON in full in Section 6.3. For example, the CSON corresponding to a model containing only `m0` is as follows:

```
{
  m0: Machine {
    os: OperatingSystem.UNIX,
    cpus: int 4,
    memory: int 1024,
    en0: NetworkInterface {
      subnet: 0,
    },
    bootDisk: ref LogicalDisk web_disk,
  },
  web_disk: LogicalDisk {
    capacityGB: int 10,
  },
}
```

3 Core Syntax of ConfSolve

This section describes the abstract grammar of ConfSolve, which is independent of the concrete grammar which we chose for our implementation, and does not include concrete syntax such as semicolons, comments, or whitespace rules. This provides a concise description of the language, free from unnecessary detail. It also allows others to use their own concrete syntax, but adopt the ConfSolve abstract syntax tree (AST) in order to apply the same translation steps to target MiniZinc.

To avoid redundancy, we first define a minimal core language, and then a series of derived constructs which are defined in terms of the core language.

Syntax of Types:

$S ::= \{i_1, \ldots, i_n\}$	integer subset
$T ::=$	type
bool	boolean
int	integer
S	integer subset
u	enumeration
c	object
$T[]$	set of T
$c[n]$	set of objects, with cardinality n

Identifiers are represented by metavariables: c is a class name, v is a variable name, u is an enum name, a_i is an enum member, l is a field name; i, m and n are integers; and b is a boolean: **true** or **false**.

A ConfSolve type is either a boolean, and unbounded integer, an finite subset of integers, an enumeration, and object, a set of any of those, or a set of objects with a fixed cardinality. Nesting of set types is not supported, as there is no direct way to represent sets of sets in MiniZinc, however an object may contain a set field, which may itself contain objects, giving the user the means to model arbitrary nesting via objects. Variable declarations may not be of type $c[]$, which is reserved for use during type checking (see section 4).

Integers are the only unbounded type in ConfSolve, as is the case in MiniZinc. Consequently a set of **int** cannot be declared, a restriction which we formally impose when we describe the type system in section 4. This restriction stems from the fact that quantifiers are unrolled as part of the MiniZinc to FlatZinc compilation process, which is not possible when the domain of the quantifier is infinite. As it is usually undesirable to have unbounded models, it is worth observing that the benefit of the **int** type is that it allows constants whose domain is not known to be declared and assigned separately. Thus one can define **var** *id* **as int** and later write the constraint $id = 4$. It also allows the user to avoid having to specify the domain of functionally defined variable values which ultimately depend on only variables with finite domains, for example the domain of $x = 5 * y + 3$, where y is a constant defined elsewhere.

To reduce the complexity of the MiniZinc encoding, sets of objects $c[n]$ have the same upper and lower bound n on their cardinality. As an alternative, the user may instead use a fixed cardinality set of objects, and a variable cardinality set of references with a constraint that the latter must be resolved to only members of the former. The derived expressions in Section 3.1 address the declaration of such fixed-cardinality sets.

Syntax of Expressions:

$e ::=$	expression
this	current object
v	variable
$e.l$	field access
$u.a$	enum member
e.**size**	set cardinality
e_1 BinOp e_2	binary operator
Fold (v **in** e_1 **where** e_2) (e_3)	fold
bool2int(e)	cast bool to int
-e	negation
!e	logical not
$[e_1, \ldots, e_n]$	set literal
b	boolean literal
i	integer literal
(e)	parenthesis
Fold ::=	fold operator
forall | **exists**	quantification
sum	summation

Variables, constants, binary and unary operators, and parenthetical expressions are defined in the standard manner. Object field access $e.l$ evaluates to the field l of

object e. Enum constants are written in a fully-qualified manner as $u.a$, where u is the name of the enumeration and a is a constituent member. The current object can be accessed via **this** within the body of a ClassDecl. For expressions with set-type, e.**size** evaluates to the cardinality of the set given by e. Three folds over sets are defined: universal quantification, **forall**, existential quantification, **exists**, and summation, **sum**. Folds include a **where** expression which filters the set prior to evaluating the fold. Finally, the function **bool2int** provides type-casting between boolean and integer types.

Binary Operators:

BinOp ::=	binary operator
= | > | >= | < | <= | **in** | **subset**	relational
union | **intersection**	set
&& | || | -> | <->	logical
+ | - | / | * | ^ | **mod**	arithmetic

Relational operators use the standard C-like notation, with the addition of **in** which is the set membership operator $\in$, and **subset** which is the subset operator $\subset$. Logical operators are *and*, *or*, *implies*, and *biconditional*. Arithmetic operators are standard, where ^ is exponentiation, and **mod** is modulo.

Syntax of Models:

Model ::=	model
Declaration*	declarations
Declaration ::=	declaration
ClassDecl	class decl.
EnumDecl	enum decl.
VarDecl	var decl.
Constraint	constraint
ClassDecl ::=	class decl.
abstract? **class** c **extends** c' { (VarDecl | Constraint)* }	
EnumDecl ::=	enum decl.
enum u $\{a_1, \ldots, a_n\}$	
VarDecl ::=	variable decl.
var v **as** T	
var v **as ref** c	object reference
Constraint ::=	constraint
e	hard constraint
maximize e	soft constraint

Identifiers are represented by metavariables: c is a class name, v is a variable name, u is an enum name, a_i is an enum member, l is a field name; i, m and n are integers.

A model consists of a series of declarations, of either classes, enumerations, variables, or constraints. A class declaration may contain any number of nested variable or constraint declarations, and it may extend another class.

A declaration **class** c **extends** c' is well-formed if and only if, there is a well-formed class declaration for c' and the inheritance hierarchy is acyclic, or if c' is the top class, denoted by the distinguished name $\varnothing$. A well-formed class may contain duplicate field names.

Enumerations consist of a name and a non-empty a set of identifiers, which defines its members. Variables are always declared with a type T. Object reference variables may be declared using **var** v **as ref** c. This creates a reference which will resolve at solve-time to an instance of c elsewhere in the model, whereas the declaration **var** v **as** c allocates a new instance of c.

3.1 Derived Syntax

The grammar above describes the core of the ConfSolve language. The full language contains a number of constructs which are derived from the core language, to keep the core and its translation as simple as possible. These syntactic re-write rules are performed by the parser in our implementation. The relation $\triangleq$ means "equal by definition".

Derived Declarations:

class c { (VarDecl | Constraint)* } $\triangleq$
  class c **extends** $\varnothing$ { (VarDecl | Constraint)* }

var v **as** m **..** n $\triangleq$
  var v **as** $\{x \mid x \in \mathbb{N},\ x \geq m \wedge x \leq n\}$

var v **as** $T[m$ **..** $n]$ $\triangleq$
  var v **as** $T[]$
  v**.size >=** m **&&** v**.size <=** n

minimize $e \triangleq$ **maximize** $-e$

Classes without a base type extend the top class, denoted by the distinguished name $\varnothing$. Integer subsets may be declared as ranges. Variable cardinality sets are given a shorthand notation. Minimisation is defined as negated maximisation.

Derived Expressions:

Fold (x **in** e_1) (e_2) $\triangleq$
  Fold (x **in** e_1 **where true**) (e_2)

count (x **in** e_1 **where** e_2) $\triangleq$
  sum (x **in** e_1 **where** e_2) (1)

count (x **in** e_1) $\triangleq$
  count (x **in** e_1 **where true**)

$e_1 != e_2 \triangleq !(e_1 = e_2)$

$\{e_1;\ldots;e_n\} \triangleq (e_1 \wedge \cdots \wedge e_n)$

Quantifiers without filters are defined as having an always-true filter. The body of a **forall** expression is defined as the logical conjunction of its sub-expressions. The **count** expression is defined in terms of summation. The "not equals" operator is defined as negated equality. Finally, a semicolon-delimited expression block is defined as the conjunction of its sub-expressions.

4 Type System

From the information presented so far, we are able to recognise a syntactically correct ConfSolve program. However, not all syntactically correct ConfSolve programs are well-formed. For example, a program which compares booleans with integers, declares a set of **int**, or makes use of undeclared variables. To complete our description of ConfSolve, it is necessary to describe its type system.

Static typing serves two purposes, firstly to provide a level of compile-time safety, and secondly to satisfy the CSP solver's requirement that a domain is specified for each variable. The smaller the domain, the better performance one can expect from the solver. This is why the type 1..3 is more desirable than the type **int**.

We formally specify ConfSolve's type system as a *proof system*, which is a declarative specification of the rules governing the assignment of types to expressions. This is separate from the actual type checking algorithm used in the ConfSolve compiler which implements these rules.

Within a proof system types are assigned to expressions via typing *judgements*, which are applied recursively, and take the form:

$$\frac{\textit{premises}}{\textit{conclusion}}\ (\text{Name})$$

where premises may be written on multiple lines or separated with a space. Judgements which contain expressions require a *typing environment* to resolve variable names to their declared type, which is similar to the concept of a symbol table, used when implementing such systems.

Environments:

$E ::= v_1 : T_1, \ldots, v_n : T_n$	type environments
$dom(v_1 : T_1, \ldots, v_n : T_n) = \{v_1, \ldots v_n\}$	environment domain

We now begin the formal specification. Type system judgements may be made with respect to a typing environment E, of the form $v_1 : T_1, \ldots, v_n : T_n$, which assigns a type to each in-scope variable. We write $\varnothing$ for the initial environment with an empty map.

Typing Judgements:

$E \vdash \diamond$	environment E is well-formed
$\vdash T$	the type T is well-formed
$T <: T'$	type T is a subtype of T'
$E \vdash e : T$	in E, expression e has type T

There are four judgements which we may make. That an environment is well formed, that a type is well-formed, that a type is a subtype of another, and that an expression has a given type.

Rules of Well-Formed Environments and Types:

Where **class** c **extends** $c'\{\ldots\}$ means that there exists such a declaration. Likewise for **enum** $u\,\{a_i{}^{i\in 1..n}\}$.

(Env Empty) $\dfrac{}{\varnothing \vdash \diamond}$

(Env Var) $\dfrac{E \vdash \diamond \quad \vdash T \quad v \notin dom(E)}{E, v : T \vdash \diamond}$

(Type Bool) $\dfrac{}{\vdash \textbf{bool}}$

(Type Int) $\dfrac{}{\vdash \textbf{int}}$

(Type Int Sub) $\dfrac{}{\vdash S}$

(Type Enum) $\dfrac{\textbf{enum}\ u\,\{a_i{}^{i\in 1..n}\}}{\vdash u}$

(Type Obj) $\dfrac{c' \neq \varnothing \rightarrow \vdash c' \quad \textbf{class}\ c\ \textbf{extends}\ c'\,\{\ldots\}}{\vdash c}$

(Type Set) $\dfrac{\vdash T \quad T \neq \textbf{int}}{\vdash T[]}$

(Type Obj Set) $\dfrac{\vdash c}{\vdash c[n]}$

A well-formed environment is either empty, or contains a mapping of variable names to types. A well-formed type is either a bool, and int, an enum, an integer subset, an object, a set of any type other than **int**, or a set of objects with a fixed cardinality. These rules and those which follow make use of definitions introduced in the syntax of types and expressions in section 3.

Rules of Subtyping

(Extends) $\dfrac{\textbf{class}\ c\ \textbf{extends}\ c'\,\{\ldots\}}{c <: c'}$

(Reflex) $\dfrac{\vdash T}{T <: T}$

(Trans) $\dfrac{T <: T' \quad T' <: T''}{T <: T''}$

(Set Subtype) $\dfrac{T <: T'}{T[] <: T'[]}$

(Obj n-Set Subtype) $\dfrac{c <: c'}{c[n] <: c'[n]}$

(Obj Set Subtype) $\dfrac{\vdash c}{c[n] <: c[]}$

(Int Sub) $\dfrac{}{S <: \textbf{int}}$

(Int Sub Union) $\dfrac{}{S <: S \cup S'}$

Our rules of type assignment make heavy use of subtyping for all types, not just objects, in order to make the type-assignment rules simpler. The first three rules define the familiar rules of class-based inheritance, reflexivity (that a type is a subtype of itself) and transitivity (that a subtype is a subtype of any of its supertypes). The next three rules extend this notion to sets. The final two rules define integer subsets as a subtype of **int**, and that an integer subset is a subtype of the union between that subset and another subset. Thus $\{1,2\} <:$ **int**, the purpose of which will be explained shortly.

With all of the pre-requisites in place, we can finally present the rules of type assignment for expressions:

Rules of Type Assignment: $E \vdash e : T$

(Subsum)
$$\frac{E \vdash e : T \quad T <: T'}{E \vdash e : T'}$$

(Var)
$$\frac{E \vdash \diamond \quad (v : T) \in E}{E \vdash v : T}$$

(Bool Const)
$$\frac{E \vdash \diamond}{E \vdash b : \textbf{bool}}$$

(Int Const)
$$\frac{E \vdash \diamond}{E \vdash i : \{i\}}$$

(Enum Const)
$$\frac{E \vdash \diamond}{E \vdash u.a : u}$$

(Set)
$$\frac{E \vdash e_i : T \quad \forall i \in 1..n}{E \vdash [e_1, \ldots, e_n] : T[]}$$

(Eq)
$$\frac{E \vdash e_1 : T \quad E \vdash e_2 : T}{E \vdash e_1 = e_2 : \textbf{bool}}$$

(Ineq Op)
$$\frac{\oplus \in \{>, >=, <, <=\} \quad E \vdash e_1 : \textbf{int} \quad E \vdash e_2 : \textbf{int}}{E \vdash e_1 \oplus e_2 : \textbf{bool}}$$

(In Op)
$$\frac{E \vdash e_1 : T \quad E \vdash e_2 : T[]}{E \vdash e_1 \textbf{ in } e_2 : \textbf{bool}}$$

(Subset Op)
$$\frac{E \vdash e_1 : T[] \quad E \vdash e_2 : T[]}{E \vdash e_1 \textbf{ subset } e_2 : \textbf{bool}}$$

(Logical Op)
$$\frac{\oplus \in \text{logical} \quad E \vdash e_1 : \textbf{bool} \quad E \vdash e_2 : \textbf{bool}}{E \vdash e_1 \oplus e_2 : \textbf{bool}}$$

(Set Op)
$$\frac{\oplus \in \{\textbf{union}, \textbf{intersection}\} \quad E \vdash e_1 : T[] \quad E \vdash e_2 : T[]}{E \vdash e_1 \oplus e_2 : T[]}$$

(Int Sub Set Op)
$$\frac{\oplus \in \{\textbf{union}, \textbf{intersection}\} \quad E \vdash e_1 : S_1[] \quad E \vdash e_2 : S_2[]}{E \vdash e_1 \oplus e_2 : (S_1 \oplus S_2)[]}$$

(Arith Op Int)
$$\frac{\oplus \in \text{arithmetic} \quad E \vdash e_1 : \textbf{int} \quad E \vdash e_2 : \textbf{int}}{E \vdash e_1 \oplus e_2 : \textbf{int}}$$

(Arith Op Int Sub)
$$\frac{\oplus \in \text{arithmetic} \quad E \vdash e_1 : S_1 \quad E \vdash e_2 : S_2}{E \vdash e_1 \oplus e_2 : \{x_1 \oplus x_2 \mid x_1 \in S_1,\ x_2 \in S_2\}}$$

(Dot)
$$\frac{E \vdash e : c \quad c <: c' \quad j \in 1..n \quad \textbf{class } c \textbf{ extends } c' \{\textbf{ var } l_i \textbf{ as } T_i^{i \in 1..n}\}}{E \vdash e.l_j : T_j}$$

(This)
$$\frac{E \vdash \diamond}{E \vdash \textbf{this}_c : c}$$

(Set Card)
$$\frac{E \vdash e : T[]}{E \vdash e.\textbf{size} : \textbf{int}}$$

(Channel)
$$\frac{E \vdash e : \textbf{int}}{E \vdash \textbf{int2bool}(e) : \textbf{bool}}$$

(Quant)
$$\frac{Q \in \{\textbf{forall}, \textbf{exists}\} \quad E \vdash e_1 : T[] \quad E, v : T \vdash e_2 : \textbf{bool} \quad E, v : T \vdash e_3 : \textbf{bool}}{E \vdash Q\ v \textbf{ in } e_1 \textbf{ where } e_2\ (e_3) : \textbf{bool}}$$

(Sum)
$$\frac{E \vdash e_1 : T[] \quad E, v : T \vdash e_2 : \textbf{bool} \quad E, v : T \vdash e_3 : \textbf{int}}{E \vdash \textbf{sum } v \textbf{ in } e_1 \textbf{ where } e_2\ (e_3) : \textbf{int}}$$

The first rule is that of subsumption, that an expression of a type may also take any of its supertypes. This is followed by variable resolution in the environment, and boolean integer and enumeration constants. The type of an integer constant is a singleton set containing the constant's value. This is significant, because ConfSolve forbids sets of integers, thus the set literal $[1,2,3]$ is legal in ConfSolve, having type $\{1\} \cup \{2\} \cup \{3\} = \{1,2,3\}$. Indeed, it is the reason why the integer literal 1 has type $\{1\}$, and why the type system makes an effort not to promote to integer subsets to **int** too readily.

The rules for comparisons (Eq)–(Logical Op) are relatively straightforward, and make heavy use of the subtyping rules. For example, recall that S is a subtype of **int** and is this subject to the rule (Ineq Op). Likewise, when we state that both expressions in the equality (Eq) rule must be of type T, that is not to say that they are of the same subtype, only that for both types there exists a common supertype for which they may be substituted.

The set operation (Set Op) follows a similar form, but (Int Sub Set Op) provides a specialised rule for handling integer subsets, which applies the intersection or union operator to the subset itself, as appropriate. The arithmetic operations (Arith Op Int) and (Arith Op Int Sub) follow this same pattern, with the latter being somewhat unusual. Namely, that for arithmetic operations between integer subsets, the resulting type is the integer subset

containing the result of the application of the operation to all pairs in the two source subsets. The motivation for this is the same as for the (Int Const) rule, that the expression $1+2$ has type 3, and thus the set literal $[1+2]$ is legal.

The next four rules, from (Dot) to (Channel) specify types for member variable access, the **this** pseudo-variable, set cardinality, and channeling integers to booleans.

The final two rules specify types for quantification and summation expressions, which are the only rules which introduce variables into the environment, *i.e.*, the scope of v is e_2 and e_3, the **where** and body clauses, respectively.

In order to provide a succinct notation, the system has the following derived properties: if $T <: T'$ then $\vdash T$ and $\vdash T'$; and if $E \vdash e : T$ then $E \vdash \diamond$ and $\vdash T$ and $\mathit{freevars}(e) \subseteq \mathit{dom}(E)$. That is, that the subtype judgement always involves well-formed types, and that any free variables in an expression are well-formed members of its environment.

5 ConfSolve and MiniZinc

In this section we provide an overview of the process of compiling and solving a ConfSolve model, and of the MiniZinc constraint modelling language.

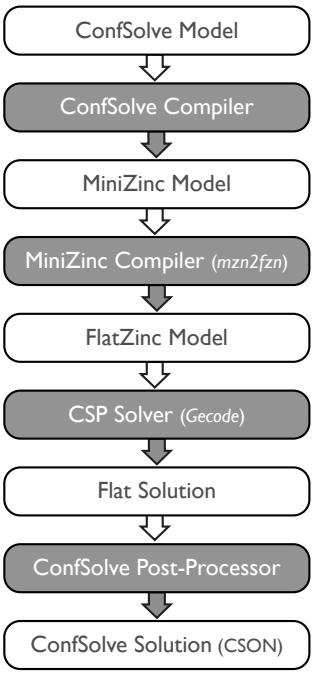


Figure 1: Compiling and solving a ConfSolve model. White boxes are files; shaded boxes are processes.

Compiling and solving a ConfSolve model requires several steps, which are illustrated in Figure 1; the steps are as follows:

1. The ConfSolve compiler is invoked; the model is translated into a CSP expressed in MiniZinc. This is described in Section 6.
2. The MiniZinc model is compiled into a FlatZinc model using *mzn2fzn* [7].
3. The FlatZinc model is solved using a constraint solver. In our implementation we use Gecode [8].
4. The solution found by the solver is parsed by the ConfSolve post-processor and combined with the original model to produce an object-tree (CSON) representing the solution. This is described in Section 6.3.

Implementation Our prototype implementation of the ConfSolve compiler and post-processor consists of approximately 1900 lines of F# / OCaml.

5.1 MiniZinc

MiniZinc [7] is a modelling language for constraint programming (CP) problems, in which models are specified as declarative constraints over sets of variables and domains. MiniZinc models are compiled into FlatZinc, which is a low-level input format supported by many CP solvers.

MiniZinc is a form of many-sorted first-order logic. Variables can be defined with integer, boolean, or set domains; arrays of variables may be declared; constraint expressions include quantification over sets, and logical implication. However, MiniZinc is restricted in its expressibility: sets are limited to bounded integer and boolean members (no sets of sets), arrays cannot contain other arrays and must be of fixed size, and quantifications are restricted to expressions with a constant value at compile-time. These restrictions arise from the fact that MiniZinc is designed to be a thin wrapper around the primitives supported by CP solvers.

Example Let us examine a MiniZinc model generated by the ConfSolve compiler. This model corresponds to the four instances of the DatabaseServer class from the example in Section 2, the relevant ConfSolve model for which is:

```
enum DatabaseRole { Master, Slave }

class DatabaseServer extends Role {
  var role as DatabaseRole;

  // slave or master
  var peer as ref DatabaseServer;

  // the peer cannot be itself
  peer != this;
```

```
  // a master's peer must be a slave,
  // and a slave's peer must be a master
  role != peer.role;
}

// instances
var masterDB as DatabaseServer;
masterDB.role = DatabaseRole.Master;
masterDB.peer = slaveDB;

var slaveDB as DatabaseServer;
slaveDB.role = DatabaseRole.Slave;
slaveDB.peer = masterDB;
```

The model begins with variable declarations. As MiniZinc does not support records or objects, each field in the DatabaseServer class is declared as a separate array, where the number of elements in the array is equal to the number of DatabaseServer instances in the model, in this case 4. The domain of the array elements correspond to the type of the field. In the following example the DatabaseServer_role has as its domain the contiguous set of integers 1..2, which are indexes into the DatabaseRole enumeration:

```
array[1..4] of var 1..2: DatabaseServer_role;
array[1..4] of var 1..4: DatabaseServer_peer;
```

Constraints are first-order expressions which restrict the values a variable may take from its domain. Each is a boolean expression which must evaluate to true. In this case, the constraints come from the DatabaseServer class, and so are wrapped with a `forall` expression which applies the constraint to all instances of DatabaseServer, and defines the value of the variable `this` to be an index into the field arrays for the DatabaseServer class. The identifier `this` has no special meaning in MiniZinc, and was chosen simply to facilitate translation:

```
constraint
 forall (this in 1..4) (
  DatabaseServer_peer[this] != this
 );

constraint
 forall (this in 1..4) (
  DatabaseServer_role[this] !=
   DatabaseServer_role[DatabaseServer_peer[this]]
 );
```

Each MiniZinc model must have a `solve` goal, which can be to either minimise or maximise an expression, or simply satisfy the constraints in the model. In this case the goal is the latter:

```
solve satisfy;
```

Finally, the model must specify which variables values will be output by the solver. This is useful when the model includes constants or intermediate variables, the values of which are not of interest. In this case, it is simply:

```
output [
 show(DatabaseServer_role),
 show(DatabaseServer_peer)
];
```

5.2 Constraint Satisfaction

The semantics of ConfSolve are defined in terms of MiniZinc. The specification of a MiniZinc problem is independent from how it will be solved, therefore ConfSolve does not make any formal guarantees about constraint satisfaction, such as the completeness of the search. Instead it is the solver which defines these properties.

There are many different approaches to CSP solving with different performance characteristics and different formal guarantees. In this paper we make use of the Gecode [8] solver which performs a global search via the classic CSP algorithm, constraint propagation. A global search is complete when variables have finite domains, but the **int** type in both ConfSolve and MiniZinc has an infinite domain. This means that any ConfSolve model which contains decision variables of type **int** will result in an incomplete search. In practice, the CSP solver uses a bound such as 32 bits for an integer, which is of course a finite domain, and the search within this range is complete. The Gecode solver which we use has a bound of 32-bits for integers.

The practical implications of search completeness are straightforward; if we limit ourselves to bounded variables then solvers exist which guarantee that if there is a solution, then it will be found. However, there is no guarantee that we will have enough time or memory to conclude the search. We evaluate some examples of this in section 7.

6 Translating ConfSolve to MiniZinc

This section defines the translation from a ConfSolve model to MiniZinc in terms of their abstract grammars. The translation occurs in two phases: a counting phase in which indexes are generated for each object, and an upper-bound on the number of object instances in the model is calculated; and a translation phase in which a MiniZinc abstract syntax tree is constructed.

6.1 Static Allocation

The static allocation phase determines the upper bound on the number of instances of each class, assigns each object an index, and records which indices are assigned

to the subclasses of a given class. Its purpose is to generate the following two data structures, for use in the later translation phase:

count is a map from a class names c to an integer representing the count of the number of instances of c in the model.

indices is a map from a class name c to a set of integers representing the indices of each instance of c or one of its subclasses.

The values of *count* and *indices* are updated incrementally as counting progresses via the method described below. In order to handle inheritance we introduce the concept of a root class, that is, the topmost class in any given hierarchy. Formally, $root(c)$ is c when the superclass of c is $\varnothing$, otherwise it is the topmost superclass of c. The process of counting is as follows:

Given the definition **class** c **extends** c', for each global declaration **var** v **as** T:

when $T = c$, $count(root(c))$ is incremented, and its value is added to the sets $indices(c)$, and to $indices(c^*)$ for each ancestor c^* which c extends. This process is then repeated for each field **var** v **as** T declared in class c or any ancestor of c.

when $T = c[n]$, the case for $T = c$ is repeated n times.

6.2 Translation

The translation describes the process of generating a MiniZinc abstract syntax tree from a ConfSolve abstract syntax tree.

We make use of the notation $[\![x]\!]$ to mean "the translation of x", where x is some syntactic construct, such as a type or expression.

Translation of Types $[\![T]\!]$:

$$
\begin{aligned}
\textbf{int} &\triangleq \textbf{int} \\
\textbf{bool} &\triangleq \textbf{bool} \\
\{i_1,\ldots,i_n\} &\triangleq \{i_1,\ldots,i_n\} \\
u &\triangleq 1\,..\,num(u) \\
c &\triangleq \begin{cases} \{\, indices(c)\,\} & \text{if } c \text{ is abstract} \\ 1\,..\,count(c) & \text{otherwise} \end{cases} \\
c[n] &\triangleq \textbf{set of } [\![c]\!] \\
B[\,] &\triangleq \textbf{set of } [\![B]\!]
\end{aligned}
$$

Here we define $[\![T]\!]$ to be the MiniZinc translation of a ConfSolve type T, where $num(u)$ is the number of elements in enum u. Each ConfSolve type maps directly onto a MiniZinc type, with the exception of enumerations and objects which are translated to integer indices. For set types, the translation is recursive, but only to one level, because MiniZinc does not permit sets of sets. The translation process has not yet begun: the translation of a type is used as an intermediate step in the translations which follow.

Translation of Global Variable Declarations:

For each global declaration **var** v **as** T, introduce a declaration:

when $T = c[n]$

$[\![T]\!]$: v = {for $i \in 1..n, index(c)$}

when $T = c$

$[\![T]\!]$: v = $index(c)$

otherwise

var $[\![T]\!]$: v

For each global declaration **var** v **as ref** c, introduce a declaration:

var $[\![c]\!]$: v

The translation phase proceeds in a similar manner to the counting phase, and makes use of object indices generated in exactly the same manner as those in the *count* map. It is important that these indices are the same, so that an index corresponds to the correct value in the *indices* map. We define a function $index(c)$ which generates a new index of class $root(c)$ by counting, and returns its value.

Translation begins with global variable declarations, as shown above. When the type of the declared variable is a set of objects of class c with cardinality n, indices are generated for each of the n objects. When the type is a object of class c, a single index is generated. For all other types no values are assigned, and a **var** declaration is introduced.

Reference variable declarations are translated in the same manner as the "otherwise" case, that is a **var** declaration is introduced whose domain is the translation of the c type given in Translation of Types above.

Translation of Class-Level Variable Declarations:

For each ClassDecl defining a class c where $count(c) > 0$, containing fields **var** f_i **as** $T_i{}^{i \in 1..n}$, introduce an array for each field f_i:

when $T_i = c'[n]$

array [1 .. $count(c)$] **of** $[\![T_i]\!]$: c_f_i =
[for $i \in 1..count(c)$,
{ for $j \in 1..n, index(c')$ **}** **]**

when $T_i = c'$

array [1 .. $count(c)$] **of** $[\![T_i]\!]$: c_f_i =
[for $i \in 1..count(c), index(c')$ **]**

otherwise

array [1 .. $count(c)$] **of var** $[\![T_i]\!]$: c_f_i

Where class c contains fields **var** f_i **as ref** c_i $^{i \in 1..n}$, introduce an array for each field f_i:

array [1 .. $count(c)$] **of var** $[\![c_i]\!]$: c_f_i

Variable declarations nested within classes are translated by introducing an **array** which will contain the values of that field for all instances of some class c. When the type of the declared variable is a set of objects of class c' with cardinality n, an array of sets (the only nested construct permitted by MiniZinc) is declared, and for each of the $count(c)$ instances, n indices are generated. When the type is an object of class c' the declaration is simpler: an array of indices is introduced, one for each of the $count(c)$ instances. For all other types no values are assigned, and array of **var** is introduced.

Once again, reference variable declarations are translated in the same manner as the "otherwise" case, where a **var** declaration is introduced whose domain is the translation of the type c_i.

Translation of Global Constraints:

For each global constraint e, introduce a statement:

constraint $[\![e]\!]$

For each global constraint **maximize** e, update the objective expression o:

when o is undefined

$o = [\![e]\!]$

otherwise

$o = o + [\![e]\!]$

The translation of hard constraints consists of translating their expressions, which we discuss later. The objective expression o is the sum of every maximisation goal's expression, and is maintained throughout the translation phase. Each **maximize** constraint corresponds to a subexpression in the objective function.

Translation of Class-Level Constraints:

For each ClassDecl defining a class c where $count(c) > 0$, for each constraint e, introduce a statement:

constraint forall (this **in** 1 .. $count(c)$) ($[\![e]\!]$)

For each global constraint **maximize** e, update the objective expression o:

when o is undefined

$o = [\![e]\!]$

otherwise

$o = o +$ **sum** (this **in** 1 .. $count(c)$) ($[\![e]\!]$)

Constraints may also be declared at the class-level, in which case they apply to every instance of that class. The translation occurs in the same manner as for global constraints, except that the constraints are placed over all instances in aggregate via **forall** for hard constraints and **sum** for maximisation constraints.

Translation of Expressions $[\![e]\!]$:

$$
\begin{aligned}
v &\triangleq [\![v]\!] \\
\textbf{this} &\triangleq \text{this} \\
e.l &\triangleq classof(e)\_l[[\![e]\!]] \\
u.a &\triangleq eindex(u,a) \\
e_1 \text{ Op } e_2 &\triangleq [\![e_1]\!]\ [\![\text{Op}]\!]\ [\![e_2]\!] \\
e.\textbf{size} &\triangleq \textbf{card}([\![e]\!]) \\
\text{Fold } (v \textbf{ in } e_1 \textbf{ where } e_2)\ (e_3) &\triangleq \text{(see below)} \\
\textbf{bool2int}(e) &\triangleq \textbf{bool2int}([\![e]\!]) \\
-e &\triangleq -[\![e]\!] \\
!e &\triangleq \textbf{not } [\![e]\!] \\
[e_1, \ldots, e_n] &\triangleq \text{(see below)} \\
e_1 \hat{}\ e_2 &\triangleq [\![e_1]\!] \textbf{ pow } [\![e_2]\!] \\
\textbf{true} &\triangleq \textbf{true} \\
\textbf{false} &\triangleq \textbf{false} \\
i &\triangleq i
\end{aligned}
$$

The MiniZinc translation $[\![e]\!]$ of a ConfSolve expression e is given above, where $eindex(u,a)$ is the index of element a in the declaration **enum** u $\{a_1 \ldots a_n\}$, and $classof(e)$ is the identifier c when the type of e is an object of class c. Expressions are translated recursively, with the exception of literals.

The keyword **this** is translated into the identifier "this", which has no special meaning in MiniZinc. Instead, it is simply a quantified variable defined in the **forall** expression in the prior Translation of Class-Level Constraints section.

Translation of Variables $[\![v]\!]$:

Within the scope of class c, the translation $[\![v]\!]$ of a variable v is:

when v is declared in class $c' \in c^*$

c'_v[this]

otherwise

v

Variables in expressions are translated in one of two ways depending on their type. If the variable occurs within the scope of class c and was declared in c' which is either c or one of its ancestors, then the result is a lookup in the array corresponding to field v of the current instance (*i.e.*, **this**) of class c'. For example, where c' is DatabaseServer, and v is "role", the translation is:

```
DatabaseServer_role[this]
```

Otherwise, v is either a global variable or a quantified variable (from a fold), and translates directly to its identifier.

Translation of Folds:

The translation of a fold expression Fold (v **in** e_1 **where** e_2) (e_3) is given by:

Fold ($[\![v]\!]$ **in** r) (b)

where $r \triangleq$

when e_1 is of type $c[n]$

1 **..** $count(c)$

when e_1 is of type $u[]$

1 **..** $num(u)$

when e_1 is of type $\{i_1, \ldots, i_n\}[]$

$\{i_1, \ldots, i_n\}$

when e_1 is of type **bool**$[]$

$\{$**true**, **false**$\}$

and $b \triangleq$

when Fold = **sum**

bool2int(v **in** $[\![e_1]\!]$ `/\` $[\![e_2]\!]$) * $[\![e_3]\!]$

otherwise

v **in** $[\![e_1]\!]$ `/\` $[\![e_2]\!]$ `->` ($[\![e_3]\!]$)

Folds such as **foreach** translate to a similar construct in MiniZinc, but one in which the fold must be over a set of constant value, because the MiniZinc compiler unrolls the fold at compile-time. Therefore, the translation consists of a fold of an expression body b over a constant range r, which need not be contiguous.

The range r depends on the type of the expression e_1, which the fold is over. When it is a set of objects of type c, the range is the indices of all c instances. When the type is an enum, it is the valid indices of the enumeration. When the type is an integer subset the range is that subset.

Ranges are larger than one might expect. Because MiniZinc requires that ranges are constant, the range must contain all values of the relevant type, and we must correspondingly wrap the body expression e_3 with the implication, $v \in e_1 \Rightarrow e_3$, so that the constraint is placed over only members of the set in the current solution. In fact, because ConfSolve also allows a filter expression e_2, this becomes $v \in e_1 \wedge e_2 \Rightarrow e_3$.

The body expression b in fact takes two forms. For a **forall** or **exists**, the logical form just mentioned is used. For a **sum**, an arithmetic form is used: $bool2int(v \in e_1 \wedge e_2) \times e_3$, where *bool2int* returns a 0/1 value given a false/true boolean.

Translation of Binary Operators $[\![$BinOp$]\!]$**:**

`&&` $\triangleq$ `/\`

`||` $\triangleq$ `\/`

`/` $\triangleq$ **div**

BinOp$'$ $\triangleq$ BinOp$'$

Binary operators are directly translated to MiniZinc operators. BinOp$'$ denotes all operators not explicitly listed.

Reduction of Set Literal Expressions:

The reduction of a set literal expression $[e_1, \ldots, e_n]$ is given by:

In the current scope, insert the declaration:

var set__s **as** T

Where T is a well-formed type which satisfies the (Set) typing judgement in section 4 and s is a unique integer.

In the current scope assume the constraint:

constraint e_1 **in** set__s $\wedge \cdots \wedge e_n$ **in** set__s

Finally, the derived expression is:

$[e_1, \ldots, e_n] \triangleq$ set__s

The translation of set literal expression is defined in terms of a reduction to a variable and associated constraints at the ConfSolve level, which should be performed before any of the other transformation steps previously listed. This reduction is necessary as it allows variables to appear inside set literals, which would otherwise not be legal in MiniZinc.

Solve Statement:

when o is undefined

solve satisfy

otherwise

solve maximize o

The translation to MiniZinc concludes with the introduction of a solve statement, the purpose of which is to provide a criteria for the solver's search, which may be either a satisfaction of the constraints, or maximisation of an objective expression.

6.3 Solutions

After solving, the output of the ConfSolve post-processor is an object-tree, the syntax of which we refer to as CSON (ConfSolve Output Notation). A concrete example of CSON is given in Section 2

To obtain a solution, the translated MiniZinc is compiled into FlatZinc and solved using Gecode, which outputs assignments for each variable in a simple text-based format defined by FlatZinc. Generating a CSON tree from this text is straightforward: the steps of the translation process are repeated, but whenever a MiniZinc variable would be introduced, we instead read its value from the output file, and emit the corresponding CSON representation:

Syntax of CSON:

$V ::=$	value
i	integer
true | **false**	boolean
$u.a$	enum member
c {Member*}	object
ref Target	object reference
$T[n]\{V_1,\ldots,V_n\}$	set literal
Member ::=	member
$v : V$	variable name : value
Target ::=	target
v	variable
Target.l	field access
Target[i]	set access

Each CSON value corresponds to a type in ConfSolve. The solution output consists of a single anonymous object representing the global scope. Nested within this are values for each variable. In the special case of references, the value is the fully-qualified name of the target variable, in which members of sets may be accessed via index, for example $v[i].f$ resolves to the value of field f of the i^{th} element of the set v.

7 Evaluation

Our evaluation of ConfSolve aims to show that the system can be used to model a number of diverse configuration problems, and to successfully analyse them. Furthermore, we want to ensure that ConfSolve is able to perform adequately on large models.

The evaluation was performed on a machine with a 2GHz Intel Core i7 processor and 8GB of RAM, running Mac OS X version 10.8.1. We used the 64-bit MiniZinc to FlatZinc converter version 1.5.1 with the `--no-optimize` flag, and the 64-bit Gecode FlatZinc interpreter version 3.7.1.

7.1 Virtual Machine Placement

In this evaluation we use ConfSolve to generate an assignment of virtual machines to physical machines in an Infrastructure as a Service (IaaS) configuration. Each physical machine is identical, having 8 CPUs and 16GB or memory. Each virtual machine has variables representing its requirements on the physical machine resources. These declarations are as follows:

```
class Machine {
  var cpu as int;     // 1 unit = 1/2 core
  var memory as int;  // MB
  var disk as int;    // GB

  cpu = 16;           // 2x Quad Core
  memory = 16384;     // 16 GB
  disk = 2048;        // 2 TB
}

abstract class VM {
  var host as ref Machine;
  var disk as int;
  var cpu as int;
  var memory as int;
}
```

Virtual machines may be one of two sizes, large and small. Large machines have 4 CPU units, 3.5GB of memory, and 500GB of disk. Small machines have 1 CPU unit, 768MB of memory and 20GB of disk:

```
class SmallVM extends VM {
  cpu = 1;
  memory = 768;
  disk = 20;
}

class LargeVM extends VM {
  cpu = 4;
  memory = 3584;
  disk = 500;
}
```

The infrastructure consists of two racks of 48 physical machines, onto which we wish to allocate 350 small and 100 large virtual machines:

```
// physical machine instances
var rack1 as Machine[48];
var rack2 as Machine[48];

// virtual machine instances
var smallVMs as SmallVM[350];
var largeVMs as LargeVM[100];
```

We define a constraint on virtual machine placement, as otherwise there is nothing to prevent every virtual machine from having the same host:

```
var machines as ref Machine[96];
var vms as ref VM[450];

forall m in machines {
  sum vm in vms where vm.host = m {
    vm.cpu;
  } <= m.cpu;

  sum vm in vms where vm.host = m {
    r.memory;
  } <= m.memory;

  sum vm in vms where vm.host = m {
    r.disk;
  } <= m.disk;
};
```

This constraint states that for each physical machine, the sum of the required quantity of each resource over all virtual machines hosted on it, must be less than the quantity of that resource provided by the physical machine. In other words, that the virtual machines assigned do not, in aggregate, consume more resources than are available. This is repeated for the three resources, `cpu`, `memory`, and `disk`.

From this model, ConfSolve is able to automatically generate assignments of virtual machines to physical machines (PMs), by automatically finding values for the `host` variable of each `VM` instance.

The performance achieved when scaling the problem up to 750 virtual machines is shown in Table 1. The problem size was increased until the Gecode solver consumed all available free memory on our test machine, which was 4.5GB.

Problem	ConfSolve	Cauldron [5]
VM Allocation 4:2	151	1717
VM Allocation 8:2	165	2115
VM Allocation 16:4	177	3995
VM Allocation 17:5	194	-
VM Allocation 100:48	1485	-
VM Allocation 250:48	8288	-
VM Allocation 450:96	44,700	-
VM Allocation 550:96	58,758	-
VM Allocation 650:96	77,174	-
VM Allocation 750:96	94,536	-

Table 1: VM : PM Allocation run-time (milliseconds), averaged over three runs.

7.2 Cauldron Test Suite

ConfSolve uses a similar object-oriented language to Cauldron [5] (see Section 8), a policy-based design tool which is able to describe CIM [9] models. Cauldron is able to generate solutions to these constraint-based system designs; an example of configuring an enterprise server with physical partitions is provided in [5].

HP Labs kindly provided us with a copy of the Cauldron binary (version `rel.10c`) and a number of sample problems from their test suite. The example described at length in [5] is very much representative in terms of scope and size, of the examples provided to us by HP. We translated a representative subset of these problems into ConfSolve models in order to confirm that ConfSolve can represent existing constraint-based configuration models. We then benchmarked the solution time of these equivalent ConfSolve and Cauldron models, the results of which can be found in Table 2. ConfSolve consistently out-performs Cauldron by a factor of around four on the test hardware described at the beginning of this section. The build of Cauldron which we were provided with makes use of a private build of the VeriFun [10] theorem prover in conjunction with a custom SAT solver which uses the LazySAT [11] algorithm. Given that Cauldron dates from 2007 and is no longer under development, more recent SAT solvers may provide a performance improvement, but Cauldron is a "black box" and we have no way to determine if this is the case.

As the Cauldron sample problems are relatively small in terms of search-space, we translated our large-scale VM example into an equivalent Cauldron model in order to compare performance at scale. The results in Table 1 show that, for this example at least, Cauldron does not scale to practical problem sizes, failing when only 17 virtual machines are to be allocated to 5 physical machines. ConfSolve was able to scale to 750 virtual machines.

Problem	ConfSolve	Cauldron [5]
Geometry	93	327
Firewall	145	504
QM4	394	2077
ServerComplex7	652	3149
ServerComplex10	875	4254

Table 2: Cauldron test suite run-time (milliseconds), averaged over three runs

8 Related Work

ConfSolve is, to the best of our knowledge, the first declarative configuration language to target a CSP solver (Cauldron targets a SAT solver). CSP is well suited to solving finite combinatorial problems, and branch-and-bound optimisation, which makes it a natural match for configuration problems.

Non-declarative policy languages with event-

condition-action (ECA) semantics, such as Ponder, have been used to describe network configuration problems [12], however the success of declarative system configuration tools such as Cfengine [1] has led to an interest in producing declarative policy tools for system administrators. Couch & Glifix were early experimenters with using Prolog for this purpose [13], followed by Narain [14], and more recently Yin [15]. Couch & Glifix conclude that a declarative language is essential for producing a convergent policy, due to the fact that an ECA language would require a policy to handle every possible failure scenario, but that Prolog is not a suitable language for system administration.

SAT solvers were used by Narain to solve network configuration problems [16, 6], via means of the the Alloy [17] modelling system and Kodkod [18] SAT-based relational logic solver. However, in both cases performance issues were encountered with the conversion of their model to SAT, and the difficulty of modelling was high: writing new predicates required expert knowledge of their modelling system's internals.

The Alloy Analyzer [17] is a mature modelling system which shares some commonality with ConfSolve: both provide an object-oriented specification language with logical constraints, and both require the user to specify an upper-bound on the number of objects in the search space. The Alloy modelling language is significantly more general than that of ConfSolve: modelling the dynamic properties of a system, *i.e.* its states, is a key feature, and its modelling language is necessarily far more general than ConfSolve, which is concerned only with the goal state of a system. As a system for model checking and verification, Alloy does not support optimisation of models, unlike MiniZinc, and in turn ConfSolve. This makes Alloy unsuitable as a backend for ConfSolve.

A notable SAT-based work is Cauldron [5], an object-oriented configuration language based on the CIM [9] model of classes, object references, and arrays, which is similar to the model adopted by ConfSolve. Solutions are generated using the VerifFun theorem prover, which itself relies on a SAT solver. Unfortunately, the Cauldron language is not rigorously defined, whether or not its search is complete is unclear, and its translation to SAT is not disclosed. Furthermore, its implementation does not scale well to practically-sized problems, such as those in Section 7.1.

PoDIM [19] is a framework for the Eiffel language which allows constraints on objects to be described with a SQL-like syntax. The language lacks a clear definition beyond its Eiffel implementation, which does not scale to problems of practical size.

An extension of the SmartFrog configuration language with constraints is hypothesised in [3], though no detailed work has been published.

s-COMMA [20] is a general-purpose object-oriented CP language which directly targets Gecode and other CP solvers, however it does not provide object references, variable cardinality sets or arrays, or quantification over decision variables.

Work towards developing ConfSolve is discussed in the workshop paper [21], which outlines an earlier, less efficient MiniZinc encoding.

9 Conclusion and Future Work

We have developed an object-oriented system configuration language in which constraints are used to specify valid configurations. We define its translation to the standard constraint modelling language MiniZinc, and find solutions to these models using a state-of-the-art constraint solver. Writing a complex object-oriented model in ConfSolve is considerably simpler than implementing the corresponding problem directly in MiniZinc. While the scalability of any constraint-based model is problem-specific, we have shown that ConfSolve models are able to scale to problems of a practical size using a common use-case.

While ConfSolve allows the user to model and analyse systems, it does not replace existing declarative configuration systems such as Cfengine or Puppet, but instead provides a platform which in the future could be used to augment existing languages, construct new tools which suggest solutions to specific problems, or perform impact analyses of proposed configuration changes.

It is possible to solve constraint problems at very large scale by performing a local search, which only explores a subset of the search-space, but can solve constraint problems many orders of magnitude larger. Local-search solvers such as [22] may soon support modelling with MiniZinc-like expressivity, which would allow them to be readily targeted by ConfSolve.

Implementation The ConfSolve compiler (v0.6) is written in F# / OCaml and is available for download at `http://homepages.inf.ed.ac.uk/s0968244/confsolve`

Acknowledgements.

Thanks to Sharad Singhal at HP Labs for providing the Cauldron test suite and binaries. This work was funded by Microsoft Research through their European PhD Scholarship Programme.

References

[1] Burgess, M.: Cfengine: a site configuration engine. USENIX Computing systems **8**(3) (1995) 309–402

[2] Puppet Labs: Puppet (2008) Available from `http://www.puppetlabs.com/puppet/`.

[3] Goldsack, P., Guijarro, J., Loughran, S., Coles, A., Farrell, A., Lain, A., Murray, P., Toft, P.: The SmartFrog configuration management framework. SIGOPS Operating Systems Review **43** (January 2009) 16–25

[4] Oppenheimer, D., Ganapathi, A., Patterson, D.: Why do Internet services fail, and what can be done about it? In: 4th USENIX Symposium on Internet Technologies and Systems, USENIX Association (2003)

[5] Ramshaw, L., Sahai, A., Saxe, J., Singhal, S.: Cauldron: A policy-based design tool. In: 7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2006), IEEE Computer Society (2006) 113–122

[6] Narain, S., Levin, G., Malik, S., Kaul, V.: Declarative infrastructure configuration synthesis and debugging. Journal of Network and Systems Management **16**(3) (2008) 235–258

[7] Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), Springer (2007) 529–543

[8] Gecode Team: Gecode: Genetic constraint development environment (2006) Available from `http://www.gecode.org`.

[9] Distributed Management Task Force Inc.: Common information model (CIM) standards (2010) Available from `http://www.dmtf.org/standards/cim/`.

[10] Walther, C., Schweitzer, S.: About VeriFun. In: Proceedings of the 19th International Conference on Automated Deduction (CADE-19). Volume 2741 of Lecture Notes in Computer Science., Springer (2003) 322–327

[11] Singla, P., Domingos, P.: Memory-efficient inference in relational domains. In: Proceedings of the 21st national conference on Artificial intelligence. Volume 1 of AAAI'06., AAAI Press (2006) 488–493

[12] Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy specification language. In: Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks (POLICY '01). (2001) 18–38

[13] Couch, A., Gilfix, M.: It's elementary, dear Watson: applying logic programming to convergent system management processes. In: Proceedings of the 13th conference on Large Installation System Administration (LISA '99), USENIX Association (1999)

[14] Narain, S., Cheng, T., Coan, B., Kaul, V., Parmeswaran, K., Stephens, W.: Building autonomic systems via configuration. In: Proceedings of IEEE Autonomic Computing Workshop. (2003)

[15] Yin, Q., Cappos, J., Baumann, A., Roscoe, T.: Dependable self-hosting distributed systems using constraints. In: Proceedings of the Fourth Conference on Hot Topics in System Dependability, USENIX Association (2008) 11–11

[16] Narain, S.: Network configuration management via model finding. In: Proceedings of the 19th conference on Large Installation System Administration (LISA '05), USENIX Association (2005) 15

[17] Jackson, D.: Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM) **11**(2) (2002) 256–290

[18] Torlak, E., Jackson, D.: Kodkod: A relational model finder. Tools and Algorithms for the Construction and Analysis of Systems (2007) 632–647

[19] Delaet, T., Joosen, W.: PoDIM: A language for high-level configuration management. In: Proceedings of the 21st conference on Large Installation System Administration (LISA '07), USENIX Association (2007)

[20] Soto, R., Granvilliers, L.: On the pursuit of a standard language for object-oriented constraint modeling. (2008) 123–133

[21] Hewson, J., Anderson, P.: Modelling system administration problems with CSPs. In: Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef'11). (2011) 73–82

[22] Benoist, T., Estellon, B., Gardi, F., Megel, R., Nouioua, K.: Localsolver 1.x: a black-box local-search solver for 0-1 programming. 4OR: A Quarterly Journal of Operations Research (2011) 1–18

Preventing the Revealing of Online Passwords to Inappropriate Websites with LoginInspector

Chuan Yue *
University of Colorado at Colorado Springs
Department of Computer Science
Colorado Springs, CO 80918, USA
cyue@uccs.edu

Abstract

Modern Web browsers do not provide sufficient protection to prevent users from submitting their online passwords to inappropriate websites. As a result, users may accidentally reveal their passwords for high-security websites to inappropriate low-security websites or even phishing websites. In this paper, we address this limitation of modern browsers by proposing *LoginInspector*, a profiling-based warning mechanism. The key idea of LoginInspector is to continuously monitor a user's login actions and securely store hashed domain-specific successful login information to an in-browser database. Later on, whenever the user attempts to log into a website that does not have the corresponding successful login record, LoginInspector will warn and enable the user to make an informed decision on whether to really send this login information to the website. LoginInspector can also report users' insecure password practices to system administrators so that targeted training and technical assistance can be provided to vulnerable users. We implemented LoginInspector as a Firefox browser extension and evaluated it on 30 popular legitimate websites, 30 sample phishing websites, and one new phishing scam discovered by M86 Security Labs. Our evaluation and analysis indicate that LoginInspector is a secure and useful mechanism that can be easily integrated into modern Web browsers to complement their existing protection mechanisms. Security system administrators in our university commented that such a tool could be very helpful for them to strengthen campus IT security.

***K*eywords:** Web browser, password, security, phishing, JavaScript, Web, authentication

1 Introduction

Text passwords still occupy the dominant position in online user authentication [2, 16, 17, 23], and they provide an effective protection to our valuable online accounts. Password security heavily depends on creating strong passwords and protecting them from being stolen. However, in recent years, especially with the rampancy of Web-based malicious activities [31, 57], passwords have increasingly been targeted by various harvesting or stealing attacks. For example, one of the most severe threats to online users is the phishing attack [6, 8, 10, 11, 19, 28, 32, 34, 35, 40, 45, 47, 51], which uses spoofed websites to steal users' passwords and financial information.

To protect users' online passwords and accounts, modern Web browsers have already implemented many security features and mechanisms. For example, the five most popular browsers (Internet Explorer, Firefox, Google Chrome, Safari, and Opera) all support extended validation (EV) certificates to help users verify the authenticity of websites, and they also provide phishing detection and warning mechanisms to help users stay away from phishing websites. However, simply relying on a single layer of protection is insufficient in many cases. Providing modern browsers with defense-in-depth mechanisms to better protect users' online passwords is definitely important and necessary.

In this paper, we investigate modern browsers' limitation in preventing users from submitting online passwords to inappropriate websites. We highlight that at least in two cases, accidental online password revealing may happen. The first case is that if a browser fails to detect a phishing webpage, users may submit their online passwords to the phishing website and become victims. This case can happen because automatic phishing detection techniques are still not able to detect all the phish-

*This work is based on Jeff Hinson's master thesis titled "Preventing the Revealing of Online Account Information to Non-Relevant Websites" [18]. Jeff built the initial prototype of LoginInspector in his thesis, and Chuan extended that prototype and completed this work. Jeff also helped revise this final version of paper after acceptance. Jeff prefers to be simply acknowledged rather than be an author of the paper. Chuan respects Jeff's choice and most sincerely appreciates him for his important contributions to this work.

ing attacks in a timely manner and meanwhile maintain a very low false positive rate [4, 13, 29, 39, 48, 49]. The second case is that when users forget the passwords for a certain website, a common practice for them is to try the passwords for other websites that they do remember. However, this practice may reveal a user's login information for a high-security website such as a banking website to an inappropriate low-security website such as a gaming website. This second case can happen and we confirmed this possibility by conducting a user study (Section 2). For ease of presentation, in this paper we refer to the first case as *undetected phishing attacks*, and the second case as *risky password tries*.

In both cases, users' online passwords can be revealed to inappropriate websites that should not receive them. Unfortunately, modern browsers do not provide sufficient protection to users in these two cases; meanwhile, no previous research has been done to seriously address this limitation of modern browsers. Indeed, in both cases, the key problem is that browsers do not have the knowledge to inform a user that such types of login actions could reveal the user's password to an inappropriate website. More specifically, when a browser has no knowledge (i.e., fails to detect) that a user is visiting a phishing webpage, the browser is not able to warn the user to leave this webpage; when a browser has no knowledge that a user is trying some mismatching passwords (i.e., passwords for other websites), the browser is not able to prevent the user from actually submitting the passwords to the current website.

To address the limitation of modern browsers in preventing users from submitting online passwords to inappropriate websites, we propose *LoginInspector*, a profiling-based warning mechanism. The key idea of LoginInspector is to continuously monitor a user's login actions and securely store hashed domain-specific successful login information to an in-browser database. Later on, whenever the user attempts to log into a website that does not have the corresponding successful login record, LoginInspector will warn and enable the user to make an informed decision on whether to really send this login information to the website. LoginInspector can also report users' insecure password practices to system administrators of an enterprise or campus environment so that targeted training and technical assistance can be provided to vulnerable users.

LoginInspector is more like a whitelist-based approach. Its in-browser database is like containing a whitelist of a user's successful login records for different websites, and the whitelist is dynamically built up and securely maintained. Using this whitelist, LoginInspector can enable a user to make informed decisions in both the aforementioned cases. In particular, in the case of undetected phishing attacks, as long as the user had not already become a victim of this specific phishing website, a corresponding successful login record for this phishing website does not exist in the database. Similarly, in the case of risky password tries, as long as a tried username and password pair does not belong to a valid account of the user on the current website, a corresponding successful login record for this tried login information on the current website does not exist in the database. Therefore, in both cases, LoginInspector can accurately warn a user and allow a user to cancel the actual submission of the password to the website; it can also send the related information to system administrators and assist them to further protect and train users who cannot properly interpret the warning messages.

LoginInspector is a pure browser-side mechanism. No server-side deployment is needed, and no modification to a user's passwords is needed. LoginInspector is designed as an auxiliary tool to help a user enhance the security of the online passwords. Therefore, it will not incur any functionality problem (albeit without providing protection) to a user's login activities even if, for example, the user needs to use a computer that does not have LoginInspector installed.

We have implemented a prototype of LoginInspector as a Firefox browser extension and evaluated it on 30 popular legitimate websites, 30 sample phishing websites, and one new phishing scam discovered by M86 Security Labs. Our evaluation and analysis indicate that LoginInspector is a secure and useful mechanism that can help users prevent the accidental revealing of passwords to inappropriate websites. It is a simple mechanism that can be easily integrated into modern Web browsers to complement their existing protection mechanisms. Security system administrators in our university commented that such a tool could be very helpful for them to strengthen campus IT security.

The remainder of the paper is organized as follows: Section 2 motivates this work by reviewing related research and presenting a user study of online login practices. Section 3 details the design of LoginInspector. Section 4 analyzes the security, usability, and deployment of LoginInspector. Section 5 describes the implementation and evaluation of LoginInspector. Finally, Section 6 makes a conclusion and discusses the future work.

2 Motivation

Password security heavily depends on creating strong passwords and protecting them from being stolen. Weak passwords suffer from brute-force and dictionary attacks [30]; therefore, many online services require users to create and use strong passwords that are sufficiently long, random, and hard to crack by attackers. How-

ever, strong passwords are difficult to remember for users [1, 9, 30, 42]. As demonstrated by a recent large-scale usability study, many users write down or otherwise store their passwords and especially those with a higher entropy [25]. Furthermore, no matter how strong they are, passwords are also vulnerable to harvesting or stealing attacks.

In some cases, users may accidentally reveal the password for one website to another inappropriate website, making their sensitive login information for the original website at risk. We now justify that at least in the cases of *undetected phishing attacks* and *risky password tries*, accidental online password revealing may happen. The case of undetected phishing attacks is related to the ever-increasing prevalence of password harvesting attacks. The case of risky password tries can be attributed to the reality that Web users have more online accounts than ever before, and they are forced to create and remember more and more usernames and passwords probably using insecure practices such as sharing passwords across different websites [12, 36].

2.1 Undetected Phishing Attacks

This first case happens when browsers fail to detect a phishing attack and give a warning about it. In such a case, a vulnerable user may submit the password for the real website to the inappropriate phishing website. Phishers can directly use the obtained login information to break into the user's online account.

To protect Web users against phishing attacks [6, 8, 10, 11, 19, 28, 32, 34, 35, 40, 45, 47, 51], modern browsers often employ automatic phishing detection and warning mechanisms [54, 55, 56]. In terms of automatic phishing detection, two general types of techniques are blacklist-based techniques [29, 39, 48] and heuristic-based techniques [4, 13, 49]. No matter what techniques are used, browsers are still not able to detect all the phishing attacks in a timely manner and meanwhile maintain a very low false positive rate [4, 13, 29, 39, 48, 49]. Therefore, modern browsers cannot protect vulnerable users if those phishing attacks cannot be detected in the first place.

LoginInspector is more like a whitelist-based mechanism: even if a browser fails to detect a phishing attack and give a warning about it, our mechanism can still provide one more layer of protection by explictly giving a warning to the user and informing the user that he or she did not log into this website before. A related work, AntiPhish [24], can also generate a warning message whenever a user attempts to give away sensitive information to a phishing website. However, LoginInspector uses password hashing techniques while AntiPhish uses password encryption techniques. We believe hashing is more appropriate than encryption in this application because essentially what we need is an authenticator rather than a reversible mapping. Moreover, the detection and warning capability of LoginInspector is more refined than that of AntiPhish. In AntiPhish, whenever a mismatch on password happens, a phishing attempt is assumed (Section 3.2 of [24]). Such a decision criterion is not accurate. For example, if a user types a wrong password on a legitimate website, AntiPhish will assume the current website is a phishing website, but LoginInspector will display one of the two warnings (Figure 5) to provide accurate information to the user.

2.2 Risky Password Tries

This second case happens when users forget the password for a certain website. In such a case, a common practice for users is to try the passwords for other websites they do remember. However, if a password for a high-security website is tried on a low-security website, then this password may be revealed to the low-security website. For example, if a user attempts to use his or her Gmail password on a low-security gaming website, the Gmail password is revealed to the gaming website. The user may also try the corresponding Gmail username, may use the same username for Gmail and for the gaming website, or may use the Gmail address as the contact information on that gaming website. Therefore, both the Gmail password and username could be known to the gaming website. If this low-security gaming website is hacked or even if its authentication log is unintentionally released, the revealed Gmail login information could be further acquired by attackers.

The authors of this paper are also guilty of this practice of risky password tries. To further validate that this risky practice is indeed a common practice, we conducted a user study. This user study was pre-approved by the IRB (Institutional Review Board) of our university. Thirty adults, 15 females and 15 males, participated in our user study. They were voluntary students and faculty members randomly recruited in our campus library, bookstore, and cafeteria; they came from 17 departments of our university. Twenty-two participants were between ages of 18 and 30, and eight participants were over 30 years old. We did not collect any other demographic or sensitive information from participants. We did not screen participants based on their Web browsing experiences. We did not provide any incentive to the participants. We interviewed the participants when we met them on campus and asked each of them to answer a five-point Likert-scale (Strongly disagree, Disagree, Neither agree nor disagree, Agree, Strongly Agree) [58] questionnaire that consists of seven close-ended questions as listed in Table 1. Note that we randomized the sequence of the seven questions for each individual participant.

Table 1: The seven close-ended questions.

Q1: The security of online passwords is a concern.
Q2: Sometimes I forget my login username for a website.
Q3: Sometimes I forget my login password for a website.
Q4: Sometimes I type the username for one website to log into another website.
Q5: Sometimes I type the password for one website to log into another website.
Q6: When I type the username for one website to log into another website, I hope the Web browser can give me a warning.
Q7: When I type the password for one website to log into another website, I hope the Web browser can give me a warning.

Table 2: Summary of the responses to the seven close-ended questions.

	Strongly disagree	**Disagree**	**Neither agree nor disagree**	**Agree**	**Strongly Agree**
Q1	0(0.0%)	3(10.0%)	1(3.3%)	17(56.7%)	9(30.0%)
Q2	1(3.3%)	5(16.7%)	2(6.7%)	20(66.7%)	2(6.7%)
Q3	1(3.3%)	3(10.0%)	0(0.0%)	24(80.0%)	2(6.7%)
Q4	1(3.3%)	1(3.3%)	3(10.0%)	20(66.7%)	5(16.7%)
Q5	0(0.0%)	3(10.0%)	3(10.0%)	18(60.0%)	6(20.0%)
Q6	0(0.0%)	3(10.0%)	11(36.7%)	12(40.0%)	4(13.3%)
Q7	0(0.0%)	2(6.7%)	9(30.0%)	13(43.3%)	6(20.0%)

A summary of the responses to the seven close-ended questions is presented in Table 2. Because the data collected are ordinal and do not necessarily have interval scales, we use the median and mode to summarize the data and use the percentages of responses to express the variability of the data. Overall, the median and mode responses are *Agree* for all the seven questions. Particularly in terms of passwords (Q3, Q5, and Q7), we can see that: (1) 86.7% of participants agree or strongly agree that sometimes they forget the password for a website; (2) 80.0% of participants agree or strongly agree that sometimes they try the password for one website on another website; and (3) 63.3% of participants agree or strongly agree that when they try the password for one website on another website, they hope the Web browser can give them a warning.

These results clearly indicate that users do sometimes forget passwords, and trying passwords for other websites is indeed a common practice. Answers to Q7 also indicate that most participants are aware of the risks of such type of password tries and they do expect browsers to give them a warning for their risky actions. Unfortunately, modern Web browsers do not provide protection to prevent risky password tries; meanwhile, no previous research has been done to seriously address this limitation of modern browsers.

2.3 Related Work on Password Management

To help Web users better manage their online accounts and enhance their password security, researchers have proposed a number of solutions such as password managers [50, 61], single sign-on systems [63], graphical passwords [7, 20], and password hashing systems [14, 33, 43]. These solutions have their own merits and advantages, but they also pose various reliability, security, and usability concerns. Browsers' built-in password managers as well as many third-party password managers [50, 61] must be able to recover the original passwords by decrypting the saved encrypted passwords. This requirement provides many opportunities for attackers to crack a password manager that is not well designed or implemented. LoginInspector does not need to recover the original passwords. Essentially, even if a user does not want to use any password manager, LoginInspector can still protect against undetected phishing attacks and risky password tries.

Web Wallet [41] is an anti-phishing solution, and essentially it is a password manager that can help users fill login forms using stored information. However, as pointed out by the authors, users have a strong tendency to use traditional Web forms for typing sensitive information instead of using the special browser sidebar user interface. Centralized single sign-on systems such as Microsoft Passport [63] may suffer from various attacks that could cause disastrous consequences [26]. Security limitations of graphical passwords are analyzed in [5, 38]. Security and usability limitations of password hashing systems such as Password Multiplier [14] and PwdHash [33] are analyzed in [3]. Both Password Multiplier and PwdHash require users to migrate their original passwords to hashed passwords, and this is a biggest usability limitation of those hashing-based password generation solutions as acknowledged in the Password Multiplier paper [14].

LoginInspector leverages the security advantages of password hashing techniques, but it does not inherit their usability disadvantages because it only uses hashing techniques to generate authenticators for determining warning types. Overall, even if some of existing solutions can to some extent help prevent accidental login information revealing, the majority of users who stick to the traditional way of using passwords (i.e., filling out a login form based on what they remember in the memory and submitting their original passwords to the remote website) still cannot be protected. What LoginInspector aims to accomplish is to prevent these users from accidentally revealing their sensitive login information.

Essentially, our key observation is that currently Web browsers do not have the knowledge to identify the afore-

mentioned accidental login information revealing cases. Therefore, they cannot help a user make informed decisions to avoid submitting sensitive login information to inappropriate websites. This observation motivated us to explore a simple profiling-based warning mechanism to provide an in-depth protection for Web users.

3 Design

In this section, we first give an overview on the design of LoginInspector. We then detail the architecture and components of LoginInspector.

3.1 Overview

The key idea of LoginInspector is to continuously monitor a user's login actions and securely store domain-specific successful login information to an in-browser database. Later on, whenever the user attempts to log into a website that does not have the corresponding successful login record, LoginInspector will warn and enable the user to make an informed decision on whether to really send this login information to the website.

We design LoginInspector as a browser extension that can be seamlessly integrated into modern Web browsers. As illustrated in Figure 1, the functioning of the LoginInspector browser extension consists of two main logical phases: the *profiling phase* and the *inspection and warning phase*. These two phases are centered around an in-browser *successful login profile database*.

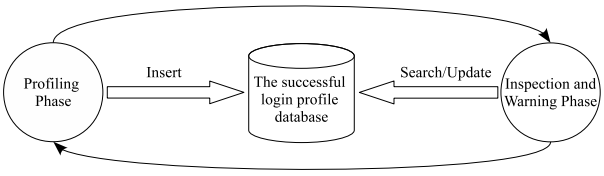


Figure 1: The functioning of the LoginInspector browser extension.

In the profiling phase, LoginInspector will build up the successful login profile for a user. Basically, whenever a user successfully logs into a website account for the first time, LoginInspector will insert a new record into the successful login profile database. Each record is uniquely determined by the domain name of the website and the username/password pair used in this successful login. For example, assume a user has two Twitter accounts A and B. The username/password pair is userA/pwdA for account A, and is userB/pwdB for account B. The first time the user successfully logs into twitter.com using account A, one new record will be created; the first time the user successfully logs into twitter.com using account B, another new record will be created. As will be elaborated in the next subsection, LoginInspector only stores hashed domain names and username/password pairs into the successful login profile database, thus minimizing security and privacy risks even if database records would be stolen by attackers.

In the inspection and warning phase, LoginInspector will leverage the information stored in the successful login profile database to enable a user to make informed login decisions. Basically, when a user types the username and password into a login form of a website, LoginInspector will first intercept this information. Second, it will inspect whether there is a corresponding successful login record for this website account in the database. Third, if there is a corresponding successful login record in the database, LoginInspector will submit the intercepted login information to the remote website; no warning will be given to the user and the user's login interaction is identical to that in the scenario of without using LoginInspector. LoginInspector can also update the corresponding database record with information such as the login timestamp.

However, if there is no corresponding successful login record in the database, LoginInspector will warn the user with accurate information regarding either the user did not log into this website before or the current login information does not match previous successful login records. It will ask the user to confirm whether this login information should really be submitted. If the user ignores the warning, LoginInspector will submit the intercepted login information to the website; if the user acknowledges the warning, LoginInspector will cancel the submission and allow the user to type in a new username and password.

LoginInspector is a pure browser-based solution – both the profiling phase and the inspection and warning phase happen inside of a user's browser. LoginInspector will ensure no password information will be transmitted to a remote website unless: (a) one corresponding successful login record exists in the database (indicating the current website is an appropriate website), or (b) the user ignores the warning and explictly confirms the transmission (indicating the user has made an informed decision based on the given warning).

3.2 Architecture and Components

Figure 2 illustrates the high-level architecture of LoginInspector, which consists of *the successful login profile database* and seven logical components: *login fields identification and protection*, *login profile inspection*, *warning generation*, *admin report*, *successful login detection*, *management*, and *import/export*.

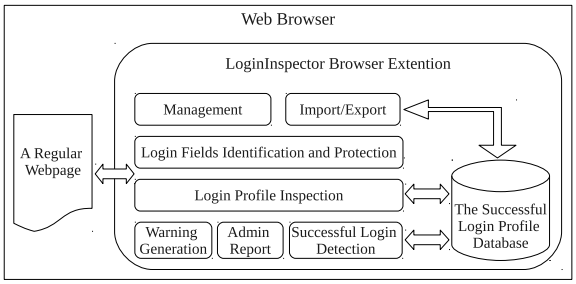


Figure 2: The architecture of the LoginInspector browser extension.

3.2.1 The successful login profile database

LoginInspector uses an in-browser database that can be implemented as an SQLite [62] database instance. SQLite has already been equipped in popular browsers such as Firefox and Google Chrome. Within the database, a *loginprofile* table is created to store all the successful login records. This table consists of six columns as shown in Table 3.

Table 3: The *loginprofile* table in the successful login profile database.

id	domainHmac	recordHmac	timesUsed	firstUsed	lastUsed

In the *loginprofile* table, each successful login record is uniquely identified by a recordHmac value. A unique *id* is generated by the database for each record. The values of domainHmac and recordHmac are calculated using Formula 1 and Formula 2, respectively:

$$domainHmac = HMAC(key,\ d) \quad (1)$$

$$recordHmac = HMAC(key,\ d \mid\mid u \mid\mid p) \quad (2)$$

where *key* is a secret key either randomly generated by the LoginInspector extension or directly specified by a user when LoginInspector is installed; *HMAC* is the Keyed-Hashing for Message Authentication [27] mechanism together with the SHA-256 [59] cryptographic hash function; d, u, and p represent domain name, username, and password, respectively; "||" is the string concatenation operator. The secret key is securely stored in the password manager of a browser and transparently used by LoginInspector. We allow a user to specify the secret key when LoginInspector is installed so that the records in the successful login profile database can be conveniently exported and imported by LoginInspector. A user can also use the master password mechanism of a browser to further protect against the wrong people extracting the secret key.

The domain name d is extracted from each login form's owner document, and it includes the full domain name prefixed with the protocol (e.g., https://www.amazon.com or http://en.wikipedia.org). In other words, we are interested in where exactly a login form comes from, instead of what the domain name of the top-level document loaded from a browser's URL address bar is. This design choice is reasonable because the owner document contains the most relevant information of a form. For example, on a mashup website (e.g., www.mashup.com), if a login form is submitted from a sub-frame document (e.g., specified by the *src* attribute of a *frame* element), the domain name of this sub-frame document will be extracted and used as the value for domain name d. Therefore, the saved successful login record can be matched no matter the owner document of the login form is included in mashup websites as a sub-frame document or is directly loaded as a top-level document. Similar to browsers' other features such as the password manager, LoginInspector uses more reliable and stable domain names instead of IP addresses and does not specifically consider pharming attacks which we believe should be addressed by some general solutions such as [21, 22].

The other three columns timesUsed, firstUsed, and lastUsed of the *loginprofile* table can keep track of the usage statistics of each record and provide a user with more detailed successful login information on a currently visited website. Users can configure whether they want to save these information to the database or not, by using the management component of LoginInspector.

3.2.2 Login fields identification and protection

When a new webpage is loaded in the browser, the login fields identification and protection component will identify the username and password login fields on the webpage; it will also intercept and protect a user's password keystrokes. It follows the strategy of first identifying the password field and then identifying the username field. To identify the password field, it uses two combined techniques: user-assisted identification and automatic identification. The user-assisted identification technique was proposed by Ross et al. in their PwdHash project [33]. This technique requires a user to either type the "@@" prefix (two consecutive "at" signs) or press the "F2" function key to explicitly indicate that an input field is a password field. The advantage of this technique is that it can effectively prevent malicious JavaScript on phishing webpages from stealing users' plaintext passwords. The disadvantage of this technique is that users need to remember to perform this additional action [3].

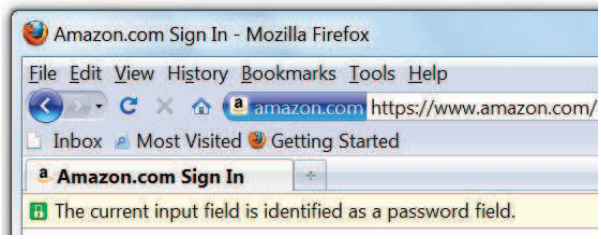


Figure 3: The password field indication message in a chrome notification box.

The automatic identification technique examines the special attribute *type="password"* in the DOM [53] tree to locate a password field. This technique can be evaded by phishers [33], but it is completely automatic.

The two identification techniques are combined in such a way that if a user indicates a password field, this field will be directly identified; otherwise, the automatic identification result will be used. Basically, once an input field is automatically identified as a password field, LoginInspector will mark that field as a protected field. Then, if and only if the input focus is on that field, LoginInspector will display a chrome type of notification box to a user below the browser's tab bar to make the user aware of that password field. Figure 3 illustrates a snapshot of the notification box. The notification box is of a chrome type, so that a LoginInspector user can customize it by putting special text or images into it. Therefore, a malicious webpage cannot easily spoof the notification box.

The user-assisted identification may be needed in two cases. One is when the automatic identification does not identify any password field. The other is when the automatically identified password field is inconsistent with the *should-be* password field as perceived by a user. In both cases, a user can set the input focus on the should-be password field and specify it as the password field using either the "@@" prefix or the "F2" function key [1]. LoginInspector will mark this user-specified password field as a protected field, and will similarly display the notification box if and only if the input focus is on this field. Note that usually this user-assisted identification may only need to be performed on phishing webpages, on which a password field could be deliberately made inaccurate [33].

If no password field is identified (either automatically or by the user), LoginInspector will do nothing anymore. Otherwise, the current webpage is regarded as a login webpage and LoginInspector will further identify the corresponding username field. Our experimental results (Section 5) indicate that the visible text input field immediately preceding the password field can be reliably identified as the username field. We found that Firefox also uses this username identification technique in its password manager. Even if the username field cannot be confidently identified on a login webpage, the functionality of LoginInspector will not be affected. In such a case, an empty string will be used as the username value for calculating the recordHmac. Therefore, the only effect is that the granularity of the successful login record becomes coarse if a user has multiple accounts sharing the same password on this website. Note that if a user has successfully logged into a website (Section 3.2.6), the username will be remembered by LoginInspector for this login session. Thus, if a user visits the change password webpage of this website, LoginInspector can still properly replace the old successful login record with the new calculated one.

After identifying the username and password fields of a login webpage, LoginInspector will monitor these two fields and will extract the login information typed by a user into these two fields. To prevent malicious JavaScript on a webpage such as a phishing webpage from recording a user's password keystrokes, LoginInspector will (1) intercept the password keystrokes as soon as a user begins to type, (2) prevent the real keypress events from propagating to the webpage, and (3) fire fake keypress events to generate the random fake password value. These three steps are basically the same as the ones developed by Ross et al. in their PwdHash project [33]. Later on, if the login form really needs to be submitted, LoginInspector will use the intercepted real password value to replace the generated fake password value, allowing the login process to proceed smoothly.

3.2.3 Login profile inspection

When a user submits a login form, the login profile inspection component will compare the intercepted user login information with the records stored in the successful login profile database. First, it will compute a *currentDomainHmac* and a *currentRecordHmac* using Formula 1 and Formula 2, respectively, based on the domain name extracted from the current login form's owner document and the username and password values intercepted from the current login form. Next, it will look up the database using the login profile inspection procedure illustrated in Figure 4. It passes the (currentDomainHmac, currentRecordHmac) pair to the procedure to get one of the three return results. The result *ExactMatch* means that there is an existing record with the recordHmac value equal to currentRecordHmac. The result *DomainMatch* means that there is no existing record with the recordHmac value equal to currentRecordHmac, but there is at least one record with the domainHmac

[1] To discard any identification result, a user can simply press the "F2" function key when the input focus in on that field.

```
Inspection (currentDomainHmac, currentRecordHmac)
1. if a record with recordHmac=currentRecordHmac exists
2.     return ExactMatch;
3. else
4.     if a record with domainHmac=currentDomainHmac exists
5.         return DomainMatch;
6.     else
7.         return NoMatch;
8.     endif
9. endif
```

Figure 4: The login profile inspection procedure.

value equal to currentDomainHmac. The result *NoMatch* means that there is no existing record with the domainHmac value equal to currentDomainHmac.

If the return result of the procedure is ExactMatch, the login profile inspection component will simply submit the login form using the intercepted real password, and no warning will be given to the user. The timesUsed and lastUsed information of the existing successful login record can also be updated. However, if the return result of the procedure is either DomainMatch or NoMatch, the login profile inspection component will further instruct the warning generation component to trigger a warning message.

3.2.4 Warning generation

This component will generate two types of warning messages based on the instruction from the login profile inspection component. One type of message, referred to as *Initial Visit*, corresponds to the return result NoMatch. In this case, the warning message will remind a user that the user may not have previously logged into this website, and will ask whether the user really wants to proceed with the login action. The other type of message, referred to as *Credential Mismatch*, corresponds to the return result DomainMatch. In this case, the warning message will remind a user that the user may not have previously used this username and password pair to successfully log into the current website, and will also ask whether the user really wants to proceed with the login action. In both cases, the warning message will be displayed in a modal chrome dialog box. This dialog box is of a modal type, so that before continuing to perform any other browsing interactions, a user must respond to the warning by either clicking the "OK" button to *ignore* it or clicking the "Cancel" button to acknowledge it. Similar to the notification box illustrated in Figure 3, this dialog box is also of a chrome type; therefore, it can be customized and cannot be easily spoofed by a malicious webpage. Figure 5(a) and Figure 5(b) illustrate the snapshots of the dialog boxes for the two types of warning messages, respectively.

The *Initial Visit* warning message can be triggered when a user tries to log into a new legitimate website or a phishing website. The *Credential Mismatch* warning message can be triggered when a user tries to log into a website using the username and password information of a new account for the first time or using the username and password information for another website. These warning messages intend to help users make informed decisions, and we expect users can properly interpret these messages and make the correct decisions. If a user ignores a warning message, the login action continues and the username and password information will be sent to the website; if a user acknowledges a warning message, login action stops and nothing will be sent to the website.

A user can follow three basic principles to decide whether to ignore or acknowledge a warning message. First, at the beginning when LoginInspector is installed and used, the *Initial Visit* warning message will be given on each website because no existing record exists in the database. A user normally should ignore the warning. Second, after a user has successfully logged into most of his or her online accounts (e.g., after a couple of weeks), the occurrence of the *Initial Visit* warning message should be rare. Therefore, a user should be very cautious about this type of warning message and should carefully inspect whether the current website is a phishing website. The user should acknowledge the warning if the website is suspicious, and should otherwise ignore the warning. Third, the *Credential Mismatch* warning message should be rare all the time. If a user has two accounts on the same website, then ideally this type of message should only occur once when the user logs into the website using the second account for the first time. Therefore, a user should be very cautious about this type of message and should think about whether the tried login information is for another website (thus the warning should be acknowledged) or for another account on the same website (thus the warning should be ignored). These principles can be provided in the manual of LoginInspector to help users properly interpret the warning messages.

3.2.5 Admin report

This component will generate and send reports to system administrators if it is enabled by either a system administrator or a user. In an enterprise or campus environment, system administrators can configure LoginInspector so that even if some users cannot properly interpret the above warning messages, the related information can be reported to an internal website of administrators through a POST type HTTP request.

The report does not contain passwords or their hash values; it only needs to contain the LoginInspector us-

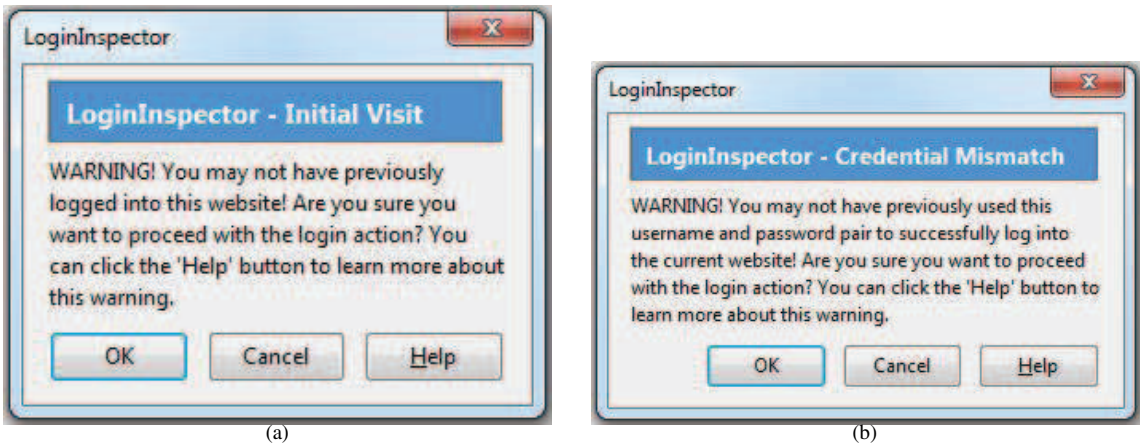


Figure 5: The modal chrome dialog boxes for (a) the *Initial Visit* warning message, (b) the *Credential Mismatch* warning message.

age information. For example, it can contain statistical information on a user's responses to the two types of warning messages in a session such as {"userid": "123456", "ignored *Initial Visit* warning": "10 times", "ignored *Credential Mismatch* warning": "6 times", "sessionStartTime": "1345846451434", "sessionEndTime": "1345846648635", ...}. If necessary, it can also contain the URLs of the websites on which warning messages were ignored. System administrators can then leverage the reported information to identify users' insecure password practices such as risky password tries or improper interpretation of warning messages, and they can further take actions to protect and train those vulnerable users. Administrators could also aggregate the reports received from different users to predict a phishing wave. For example, if suddenly a large number of *Initial Visit* messages are ignored by a bunch of users on some associated IP addresses or domain names, the likelihood of a new phishing scam is high. Security system administrators in our university commented that LoginInspector and its report capability could be very helpful for them to strengthen campus IT security.

3.2.6 Successful login detection

This component will detect whether a user's login attempt is successful in the case when the user ignores a warning (Figure 5) and LoginInspector submits the login information to the website. The case that the login profile inspection procedure returns ExactMatch is directly considered as a successful login and is not handled by this component. Originally, we intended to make this detection process completely automatic. We found that one reliable heuristic for automatically detecting a successful login is to examine whether the login response webpage loaded on the browser also contains a visible password field. If so, this login action is considered unsuccessful because normally a failed login webpage asks a user to type in the username and password information again. Otherwise, the login action is considered successful. This technique works accurately on over 95% of websites that we tested (Section 5), but in rare cases the detection result was wrong. For example, the response webpage for a failed login may simply contain a link or button, which can take a user to the re-login webpage. In such a case, a failed login will be incorrectly identified as a successful login.

To overcome the limitation of the automatic detection, we introduced a user-assisted successful login detection method. Basically, once a login response webpage is fully loaded on the browser, LoginInspector will explictly ask a user to confirm whether this login attempt is successful. This confirmation message is also displayed in a modal chrome dialog box, and a user's response on the "Yes" or "No" button will be directly used as the detection result. This method is intuitive because a well-designed login response webpage often clearly manifests the status of the login action, which can be easily and accurately identified by a user. Moreover, this method will not impose too much burden on a user because the confirmation is needed only when a warning (Figure 5) is given and meanwhile the warning is ignored by the user. Once a successful login is confirmed by a user, a new record (Table 3) will be added to the *loginprofile* database table.

3.2.7 Management and import/export

Finally, the management component will enable a user to perform tasks such as customizing warning messages, removing successful login records, and configuring whether to track the timesUsed, firstUsed, and lastUsed information. The import/export component will enable a user to export the successful login records to a

file and later to import those records from this file to a new browser on another computer. This import/export functionality is similar to that of the bookmark feature in Web browsers. We mentioned in Section 3.2.1 that the secret key used in Formula 1 and Formula 2 can be specified by a user. The advantage is that when a user imports the successful login records to a new browser on another computer, the user can directly type the specified key into LoginInspector through its management user interface. Otherwise, the randomly generated secret key also needs to be exported from the original browser and then supplied to the new browser, and such a functionality should only be accessed by authorized users. A user's profile should be synchronized between computers so that previously processed warning messages will not appear again. Currently, users can use this import/export functionality to achieve profile synchronization. In the future, we plan to enable users to take advantage of the data synchronization mechanism of browsers such as Firefox and Google Chrome to easily perform profile synchronization.

4 Analysis

We now analyze the security, usability, and deployment of LoginInspector.

4.1 Security

On the one hand, LoginInspector itself is designed to be secure. A user's login information for a website is hashed using the HMAC [27] message authentication mechanism together with the cryptographically strong SHA-256 [59] hash function, and only hashed values are stored in the successful login profile database. Therefore, even if attackers can manage to steal the complete database file or some successful login records, it is computationally infeasible for them to figure out the original plaintext username/password and domain name values that map to those HMAC values. Meanwhile, because this mechanism does not send the intercepted login information or hashed successful login records to any third-party server, it will not incur any new security or privacy problems.

On the other hand, LoginInspector can provide security benefits to a user by giving two types of warning messages (Figure 5) based on the previous successful login history of the user. This capability of LoginInspector is unique and is exactly what is lacking in existing Web browsers. Moreover, thanks to domain-based hashing, this capability is robust regardless of whether a user will reuse usernames and passwords across different websites or not. We now further analyze this capability for three categories of users based on the two accidental login information revealing cases defined in Section 2.

The first category of users are security conscious users who will never visit phishing websites and never perform risky password tries. These users do not need to use LoginInspector. If they do use, they will see very few warning messages once their successful login profiles become stable, and they can simply ignore those messages. The second category of users may accidentally visit phishing websites and become victims. Whenever a user in this category tries to log into a phishing website, regardless of the browser's ability to detect the attack, LoginInspector will display the *Initial Visit* warning message and explicitly inform the user that he or she may not have previously logged into the website. The third category of users may sometimes perform risky password tries. Whenever a user in this category performs a risky password try, LoginInspector will display the *Credential Mismatch* warning message and explicitly inform the user that he or she may not have previously used this username and password pair to successfully log into the current website. Note that there could be an overlap between the second category and the third category of users.

Modern Web browsers display "active" warnings to boost the effectiveness of their phishing and SSL error protection mechanisms [8, 37]. Because LoginInspector displays warning messages in a modal chrome dialog box, these warnings are also "active" and a user has to take an action. Therefore, it is reasonable to expect that by following the principles suggested in Section 3.2.4, users in the second and third categories can, to some extent, properly interpret the warning messages and protect themselves from accidentally revealing login information. Training users to read, understand, and (most importantly) pay serious attention to the warning messages is absolutely critical to the effectiveness of security warning mechanisms. Herley discussed that "users' rejection of the security advice they receive is entirely rational from an economic perspective" [15]. Following the recommendations provided in [15], we suggest that such a training should target at-risk population, that is, those who are vulnerable to phishing attacks and/or who have the practice of risky password tries, so that a better cost-benefit ratio can be achieved.

4.2 Usability

LoginInspector has two major usability advantages. One is that a user does not need to change the original passwords for any website. Some existing password management or phishing protection solutions such as Password Multiplier [14], PwdHash [33], and Passpet [43] all require users to visit the "change password" page of each individual website to migrate the original unmanaged password to a hashed password. However, such a

requirement imposes a big burden on a user. As pointed out in the Password Multiplier paper [14], "all the hash-based schemes have difficulty with the transition from unmanaged passwords."

The other usability advantage is that the login action of a LoginInspector user will not be affected at all, even if the user needs to use a browser on a computer that does not have LoginInspector installed. This is because LoginInspector is designed as an auxiliary tool to help a user enhance the security of the online passwords. The regular login functionality will be enhanced with one more layer of security protection, but it will not be degraded or disrupted when LoginInspector becomes unavailable. This usability advantage is lacking in those password hashing solutions [14, 33, 43]; it is also lacking in browsers' built-in or third-party password managers [50, 61].

LoginInspector also has two main usability disadvantages. One is that the two types of warning messages, especially the *Initial Visit*, will be frequently displayed during the profiling phase, and a user should ignore the warnings to build up the successful login profile. To better ensure the quality of this profiling phase, we suggest a user to perform it in a batch manner once LoginInspector is installed. For example, if a user has 30 online accounts on 20 websites, the user can log into those 30 accounts in about one hour to establish his or her successful login profile. During this process, if the user carefully submits the valid login information for all the 30 accounts, then 20 *Initial Visit* warning messages will be displayed and 10 *Credential Mismatch* warning messages will be displayed. The user simply needs to ignore all these 30 warnings to finish the profiling phase. Later on, the user will not see any warning message if the login information of those 30 accounts are used to log into the corresponding websites. The user only needs to be cautious about the two types of warning messages if they appear again. In an enterprise or campus environment, system administrators can also help regular users build up the profile, thus reducing the number of required actions from regular users and making the profiling stage of LoginInspector easier, faster, and less error-prone.

Note that the impact of webpage redirection to LoginInspector is very limited because similar to browsers' built-in password managers, LoginInspector only extracts URLs from the final login webpage instead of from an intermediate redirection webpage. It is very rare for the same website to use different URLs to host the login webpage for the same type of accounts[2]; if such a case happens, LoginInspector may display the *Initial Visit* warning message, and browsers' built-in password managers may ask a user to save another record to the database.

The other main usability disadvantage is that the established successful login profile is associated with a browser on one computer, and is not directly accessible on other computers. To address this issue, we suggest that if a user has multiple computers, the user can simply export the established successful login profile and import it to other computers. Therefore, the aforementioned profiling phase still only needs to be performed once in a batch manner, minimizing the burden on a user. However, if a user simply wants to temporarily use another computer such as a public computer [46], we do not suggest the user import his or her successful login profile to that computer. In other words, LoginInspector mainly focuses on protecting a user on his or her own computers.

[2] A bank website may use different login URLs, but normally they correspond to different types of accounts such as credit card accounts and saving accounts.

4.3 Deployment

LoginInspector can be incrementally deployed and the deployment is very simple. One reason is this mechanism is a pure browser-based solution and it can be seamlessly integrated into modern Web browsers as an extension. No server-side modification is needed. The other reason is that this mechanism simply provides one additional layer of protection to Web users. It only uses the existing login information of a user and it does not require any modification to the existing online user authentication mechanisms.

5 Evaluation

LoginInspector is designed to be implementable on different Web browsers. In-browser databases such as SQLite [62] are equipped in popular browsers such as Firefox and Google Chrome, making the implementation of the successful login profile database feasible. Meanwhile, the end-user extensibility of modern browsers such as Firefox, Internet Explorer, and Google Chrome also makes the implementation of other LoginInspector components feasible. We have implemented a prototype of LoginInspector as a Firefox browser extension. It works well as tested on Firefox versions from 3 to 9, which was the latest version at the time of this writing. We do not anticipate problems with newer versions. The extension is purely written in JavaScript with approximately 1600 lines of code. We believe LoginInspector can also be easily implemented as the extension for other popular browsers.

We have evaluated this LoginInspector extension on 30 popular legitimate websites, 30 sample phishing websites, and one new phishing scam discovered by M86 Security Labs [60]. We selected the 30 popular legitimate websites (as listed in Table 4) from two sources. One

source is the top 50 websites listed by Alexa.com; however, we removed non-English websites, gray content (e.g., adult) websites, and the websites that did not allow us to create an account. The other source is some of our frequently used websites. Websites such as paypal.com and wellsfargo.com set the *autocomplete="off"* property on their password fields or login forms; therefore, browsers' autocomplete feature [44] will not save users' form filling history to help speed up their future form filling process [44]. LoginInspector only stores hashed values to dramatically reduce the risk of having users' passwords cracked by attackers; thus its current version does not respect the *autocomplete* property and can work well on websites with the "autocomplete=false" property. Note that LoginInspector may not work well on websites that use one-time passwords because it could always raise the *Credential Mismatch* warning after the initial visit. What a user can do is to configure LoginInspector to ignore those sites based on their domain names.

We selected the 30 phishing websites from phishtank.com, which is a community based anti-phishing service widely used in research [29, 48, 49]. These phishing websites were randomly sampled with the criteria that they were online during our experiments, they were containing login webpages, and they were hosted on different domains or IP addresses. In our experiments, we mainly focused on evaluating the correctness and performance of this LoginInspector extension.

Table 4: The 30 popular legitimate websites.

mail.google.com	facebook.com	mail.yahoo.com
wikipedia.com	twitter.com	amazon.com
linkedin.com	wordpress.com	ebay.com
fc2.com	craigslist.org	imdb.org
aol.com	digg.com	careerbuilder.com
buy.com	aaa.com	newegg.com
tumblr.com	alibaba.com	4shared.com
cnn.com	nytimes.com	foxnews.com
weather.com	groupon.com	photobucket.com
myspace.com	webmail.uccs.edu	portal.prod.cu.edu

5.1 Correctness

We verified that this LoginInspector extension integrates seamlessly with the Firefox Web browser and works correctly on all of the 30 popular legitimate websites, the 30 sample phishing websites, and the new phishing scam discovered by M86 Security Labs [60].

5.1.1 Results on legitimate websites

On the 30 popular legitimate websites, LoginInspector can correctly and automatically identify the password field and the username field on each of the login webpages. It can correctly intercept password keystrokes and replace the intercepted password with a generated fake password; it can properly replace back the intercepted password when the login information really needs to be sent to the website. We observed the heuristic for automatic successful login detection that we originally planed to use (Section 3.2.6) works correctly on 29 websites except for aaa.com, which uses an extra response webpage that contains a link "Return to sign in page" for a failed login attempt. As discussed in Section 3.2.6, we switched to a user-assisted successful login detection method to overcome the limitation of the automatic detection. This method works correctly based on the user confirmation action.

Through logging all the operations and manually checking the content of the *loginprofile* table, we verified that all the database operations – including insert, update, and select – were correctly performed for the 30 websites. Meanwhile, the login profile inspection procedure illustrated in Figure 4 works correctly based on the existing records in the *loginprofile* database table, and the decisions on whether and what type of warning messages (Section 3.2.4) should be displayed were also precisely made. Note that in these correctness evaluations, whenever doable, we created at least two accounts on each website to test all the possible usage scenarios.

5.1.2 Results on phishing websites

On the 30 sample phishing websites, we observed that the password field and the username field can be correctly and automatically identified on 29 login webpages. Only on one login webpage the password field was not automatically identified by LoginInspector. We checked that the password field on that login webpage has the property *type="text"*, and every password character was displayed to a user in the input field. These results indicate that, overall, the sophisticated phishing attacks presented by Ross et al. [33] are not yet used in many phishing attacks. However, as analyzed in Section 3.2.2, if the automatic identification does not work well on phishing websites like in the above *type="text"* case, a user should specify the password field using either the "@@" prefix or the "F2" function key.

We also verified that the *Initial Visit* warning message was correctly displayed by LoginInspector on all the phishing login webpages. In our experiments, we acknowledged all those warnings because we do not want to save any phishing website record to the successful login profile database. In addition, among these 30 phishing websites, we observed that Firefox failed to detect seven of them and Google Chrome failed to detect eight

of them[3]. Therefore, on those phishing websites, no warning was displayed by these two popular browsers. This observation further justifies one of the motivations of our work, that is, *undetected phishing attacks* are commonplace.

5.1.3 Results on one new phishing scam

A new phishing scam was discovered by M86 Security Labs in 2011 [60]. Basically in this scam, phishers attach an HTML file to the spam email, luring a user to open the attached HTML file and submit a form to perform some urgent tasks. Once a user submits the form, the stolen sensitive information will be transmitted through a POST type HTTP request to a hacked legitimate website. This new phishing scam is very stealthy because: (1) a browser simply loads the phishing webpage as a local file such as *file:///C:/Users/.../home.html*; (2) the form submission target is a legitimate, albeit hacked, website. Therefore, neither a browser nor a user can easily identify such a phishing attack. As reported by M86 Security Labs [60], popular browsers such as Firefox and Google Chrome did not detect any such malicious activity; meanwhile, there is an increase in these types of phishing spam campaigns over the last few months.

We decided to test whether LoginInspector can defend against this new phishing scam. We created emails to attach various login webpages, and then opened those attachments using latest versions of Firefox, Google Chrome, and Internet Explorer browsers. Obviously, no phishing warning was given by those Web browsers, and it seems those browsers currently do not use heuristic-based phishing detection techniques to inspect locally opened (i.e., *file:///...*) HTML webpages. As expected, LoginInspector also works correctly on the locally opened HTML webpages. It correctly displayed the *Initial Visit* warning message, thus enabling a user to make the informed decision to acknowledge warnings and cancel the submissions when those undetected phishing attacks occur.

5.2 Performance

We also measured the performance overhead of LoginInspector on the 30 popular legitimate websites. Firefox and the LoginInspector browser extension were installed on a laptop with a 2.67GHz CPU. Other JavaScript operations and HMAC calculations (Formula 1 and Formula 2) cause negligible overhead. For example, all the tested HMAC calculations were completed within 3 milliseconds. The overhead is mainly on the SQLite [62] database operations invoked by the JavaScript code. On each of the 30 legitimate websites, we measured the overhead of the database operations five times. We observed that all the select operations were completed within one millisecond. The average performance overhead for the insert operations is 140.6 milliseconds with a standard deviation of 47.2. The average performance overhead for the update operations is 70.2 milliseconds with a standard deviation of 13.1. We can see that, overall, the database operation overhead is still very low, and is only incurred when a login form is submitted. In addition, the insert operation overhead is only incurred when a new successful login record needs to be added to the database.

[3]Both Firefox and Chrome use the same blacklist provided by Google [52]; this slight difference in false negative rate could be caused by the blacklist download time difference between the two browsers.

6 Conclusion and Future Work

In this paper, we determined that modern Web browsers do not provide sufficient protection to prevent users from submitting their online passwords to inappropriate websites. We highlighted that in the cases of undetected phishing attacks and risky password tries, users may accidentally reveal their passwords for high-security websites to inappropriate low-security websites or even phishing websites. We proposed and presented *LoginInspector*, a profiling-based warning mechanism to address this limitation of modern browsers. LoginInspector establishes a successful login profile for a user and leverages this profile to enable a user to make informed login decisions and also enable system administrators to provide further protection or targeted training to vulnerable users. We analyzed the security, usability, and deployment of LoginInspector. We also evaluated the correctness and performance of the Firefox LoginInspector browser extension on legitimate and phishing websites. Our evaluation and analysis indicate that LoginInspector is a secure and useful mechanism, and it can complement Web browsers' existing mechanisms to provide an in-depth protection to a user's online login process.

In our future work, we plan to design a visual way (e.g., by using icons) to clearly differentiate the two types of warning messages (Figure 5). We want to evaluate whether a visually distinguishing factor could help users better understand what is happening without having to read those warning messages. We also plan to evaluate the usability of this standalone browser extension and then integrate it into the password managers of modern browsers. This integration can leverage the existing components of browsers' password managers. However, this integration will still allow users to independently enable either the LoginInspector or the password manager in case some users do not want to use both features.

7 Acknowledgments

We thank anonymous reviewers for their insightful comments and valuable suggestions. We thank our shepherd Mario Obejas for his great help in improving the final version of this paper. We also thank all the voluntary students and faculty members who participated in our user study. Jeff Hinson made important contributions to this work as highlighted on the first page of the paper. This work was partially supported by a UCCS 2011-2012 CRCW research grant.

References

[1] ADAMS, A., AND SASSE, M. A. Users are not the enemy. *Commun. ACM 42*, 12 (1999), 40–46.

[2] BONNEAU, J., HERLEY, C., VAN OORSCHOT, P. C., AND STAJANO, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of the IEEE Symposium on Security and Privacy* (2012), pp. 553–567.

[3] CHIASSON, S., VAN OORSCHOT, P. C., AND BIDDLE, R. A usability study and critique of two password managers. In *Proceedings of the USENIX Security Symposium* (2006), pp. 1–16.

[4] CHOU, N., LEDESMA, R., TERAGUCHI, Y., AND MITCHELL, J. C. Client-side defense against web-based identity theft. In *Proceedings of the NDSS* (2004).

[5] DAVIS, D., MONROSE, F., AND REITER, M. K. On user choice in graphical password schemes. In *Proceedings of the USENIX Security Symposium* (2004), pp. 151–164.

[6] DHAMIJA, R., AND J.D.TYGAR. The battle against phishing: Dynamic security skins. In *Proceedings of the SOUPS* (2005), pp. 77–88.

[7] DHAMIJA, R., AND PERRIG, A. Déjà vu: A user study using images for authentication. In *Proceedings of the USENIX Security Symposium* (2000), pp. 45–58.

[8] EGELMAN, S., CRANOR, L. F., AND HONG, J. You've been warned: An empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the CHI* (2008), pp. 1065–1074.

[9] FELDMEIER, D. C., AND KARN, P. R. Unix password security – ten years later. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)* (1989), pp. 44–63.

[10] FELTEN, E. W., BALFANZ, D., DEAN, D., AND WALLACH, D. S. Web Spoofing: An Internet Con Game. In *Proceedings of the 20th National Information Systems Security Conference* (1997).

[11] FLORÊNCIO, D., AND HERLEY, C. Password rescue: A new approach to phishing prevention. In *Proceedings of the HotSEC* (2006).

[12] FLORÊNCIO, D., AND HERLEY, C. A large-scale study of web password habits. In *Proceedings of the WWW* (2007), pp. 657–666.

[13] GARERA, S., PROVOS, N., CHEW, M., AND RUBIN, A. D. A framework for detection and measurement of phishing attacks. In *Proceedings of the Workshop on Rapid Malcode (WORM)* (2007).

[14] HALDERMAN, J. A., WATERS, B., AND FELTEN, E. W. A convenient method for securely managing passwords. In *Proceedings of the WWW* (2005), pp. 471–479.

[15] HERLEY, C. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the New security Paradigms Workshop (NSPW)* (2009), pp. 133–144.

[16] HERLEY, C., AND VAN OORSCHOT, P. C. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy 10*, 1 (2012), 28–36.

[17] HERLEY, C., VAN OORSCHOT, P. C., AND PATRICK, A. S. Passwords: If we're so smart, why are we still using them? In *Proceedings of the Financial Cryptography* (2009), pp. 230–237.

[18] HINSON, J. Preventing the Revealing of Online Account Information to Non-Relevant Websites. Master's Thesis (advised by Chuan Yue) at UCCS, `http://library.uccs.edu/search/o693952729`.

[19] JAKOBSSON, M., AND MYERS, S. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley-Interscience, ISBN 0-471-78245-9, 2006.

[20] JERMYN, I., MAYER, A., MONROSE, F., REITER, M. K., AND RUBIN, A. D. The design and analysis of graphical passwords. In *Proceeding of the USENIX Security Symposium* (1999), pp. 1–14.

[21] JUELS, A., JAKOBSSON, M., AND JAGATIC, T. N. Cache cookies for browser authentication (extended abstract). In *Proceedings of the IEEE Symposium on Security and Privacy* (2006), pp. 301–305.

[22] KARLOF, C., SHANKAR, U., TYGAR, J. D., AND WAGNER, D. Dynamic pharming attacks and locked same-origin policies for web browsers. In *Proceedings of the CCS* (2007), pp. 58–71.

[23] KELLEY, P. G., KOMANDURI, S., MAZUREK, M. L., SHAY, R., VIDAS, T., BAUER, L., CHRISTIN, N., CRANOR, L. F., AND LOPEZ, J. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proceedings of the IEEE Symposium on Security and Privacy* (2012), pp. 523–537.

[24] KIRDA, E., AND KRUEGEL, C. Protecting users against phishing attacks with antiphish. In *Proceedings of the Annual International Computer Software and Applications Conference (COMPSAC)* (2005), pp. 517–524.

[25] KOMANDURI, S., SHAY, R., KELLEY, P. G., MAZUREK, M. L., BAUER, L., CHRISTIN, N., CRANOR, L. F., AND EGELMAN, S. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the CHI* (2011), pp. 2595–2604.

[26] KORMANN, D. P., AND RUBIN, A. D. Risks of the passport single signon protocol. *Comput. Networks 33*, 1-6 (2000), 51–58.

[27] KRAWCZYK, H., BELLARE, M., AND CANETTI, R. RFC 2104, HMAC: Keyed-Hashing for Message Authentication, 1997. `http://www.ietf.org/rfc/rfc2104.txt`.

[28] KUMARAGURU, P., RHEE, Y., ACQUISTI, A., CRANOR, L. F., HONG, J., AND NUNG, E. Protecting people from phishing: The design and evaluation of an embedded training email system. In *Proceedings of the CHI* (2007), pp. 905–914.

[29] LUDL, C., MCALLISTER, S., KIRDA, E., AND KRUEGEL, C. On the effectiveness of techniques to detect phishing sites. In *Proceedings of the DIMVA* (2007).

[30] MORRIS, R., AND THOMPSON, K. Password security: a case history. *Commun. ACM 22*, 11 (1979), 594–597.

[31] PROVOS, N., RAJAB, M. A., AND MAVROMMATIS, P. Cybercrime 2.0: when the cloud turns dark. *Commun. ACM 52*, 4 (2009), 42–47.

[32] RACHNA DHAMIJA, J.D.TYGAR, AND MARTI HEARST. Why phishing works. In *Proceedings of the CHI* (2006), pp. 581–590.

[33] Ross, B., Jackson, C., Miyake, N., Boneh, D., and Mitchell, J. C. Stronger password authentication using browser extensions. In *Proceedings of the USENIX Security Symposium* (2005), pp. 17–32.

[34] Schechter, S. E., Dhamija, R., Ozment, A., and Fischer, I. The emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *Proceedings of the IEEE Symposium on Security and Privacy* (2007), pp. 51–65.

[35] Sheng, S., Magnien, B., Kumaraguru, P., Acquisti, A., Cranor, L. F., Hong, J., and Nunge, E. Anti-Phishing Phil: the design and evaluation of a game that teaches people not to fall for phish. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2007), pp. 88–99.

[36] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R. A., Kruegel, C., and Vigna, G. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the CCS* (2009), pp. 635–647.

[37] Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., and Cranor, L. F. Crying wolf: an empirical study of ssl warning effectiveness. In *Proceedings of the 18th conference on USENIX security symposium* (2009), pp. 399–416.

[38] Thorpe, J., and van Oorschot, P. Human-seeded attacks and exploiting hot-spots in graphical passwords. In *Proceedings of the USENIX Security Symposium* (2007), pp. 103–118.

[39] Whittaker, C., Ryner, B., and Nazif, M. Large-scale automatic classification of phishing pages. In *Proceedings of the NDSS* (2010).

[40] Wu, M., Miller, R. C., and Garfinkel, S. L. Do security toolbars actually prevent phishing attacks? In *Proceedings of the CHI* (2006), pp. 601–610.

[41] Wu, M., Miller, R. C., and Little, G. Web wallet: preventing phishing attacks by revealing user intentions. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2006), pp. 102–113.

[42] Yan, J., Blackwell, A., Anderson, R., and Grant, A. Password memorability and security: Empirical results. *IEEE Security and Privacy 2*, 5 (2004), 25–31.

[43] Yee, K.-P., and Sitaker, K. Passpet: convenient password management and phishing protection. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2006), pp. 32–43.

[44] Yue, C. Mitigating cross-site form history spamming attacks with domain-based ranking. In *Proceedings of the DIMVA* (2011), pp. 104–123.

[45] Yue, C., and Wang, H. Anti-Phishing in Offense and Defense. In *Proceedings of the ACSAC* (2008), pp. 345–354.

[46] Yue, C., and Wang, H. SessionMagnifier: A Simple Approach to Secure and Convenient Kiosk Browsing. In *Proceedings of the Ubicomp* (2009), pp. 125–134.

[47] Yue, C., and Wang, H. BogusBiter: A Transparent Protection Against Phishing Attacks. *ACM Transactions on Internet Technology (TOIT) 10*, 2 (2010), 1–31.

[48] Zhang, Y., Egelman, S., Cranor, L. F., and Hong, J. Phinding phish: Evaluating anti-phishing tools. In *Proceedings of the NDSS* (2007).

[49] Zhang, Y., Hong, J., and Cranor, L. CANTINA: A content-based approach to detecting phishing web sites. In *Proceedings of the WWW* (2007), pp. 639–648.

[50] 1Password. `http://agilebits.com/products/1Password`.

[51] Anti-Phishing Working Group. `http://www.antiphishing.org`.

[52] Client specification for the Google Safe Browsing v2.2 protocol. `http://code.google.com/p/google-safe-browsing/wiki/Protocolv2Spec`.

[53] Document Object Model (DOM). `http://www.w3.org/DOM/`.

[54] Firefox Phishing and Malware Protection. `http://www.mozilla.com/en-US/firefox/phishing-protection/`.

[55] Google Chrome and Browser Security. `http://www.google.com/chrome/intl/en/more/security.html`.

[56] Internet Explorer 8 Readiness Toolkit. `http://www.microsoft.com/windows/internet-explorer/readiness/new-features.aspx`.

[57] Internet Security Threat Report, Security research and analysis — Symantec. `http://www.symantec.com/business/theme.jsp?themeid=threatreport`.

[58] Likert scale. `http://en.wikipedia.org/wiki/Likert_scale`.

[59] NIST: Secure Hashing. `http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html`.

[60] Phishing Scam in an HTML Attachment. `http://labs.m86security.com/2011/03/phishing-scam-in-an-html-attachment/`.

[61] RoboForm Password Manager. `http://www.roboform.com/`.

[62] SQLite Home Page. `http://www.sqlite.org`.

[63] Windows Live ID. `http://msdn.microsoft.com/en-us/library/bb288408.aspx`.

XUTools: Unix Commands for Processing Next-Generation Structured Text

Gabriel A. Weaver
Dartmouth College

Sean W. Smith
Dartmouth College

Keywords: Text processing, Configuration Management, Change Management, Automation, Tools, Unix

Abstract

Traditional Unix tools operate on sequences of characters, bytes, fields, lines, and files. However, modern practitioners often want to manipulate files in terms of a variety of language-specific constructs—C functions, Cisco IOS interface blocks, and XML elements, to name a few. These language-specific structures quite often lie beyond the regular languages upon which Unix text-processing tools can practically compute. In this paper, we propose eXtended Unix text-processing tools (xutools) and present implementations that enable practitioners to extract (`xugrep`), count (`xuwc`), and compare (`xudiff`) texts in terms of language-specific structures. We motivate, design, and evaluate our tools around real-world use cases from network and system administrators, security consultants, and software engineers from a variety of domains including the power grid, healthcare, and education.

1 Introduction

During our fieldwork, we observed the need to generalize Unix text processing tools so that practitioners can process the right type of string for the job at hand. We thus need to extend traditional Unix tools because many modern, structured-text formats break assumptions of traditional Unix tools.

Traditional Unix tools operate on sequences of characters, bytes, fields, lines, and files. When lines and files correspond to language-specific constructs, traditional Unix tools work well. For example, lines of Apache log files correspond to HTTP requests.

However, there are many other language-specific constructs besides the line. Many file formats found in markup, configuration, and programming languages encode meaningful structures via nested blocks. Natural-language documents may be divided up into chapters, paragraphs, and sentences. Sentences themselves may be further parsed. Configuration files and programming languages, similarly, may be divided up into blocks of code. For example, Cisco IOS interface blocks or C function blocks are information units that network administrators and developers reference daily.

The prevalence of multi-line, nested-block-structured formats has left a capability gap for traditional tools. Today, if practitioners want to extract interfaces from a Cisco IOS router configuration file, they must craft an invocation for `sed(1)`

```
sed -n '/^interface
  ATM0/,/^!/\{/^\!d;p;\}'
```

Using our xutools, practitioners need only type

```
xugrep '//ios:interface'
```

We designed and built eXtended Unix text-processing tools (xutools) *so that practitioners could process files in terms of the language constructs appropriate to the problem at hand—even if these languages lie beyond regular expressions.*

Besides "eXtending Unix," we also chose the *xu* prefix from the Greek word ξυλον, denoting "tree" or "staff." We find the first sense especially appropriate given that xutools operate on parse trees and process texts (traditionally printed on trees). The second sense is appropriate because xutools are designed to support IT staff in their real-world needs. Finally, ξυλον comes from the word ξυω , meaning to scrape, and our xutools, particularly `xugrep`, are well-suited to scraping content from document trees.

In order to support a variety of languages, our xutools rely upon a modular grammar library, as well as *xupath*, a general purpose querying language for structured text, which we based upon XPath. Although xutools may eventually encompass a broad range of Unix tools, this paper

presents xutools that extract (`xugrep`), count (`xuwc`), and compare (`xudiff`) structured text formats.

Our xutools generalize the class of languages that we can practically process on the command-line. Recall (e.g., [33]) that a **language** is a set of strings and a **regular language** is a set of strings that can be recognized by some finite automaton or equivalently, by a regular expression. In other words, we can write a regular expression to recognize whether a string is in a given language if and only if that language is regular. In contrast, a **context-free** language is a set of strings that can be recognized by a finite automaton with a stack. In practice, this means that we can write a context-free grammar to recognize a string in a context-free language. All regular languages are context-free (they just don't use the stack)—but not vice-versa. For example, the language of all strings with properly-nested parentheses is context-free but not regular. One cannot solve the **parenthesis-matching problem** with regular expressions.

In traditional Unix text-processing tools, most of the types of strings (bytes, fields, characters, files, lines) are either directly matched or indirectly split via a regular expression. In contrast, many of the languages in which practitioners are interested (XML elements, JSON subtrees, Cisco IOS interfaces, C function blocks), are not regular—but are context-free. This is no coincidence since the syntax of many programming languages was traditionally specified via context-free grammars. Context-free languages allow practitioners to recognize strings that possess a recursive or hierarchical structure. Furthermore, as languages evolve and acquire new constructs, a grammatical description of a language is easily extended (even for regular languages) [1].

Furthermore, Unix text-processing tools tend to operate on all the lines in a given input stream. For example `grep(1)` reports lines that contain a substring that matches a regular expression, `diff(1)` reports all differences between two files, and `wc(1)` counts all of the bytes, characters, words or lines in a file [32, 38, 39]. Often, however, practitioners are interested in reporting a matching line in terms of the block in which that match occurs; they are interested in reporting line-level differences between two router configurations in terms of interface blocks; or they are interested in counting the lines within each function block in C source. We extend Unix tools to report results in terms of contexts other than the file.

This Paper: We motivate (Section 2), design and implement (Section 3), evaluate (Section 4), and show the novelty of (Section 5) our tools relative to real-world examples. Table 1 illustrates our path through each of these sections. Finally, in Section 6 we discuss future work and in Section 7 we conclude.

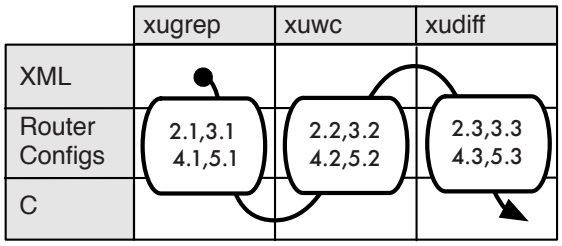


Table 1: For each section of our paper, we will discuss each of our tools relative to one or more real-world use cases that use the following structured-file formats.

2 Motivating use cases

During our fieldwork we observed the need to generalize Unix text-processing tools. Specifically, we worked with various network and system administrators and auditors of major electrical power utilities in the United States and discussed their challenges. We have also met with network administrators at Dartmouth College. In addition, we received much feedback from a poster on our tools [42] and recorded and analyzed these interactions at `http://www.xutools.com/`. As a result, we have discussed our work with practitioners from the RedHat Security Response team. Finally, we have met with the CEO of a network security company based in Germany regarding uses for our tools.

The intent of xutools is to help practitioners process the right type of string for the job at hand. Users (such as network and system administrators, security consultants, and software engineers) from a variety of domains (including the power grid, healthcare, and education) have emphasized the need for tools that let them operate on strings in languages that are relevant to them.

Many interesting strings are not in regular languages. Current Unix text-processing tools do not allow practitioners to operate on blocks of text nested arbitrarily deep or to report results with respect to that nesting. However, many practitioners must process these blocks in order to solve problems they encounter.

Figures 1 and 2 illustrate the mismatch between strings of interest in modern file formats and the strings upon which current Unix text-processing tools operate.

2.1 Use cases for rethinking `grep(1)`

NVD-XML: Very recently, practitioners at the RedHat Security Response Team wanted a way to grep the XML feed for the *National Vulnerability Database (NVD)*.[1] Specifically, they wanted to know how many

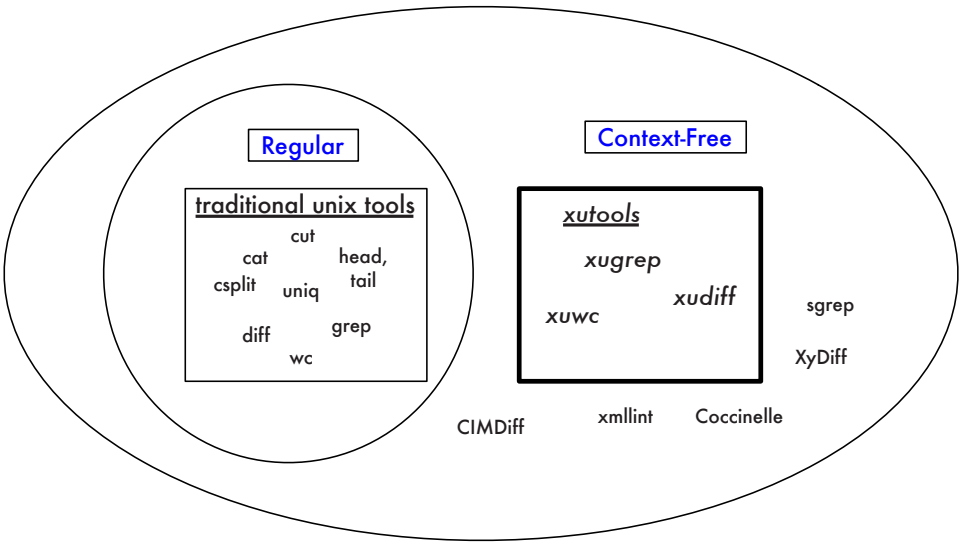


Figure 2: Our xutools improve on prior work by systematically extending Unix tools to operate on the broader class of languages that are encountered in modern file formats.

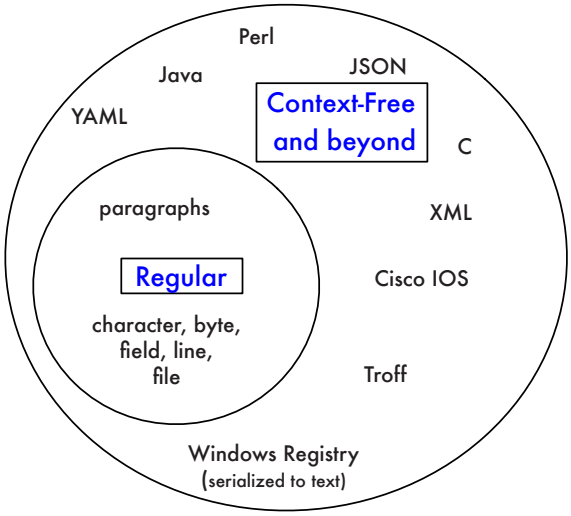


Figure 1: The languages upon we which we need to operate are now more general than can be described with regular expressions.

NVD entries contained the string `cpe:/a:redhat`, the vulnerability score of these entries, and how many XML elements in the feed contain `cpe:/a:redhat`.

Traditional grep cannot handle this use case because it requires us to solve the parenthesis-matching problem. This limitation motivates the capability to be able to report matches with respect to a given context.

In contrast, we need a `grep(1)` that can handle strings in context-free languages because when we extract XML elements, we want to ensure that every opening tag is matched by a closing tag. Multiple XML elements may share the same closing tag, and XML elements may be nested arbitrarily deep. Therefore, we need parenthesis matching to recognize XML elements. We need a `grep(1)` that can report matches with respect to the contexts defined within the NVD-XML vocabulary. (Although `xmllint(1)`'s shell-mode grep certainly provides one solution, it is not general enough to deal with languages other than XML [43]. We will discuss `xmllint(1)` in more detail in Section 3.)

C: Practitioners may want to be able to recognize (and thereby extract) all C function blocks in a file. As stated by one person on Slashdot following our LISA 2011 poster presentation, "it would be nice to be able to grep for a function name as a function name and not get back any usage of that text as a variable or embedded in a string, or a comment" [30, 42][2]. Traditional `grep(1)` cannot do this.

Alternatively, there may also be other constructs not explicitly defined via a standard grammar for a file format or language (such as ANSI C). For example, if we

could specify a context-free grammar for a C patch, then we could extract sections of code that are similar to that patch. Practitioners at the RedHat Security Response Team, for example, could use this to find all locations of unpatched code within embedded libraries scattered throughout source distributions.

Traditional `grep(1)` cannot handle these use cases. These use cases require us to solve the parenthesis-matching problem and also motivate a tool that can report matches with respect to a given context.

A regular expression that directly matches a C block cannot be created because the language of C blocks is not regular. Specifically, brackets close C functions, but they also close other kinds of blocks (such as if-statements) and so without matching, the closing bracket is ambiguous.

Secondly, these examples illustrate the benefits of a tool that can report matches with respect to the contexts defined within the C grammar. Practitioners need a suitable tool to report lines that contain a call to `malloc`, or even functions that contain the match.

2.2 Use cases for rethinking `wc(1)`

Router Configuration Files Network administrators want tools to understand their network configuration because many errors are network configuration errors [23, 35]. The network configuration literature attests that administrators may want to see how those network configurations change across time [26, 34], but no tools are available for admins to quickly and easily perform longitudinal studies on their own network data.

The usual metrics employed during longitudinal studies of network configuration data, such as lines of configuration, are general purpose but do not take other, language-specific measures into account. One exception here is Plonka et al. who look at stanzas. A stanza is a "set of adjacent related lines, or a paragraphs of configurations with a common purpose," such as an interface definition [26].

Network administrators configure and maintain language-specific constructs, such as interfaces, and as such, they would like to be able to measure their configuration files relative to these language-specific constructs. Administrators might like to measure the number of interfaces per router, or even the number of lines or bytes per interface. For example, one network administrator at Dartmouth Computing Services wanted to know how many interfaces within the set of campus routers use a particular active VLAN.

Traditional `wc(1)` cannot handle this use case. Traditional `wc(1)` counts lines in a file and so can calculate lines of configuration. Language-specific measures of configuration files, however, need a tool that can parse and count relative to language-specific constructs. `wc(1)` only counts languages tied to physical units of storage such as bytes, characters, words, and lines.

We need a `wc(1)` that can solve the parenthesis-matching problem. Practitioners want to count how many lines there are per interface block within a router configuration file. Since in general the set of block-based constructs in Cisco IOS is a context-free language, we must make the Unix `wc(1)` utility aware of context-free languages. (In fact, we could accomplish this by using our `xugrep` above to extract the interfaces in document order, escape the newlines in each block, and pipe the sequence of interfaces into `wc -l`.)

2.3 Use cases for rethinking `diff(1)`

Router Configuration Files Current tools such as *Really Awesome New Cisco config Differ (RANCID)* [28] let network administrators view changes to router configuration files in terms of lines. However, administrators, may want to view changes in the context of other structures defined by Cisco IOS. Alternatively, network administrators may want to compare configurations when they migrate services to different routers.

For example, if a network administrator moves a network interface for a router configuration file, then an edit script for the router configurations in terms of lines may report the change as 8 inserts and 8 deletes. However, an edit script that compares configurations in terms of interfaces ("interface X moved") might be more readable and less computationally intensive.

Traditional `diff(1)` cannot handle this use case because it requires us both to solve the parenthesis-matching problem, and to process and report changes relative to the context encoded by the parse tree.

Although the full Cisco IOS grammar is context-sensitive, meaningful subsets of the grammar, such as interface blocks and other nested blocks, are context-free [6]. Before we can compare interface blocks, we need to be able to easily extract them.

In this use case, we are interested in how the sequence of interface blocks changed between two versions of a router configuration file. If we wanted only to understand how the sequence of lines or sequence of interfaces changed, then we could use our `xugrep` to extract the interfaces or lines in document order, escape the newlines in each block, and pipe the sequence of interfaces or lines into `diff(1)` (where each line corresponds to an interface or line in an interface).

However, we want to understand how the lines in a configuration change with respect to the contexts defined by the Cisco IOS language. In this use case, we want to report changed lines in the context of their containing interface blocks. Alternatively, we could also choose to

report changes to interface blocks themselves.

Document-Centric XML: A wide variety of documents ranging from webpages, to office documents, to digitized texts are encoded according to some XML schema. Practitioners may want to compare versions of these documents in terms of elements of that schema. For example, a security analyst may want to compare versions of security policies in terms of sections, or subsections. Although tools exist to compare XML documents (see above), we offer a general-purpose solution for a wider variety of structured texts.

3 Design and implementation of our tools

We now present the design and implementation of each of our xutools: `xugrep`, `xuwc`, and `xudiff`. Our tools are intended to provide extended versions of `grep(1)`, `wc(1)`, and `diff(1)` respectively [32, 38, 39]. For each tool, we will give examples of the command syntax, describe the algorithm upon which the tool is based, and discuss our current implementation.

Before we discuss the tools, however, we will first present the design and implementation of two common components shared among all of our tools: a library of language grammars, and xupath, an XPath-like querying language for structured text in general (not just XML) [45].

3.1 Grammar library

The goal of our eXtended Unix text-processing tools (xutools) is to allow practitioners to operate on strings in languages that are relevant to them. Therefore, we want each of our grammars to be small and lightweight.

Design Although system administrators might not currently be able to write context-free grammars as quickly as they would write regular expressions, our strategy is to provide a library of grammars that satisfy a broad range of use cases. Just as C developers need to know the name, purpose, and calling sequence of a function, but not its implementation, in order to use that function in their own code, so do our users need to know the name of a production in our grammar library, not how the production is written, in order to process the corresponding construct with xutools. Since our grammar library's production names are aligned with the constructs practitioners use in practice, we expect our interface to the grammar library will be relatively easy to learn. For example, the production name for an interface in the Cisco IOS grammar is IOS:INTERFACE while the production name for a section in a document encoded using the *Text Encoding Initiative's XML guidelines (TEI-XML)* [5] is TEI:SECTION.

Implementation We implement our grammars using the PyParsing library [21]. We have currently implemented small grammars for subsets of two XML vocabularies (NVD-XML and TEI-XML), and for Cisco IOS. We have implemented an xupath grammar based upon Zazueta's Micro XPath grammar [47]. In addition, we are also using McGuire's subset-C parser [20]. Finally, we have a BUILTIN grammar for commonly-used, general-purpose constructs such as lines.

3.2 xupath

Our goal for xupath is to implement a powerful and general-purpose querying syntax for structured texts, including but not limited to XML.

Our intent is for practitioners to use this syntax to extract a set of high-level-language constructs that they want to process. Elements in an xupath query result set have the following fields: *text*, *production-name*, and *label*. The value of an element's *text* field is the string extracted from the text. The value of an element's *production-name* field is the sequence of productions applied to extract that string. Finally, the value of an element's *label* field depends upon the grammar. In our TEI-XML security policies, the value of a *label* is that node's passage header, for example *Introduction*. In our Cisco IOS security policies, the label corresponds to the network primitive's name such as *interface GigabitEthernet0/2*.

Design Our eXtended Unix text-processing tools generalize traditional Unix tools to (1) match context-free strings in the language of a context-free grammar, and (2) describe the context in which processing and reporting should occur. We observed that the XPath Query language [45] performs this function for XML documents. Therefore, we use an XPath-like syntax to express our queries on texts. This design decision offers us the additional benefit of not having to re-invent the wheel when thinking about how to select nodes from a set of parse trees.

Consider, for example, our first use case from the RedHat Security Response team. The team member wanted to know (1) how many NVD entries contain the `cpe:/a:redhat` string, (2) the vulnerability score of these entries, and (3) how many elements in the NVD-XML feed contain the `cpe:/a:redhat` string.

If all of our queries were on XML document trees, then we could answer these by using XPath within `xmllint`. For example, we could answer the third question by issuing the `grep a:redhat` command within the `xmllint` shell.

Practitioners, however, want to process Cisco IOS, C, JSON, or other languages (see Figure 1) and none of these languages are XML. Our xupath provides XPath-like functionality to context-free languages.

Consider these examples. Just as the XPath //DEFAULT:ENTRY selects all entry elements in the DEFAULT namespace, so does our xupath //NVD:ENTRY select all strings in the language of the ENTRY production within the NVD grammar. If we are concerned about namespaces, then we can prefix the production name with the appropriate namespace within the NVD grammar. Just as the XPath

```
//DEFAULT:ENTRY /CPSS:SCORE
```

references all score elements from all entry elements, so does our xupath

```
//NVD:ENTRY/NVD:SCORE
```

select all strings in the language of the score production from the set of all strings in the language of the entity production in the NVD grammar. Just as the XPath

```
//DEFAULT:ENTRY
[.//*[contains(*,'cpe:/a:redhat')]]
```

selects all entry elements whose descendants contain the string `cpe:/a:redhat`, so does our xupath

```
//NVD:ENTRY
[re:testsubtree('cpe:/a:redhat')]
/NVD:SCORE
```

select all strings in the language of the entry production in the NVD grammar that contain a match to the regular expression `cpe:/a:redhat`.

Our xupath generalizes such queries beyond XML to context-free languages in general such as subsets of Cisco IOS and C.

In Cisco IOS, we can use an xupath to reference all strings in the language of the interface production in our Cisco IOS grammar: //IOS:INTERFACE. We can also use an xupath to select all interfaces that contain an access-group:

```
//IOS:INTERFACE
[re_testsubtree('access-group')].
```

The xupath syntax also gives us an easy way to answer queries such as *where is acl 23 used?*.

In C, we can use an xupath to reference all strings in the language of the function production in a C subset grammar: //CSPEC:FUNCTION. We can also, however, grab all the lines within those functions with an xupath that mixes productions from the C specification grammar and our BUILTIN grammar: //CSPEC:FUNCTION/BUILTIN:LINE.

Implementation: Since the xupath syntax is based upon XPath, we implemented xupath in Python as a modified MicroXPath [47] grammar ported to PyParsing [21]. Given an xupath query, our grammar generates a parse tree of six types of nodes. Xupath parse trees are implemented using Python dictionaries. Consider the following examples in NVD and C shown in Figure 3.

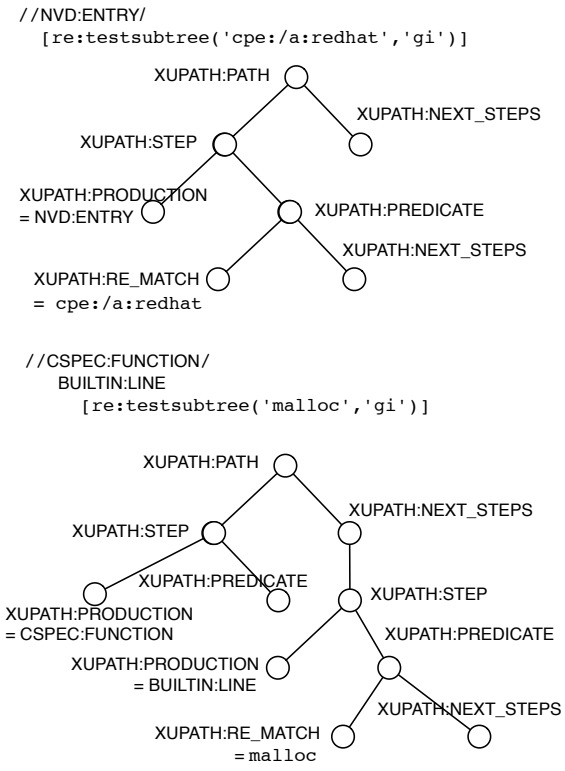


Figure 3: Two sample parse trees for xupath queries on NVD-XML and C.

3.3 xugrep

Our `xugrep` generalizes `grep(1)`; `xugrep` extracts all strings in a set of files that match an xupath.

Design: Traditional Unix `grep(1)` extracts all lines in a file that contain strings in the language of a regular expression. `Grep(1)` outputs each line containing a substring that matches a regular expression. The positions of those strings are line numbers.

Our `xugrep` generalizes the class of languages that we can practically extract on the Unix command-line from regular to context-free. A call to `xugrep` reports all strings that satisfy the given xupath within the context of the input files. The `-l` option tells our tool to

xugrep example input/output

```
bash-ex1$ xugrep -1 "//nvd:entry[re:testsubtree('cpe:/a:redhat','gi')]/nvd:score"
        nvdcve-2.0-2012.xml

file_path                 nvd:entry       nvd:score       region
nvdcve-2.0-2012.xml       CVE-2012-2110   1               <cvss:score>7.5</cvss:score>

bash-ex2$ xugrep -1 "//cspec:function" example.c

file_path  cspec:function  region
example.c  putstr          int/nputstr(char *s)/n{/n    ... putchar(*s++);/n}
example.c  fac             int/nfac(int n)/n{/n    if (n == 0)  ... n*fac(n-1);/n}
example.c  putn            int/nputn(int n)/n{/n if (9 < n)... putchar((n%10) + '0');/n}
example.c  facpr           int/nfacpr(int n)/n{/n   ...  putstr("\n");/n}
example.c  main            int/nmain()/n{/n    int i;/n ... return 0;/n}
```

Table 3: We can use `xugrep` on XML files as well as C source files to extract language-specific regions of text.

xugrep usage

```
xugrep [-1] <xupath> <input_file>+
```

Table 2: Our `xugrep` will report all strings that satisfy the given xupath within the context of the input files. Results may be reported in a one-line-per-match format via the `-1` option.

escape newlines within each of the result strings so that each match fits neatly on a single line.

The use cases of Section 2 motivate the need to be able to extract strings ranging from XML elements to C function blocks. We designed xugrep for practitioners to satisfy both these requirements as shown in Table 3.

Algorithm: `xugrep` generates a result set for the xupath query as it traverses the query's parse tree. As we traverse the xupath parse tree, we create a set of elements whose corresponding strings (stored in each element's *text* field) satisfy the query. We update the query result set when we encounter nodes whose production names are XUPATH:PRODUCTION and XUPATH:RE_MATCH in the xupath parse tree. When we encounter an XUPath parse-tree node whose *production-name* has the value XUPATH:PRODUCTION, we update the query result set to only include strings in the language specified by the production specified by the grammar corresponding to that parse-tree node. Similarly, when we encounter an XUPATH:RE_MATCH node, we update the query result set to only include strings that contain matches in the language of the corresponding regular expression for that node.

Implementation: We implemented `xugrep` in Python using a functional programming style. We have one function for every type of xupath parse tree node. Our current implementation of `xugrep` is 191 lines.

3.4 xuwc

Our `xuwc` generalizes `wc(1)` to count the number or size of strings in an xupath result set relative to some context.

Design: As stated by the Unix man pages, traditional `wc(1)` counts the number of words, lines, characters, or bytes contained in each input file or standard input [32]. Our `xuwc` generalizes `wc(1)` to count strings in context-free languages and to report those counts relative to language-specific contexts.

First, `xuwc` lets practitioners match strings in context-free languages. Section 2 reported how practitioners may want to count the size and number of language-specific structures within a Cisco IOS router configuration, C source file, or some other language. In the first two examples in Table 5 (**bash-ex1** and **bash-ex2**), `xuwc` allows practitioners to perform these tasks. `xuwc` does not have `wc(1)`'s `-clmw` flags because practitioners can specify their own structures to count (including bytes, lines, characters, and words) via the `--count` option.

Second, `xuwc` generalizes `wc(1)` to report counts relative to language-specific contexts. The Unix `wc(1)` utility counts the number of bytes, characters, words, or lines *relative to a file*. However, instead of reporting the number of lines in each router configuration file, we might prefer to report the number of lines in each interface. Alternatively, we may also want to count the number of entries in an *Access Control List (ACL)*. The last two examples in Table 5 (**bash-ex3** and **bash-ex4**) illustrate

xuwc usage

```
xuwc [--count=<grammar:production> | --re_count=<regexp>]
     [--context=<grammar:production>]
     <xupath> <input_file>+
```

Table 4: Given an xupath and a set of files, `xuwc` will count all matches in the result set in the context of the file.

xuwc example input/output

```
bash-ex1$ xuwc "//cspec:function" example.c
5       cspec:function  example.c
5       cspec:function  1       file_path       TOTAL

bash-ex2$ xuwc "//ios:interface/builtin:line" router.example
37      builtin:line    router.example
37      builtin:line    1       file_path       TOTAL

bash-ex3$ xuwc --context=ios:interface "//ios:interface/builtin:line" router.example
8       builtin:line    router.example.GigabitEthernet4/1
3       builtin:line    router.example.Null0
18      builtin:line    router.example.GigabitEthernet4/2
8       builtin:line    router.example.Loopback0
37      builtin:line    4       ios:interface   TOTAL

bash-ex4$ xuwc --count=builtin:byte --context=builtin:line "//ios:interface/builtin:line"
         router.example
        ...
24      builtin:byte    router.example.GigabitEthernet4/2.10
31      builtin:byte    router.example.GigabitEthernet4/2.11
13      builtin:byte    router.example.GigabitEthernet4/2.12
761     builtin:byte    37      builtin:line    TOTAL
```

Table 5: Four examples of our `xuwc` tool applied to C source code and Cisco IOS configuration files. Note our ability to *drill down* into the sizes of interfaces and then lines in Examples 3 and 4 (**bash-ex3** and **bash-ex4**) respectively.

the former scenario as well as the ability to combine these two generalizations of counting and context.

Algorithm: We implement `xuwc` by processing an `xugrep` report for the provided xupath. By default, if the `--count` parameter is unspecified, `xuwc` counts the number of strings that are in the language of the last production name in the xupath. If `--context` is unspecified, then `xuwc` reports counts relative to the entire file.

Our `xuwc` proceeds as follows. First, we call `xugrep` on the xupath to obtain a query result set. One of two cases follows next.

In the first case, we count the number of strings in the languages of the following productions–BUILTIN:BYTE, BUILTIN:CHARACTER, BUILTIN:WORD–or a regular expression, then we iterate through each element in the query result set. For each element, we scan the string stored in the element's *text* field and count the number of matches in the language of the production. Note that the context here *must* be that of the last production name in the xupath since these are the only strings stored in the result set elements.

Otherwise, we are counting strings in the language recognized by some other grammar production. This production name must be in our xupath or else an exception is thrown. For each element in the query result set, we scan the string stored in the element's *text* field and count the number of matches in the language of the production relative to the context in which it occurs.

Implementation: We implemented `xuwc` in Python. The method that implements our algorithm is 96 lines. The total number of lines for `xuwc` is 166 lines.

3.5 `xudiff`

`xudiff` generalizes `diff(1)` to compare two files (or the contents of two files) in terms of higher-level language constructs specified by productions in a context-free grammar.

Design: Traditional Unix `diff(1)` computes an edit script between the sequences of lines in a file. `diff(1)` outputs an edit script that describes how to transform the sequence of lines in the first file into the sequence of lines in the second file via a sequence of edit operations (insert, delete, update) [38]. All of these edit operations are performed upon *lines* in the context of the *entire file*.

While traditional `diff(1)` lets practitioners compare files in terms of the line, our `xudiff` allows practitioners to be able to compare files in terms of higher-level language constructs specified by productions in a context-free grammar.

xudiff usage

```
xudiff [--count=<grammar:production>]
       <xupath> <input_file1> <input_file2>
```

Table 6: Our `xudiff` compares two files in terms of the parse trees generated by applying an xupath to each file. Practitioners may also specify units in which to calculate edit costs.

Our `xudiff` lets practitioners compare files in terms of high-level language constructs. In Section 2, we saw that practitioners need to be able to compare both sections and subsections in TEI-XML documents as well as interface blocks in Cisco IOS router configurations. We designed `xudiff` so that practitioners could easily make such comparisons as shown in Table 8. In these examples, we show the output from comparing two versions of a router configuration file in Table 7.

Our `xudiff` also allows practitioners to specify the units that they want to use to compute the cost of a change. Example 1 (**bash-ex1**) issues the same query as Example 2 (**bash-ex2**), but the first example counts the cost of changes in terms of lines (for a distance of 4) whereas the second example computes the cost of changes in terms of words (for a distance of 9). The `I`, `D`, and `U` characters stand for insert, delete, and update respectively while the first number in the square brackets corresponds to the running edit cost, and the second number to cost of a particular edit. Alternatively, if we computed the cost of changes in terms of characters, then we would see that only 3 characters were changed in the second line of the `Loopback0` interface.

Algorithm and Implementation We have implemented the Zhang and Shasha tree edit distance algorithm [48] that is the basis for our `xudiff`. As a result, we can currently compute edit scripts for parse trees in TEI-XML, Cisco IOS, and C. The above examples are based upon the edit scripts we currently compute. Our Python implementation of the algorithm is 254 lines.

4 Evaluation

In this section we evaluate each of our tools qualitatively and quantitatively. The qualitative evaluation consists of anecdotal feedback regarding our tools from real-world practitioners. The quantitative evaluation includes the worst-case time complexity of our tools, lines of code, and test coverage.[3]

In addition to the above evaluation, Section 5 compares our xutools against existing tools that handle the formats we discuss in Section 2.

router.v0.config	router.v1.config
interface Loopback0 description really cool description ip address **333.444.1.185** 255.255.255.255 no ip redirects no ip unreachables ip pim sparse-dense-mode crypto map **azalea** ! interface GigabitEthernet4/2 description Core Network ip address 444.555.2.543 255.255.255.240 ip access-group outbound_filter in **ip access-group inbound_filter out** no ip redirects no ip unreachables no ip proxy-arp !	interface Loopback0 description really cool description ip address **333.444.1.581** 255.255.255.255 no ip redirects no ip unreachables ip pim sparse-dense-mode crypto map **daffodil** ! interface GigabitEthernet4/2 description Core Network ip address 444.555.2.543 255.255.255.240 ip access-group outbound_filter in no ip redirects no ip unreachables no ip proxy-arp **ip flow ingress** !

Table 7: The two Cisco IOS router interfaces upon which examples 2 through 4 are based. Changes are highlighted in **bold**.

xudiff example input/output

```
bash-ex1$  xudiff "//ios:config" router.v0.config router.v1.config

Distance: 4.0
I [4.0, 1.0] _.GigabitEthernet4/2.ip flow ingress (line)
D [3.0, 1.0] _.GigabitEthernet4/2.ip access-group inbound_filter out (line)
U [2.0, 1.0] _.Loopback0.crypto map azalea (line) -> _.Loopback0.crypto map daffodil (line)
U [1.0, 1.0] _.Loopback0.ip address 333.444.1.185 255.255.255.255 no ip redirects (line) -> _.Loopback0.ip
address 333.444.1.581 255.255.255.255 no ip redirects (line)
bash-ex2$  xudiff --count=builtin:word "//ios:config" router.v0.config router.v1.config

Distance: 9.0 builtin:word
I [9.0, 3.0] _.GigabitEthernet4/2.ip flow ingress (line)
D [6.0, 4.0] _.GigabitEthernet4/2.ip access-group inbound_filter out (line)
U [2.0, 1.0] _.Loopback0.crypto map azalea (line) -> _.Loopback0.crypto map daffodil (line)
U [1.0, 1.0] _.Loopback0.ip address 333.444.1.185 255.255.255.255 no ip redirects (line) -> _.Loopback0.ip
address 333.444.1.581 255.255.255.255 no ip redirects (line)
```

Table 8: Output of our `xudiff` utility when comparing the two versions of the Cisco IOS file in Table 7. Notice the ability to express the same set of changes in terms of lines (**bash-ex1**) or in terms of words (**bash-ex2**).

4.1 Grammar Library

The goal of our grammar library was to provide a common interface by which all of our xutools could parse text in terms of language-specific constructs. We recognize that system administrators want to avoid writing grammars, but desire the power and convenience of processing blocks of markup, configuration, and programming languages. Therefore, we wanted to create a modular, lightweight approach to parsing language-specific constructs.

Qualitative: Some of our grammars such as Cisco IOS and TEI-XML were handwritten, while others, such as C, were adapted from extant work. We realize that the utility of our tools depends heavily upon the kinds of languages for which a xutools-friendly grammar exists. Therefore, our ongoing work on this problem considers three strategies based upon feedback from practitioners in order to grow the library of grammars.

First we are writing a library of grammars for commonly-used languages. In last year's LISA poster session, practitioners suggested that we cover a few useful languages such as C, Java, Perl, XML, JSON, and YAML. Furthermore a repository of grammars for a variety of languages may also improve system security in the long run according to LangSec, Language-Theoretic Security, researchers [4].

Second, we could write a parser for languages used to specify grammars. This strategy would allow us to reuse work done to write grammars for traditional parsers such as Yacc and ANTLR, cutting-edge configuration tools like Augeas, and grammars for XML vocabularies such as XSD and RELAX-NG [19, 24, 29, 44, 46].

Finally, practitioners might be able to parse a text written in one language using a grammar for a very *similar* language. For example, if you try to open a Perl file in emacs' bash mode, then it looks decent. This *approximate matching* strategy could be a mechanism for xutools to handle languages that they haven't seen before.[4]

Quantitative: Our goal is to let practitioners write or use small, lightweight grammars to extract and process language-specific structure. The size of our grammars (shown in Table 9) illustrates that we can process texts in a variety of languages, in meaningful ways with grammars that are relatively small in terms of total number of productions, number of production names, and the number of lines needed to encode the grammar.

We currently use the PyParsing [21] library to implement our grammars and parse files according to these grammars. PyParsing implements a recursive-descent parser. In the worst-case, recursive descent parsing may require backtracking and so the time complexity may become exponential in the length of the input string (although this can be mitigated to some extent by memoization). Although we have not encountered any practical issues in using our tools on natural-language security policies and configuration files, we can improve our performance by using a different kind of parser. As stated by Aho, Sethi, and Ullman, "linear algorithms suffice to parse essentially all languages that arise in practice" [1].

Grammar	# productions	# production names	# lines
NVD-XML	6	3	8
TEI-XML	19	8	26
Cisco IOS	8	7	23
C	26	1	55
XUPath	11	0	28
Builtin	1	1	2

Table 9: The sizes of the grammars used by our xutools in terms of total number of productions, production names, and number of lines to encode the grammar.

4.2 xupath

We designed and implemented xupath so that practitioners could query a broader class of structured texts than XML. We only provide a qualitative evaluation of xupath.

Qualitative: xupath provides a consistent interface between language constructs and how those constructs are encoded within text. Encoding formats will continue to change. The SGML of the 80s is the XML of today. Although XML allows one to repurpose information, it may be an unreadable and/or bloated format for system administrators to directly edit. Furthermore, it is unnecessary (and potentially expensive in terms of resources) to convert C, Java, or some other format into XML.

Our tools use the xupath syntax to specify references to high-level language constructs so that the encoding of these constructs is transparent to the practitioner.

Since the *encodings* of meaningful constructs will change over time, we want to build tools to operate in terms of *references* to those constructs. Consider the historical transmission of text in which books and lines of Homer's Odyssey migrated from manuscripts, to books, to digital formats. The encoding of the text changed, but the constructs of book and line survived. In other words, the way in which people *referenced* the information remained stable although the encoding changed.[5]

Furthermore, formats such as CFEngine and Apache httpd configurations hint that there are structured text formats that are "in-between" traditional line-based formats and XML. In addition, there may be multiple ways to parse a text. Since our xupath syntax allows one to query a text using a sequence of production names from different grammars, practitioners can reference and parse these "in-between" formats in a variety of ways.

Finally, because xupath's syntax is based on XPath, we argue that the learning curve for xupath is lower than other querying languages might have been.

4.3 `xugrep`

`xugrep` extracts all strings in a set of files that match a xupath.

Qualitative: In Section 2 we motivated `xugrep` with two (of several) real-world use cases. We now briefly discuss how `xugrep` satisfied each of those use cases.

In Table 3 of Section 3 we presented actual output of how `xugrep` satisfies the two use cases of Section 2.

Our first `xugrep` use case was inspired by practitioners at the RedHat Security Response Team. They wanted a way to parse and query XML feeds of the National Vulnerability Database. During our discussion of xupath, we showed that `xmllint(1)` could satisfy this use case but that our `xugrep` tool operates on a more general class of languages.

Our second `xugrep` use case allows practitioners to extract blocks of C code and is based upon discussions we had with practitioners at the LISA 2011 poster session as well as the subsequent discussions on Slashdot [30, 42].

Quantitative: We implemented `xugrep` as a postorder traversal of the xupath parse tree and so this takes linear time in the number of nodes in the xupath query. For the examples we have considered in this paper, an xupath query resolves to a handful of nodes. Nonetheless, when we visit a node whose production name is of type XUPATH:PRODUCTION or XUPATH:RE_MATCH, we must iterate through each element in our query result set and scan for matches. As previously noted in our evaluation of our grammar library, we use the PyParsing library which is a recursive-descent parser [21]. The worst-case time complexity for recursive-descent parsing is exponential in the size of the input string [1]. Therefore, we can probably improve our practical performance by reimplementing our tools with a linear-time parser.

Our implementation of `xugrep` is 191 lines of Python. This is small enough to quickly port to other languages if desired. Furthermore, we have 438 lines of unit tests for this tool that validate behavior for example queries for subsets of TEI-XML, NVD-XML, Cisco IOS, and C.

4.4 `xuwc`

Our `xuwc` allows practitioners to count the number of language-specific constructs within a file (or another language-specific context).

Qualitative: Section 2 motivates `xuwc` with a use case involving Cisco IOS. In Table 5 of Section 3 we presented actual output of how `xuwc` satisfies this use case as well as a use case involving C function blocks.

Our `xuwc` allows network administrators to perform longitudinal studies upon their router-configuration files. For example, we could look at how the total number of lines per interface evolved over multiple versions of a router configuration. Alternatively, we could chart the number of ACLs over time.

Furthermore, `xuwc` may also be useful for looking at code complexity over time. Analogous to network configuration complexity, we can start to look at the number of lines per function or even the number of functions per module over the entire lifetime of software.

Quantitative: We implemented `xuwc` as a routine to process the query result set returned by `xugrep`. Therefore, the time-complexity of `xuwc` includes that of `xugrep`. If `xugrep` returns a result set of m elements, then in the worst case `xuwc` loops through all the elements in the result set exactly once. Therefore, the worse-case time complexity of our `xuwc`, not including the time `xugrep` takes to generate the query result set, is linear in the size of that result set.

Our Python implementation of `xuwc` is 96 lines and so we can easily port this to another language if desired. We have 408 lines of unit tests for this tool that cover examples in TEI-XML, NVD-XML, Cisco IOS, and C.

4.5 `xudiff`

Our `xudiff` allows practitioners to compare two files in terms of higher-level constructs specified by productions in a context-free grammar.

Qualitative: Section 2 motivates `xudiff` with RANCID as well as document-centric XML. In Table 8 of Section 3 we demonstrated how our `xudiff` satisfies the former of these use cases (although it currently has the capability to satisfy the latter use case as well).

We should note that one benefit of `xudiff` is the ability for practitioners to choose the right level of abstraction with which to summarize a change. Parse trees

encode recursive or hierarchical structure and this hierarchy often reflects levels of abstraction within a language. For example, a developer could generate a high-level edit script by reporting changes in the context of functions or modules. In contrast, if an implementation of an API method changed, then perhaps the developer would want to generate an edit script that describes changes in terms of lines within interfaces.

Quantitative: Our `xudiff` design relies upon the Zhang and Shasha algorithm to compute edit scripts. The time complexity for this algorithm is $O(|T_1| * |T_2| * min(depth(T_1), leaves(T_1)) * min(depth(T_2), leaves(T_2)))$ and its space complexity is $O(|T_1| * |T_2|)$ [48]. In these formulae, $|T_1|$ is the number of nodes in the first tree and $|T_2|$ is the number of nodes in the second tree.

Our Python implementation of Zhang and Shasha's algorithm is currently 254 lines. We have 563 lines of unit tests for this algorithm that tests every iteration of the example instance given in the Zhang and Shasha paper as well as an additional example based upon our TEI-XML dataset.

4.6 Overall xutools evaluation

Overall, we tried to design our xutools in a modular fashion so as to make them extensible and usable while avoiding needless complexity. The same grammars can be used by all of our tools. We have a simple, unified querying interface (xupath) shared among `xugrep`, `xuwc`, and `xudiff`. In the past, when we presented designs for our tools, practitioners warned us not to develop a set of command line flags that was too complex. We believe that the syntax for our current tools is straightforward.

Our xutools need not only run on Unix. Since we have implemented xutools in Python, we can also run these tools on a Windows' command prompt. In fact, we have documented and tested the full installation of an earlier version of `xugrep` on Windows. This is especially useful in the context of electrical power control systems in which Windows machines are common.

5 Related work

We now discuss our xutools in the context of prior work done in both industry and academia. We break out the relevant work discussion so as to make it clear what is novel about our work from an academic perspective as well as the gain in utility over the currently-available tools in industry. Furthermore, this section gives us an opportunity to compare our xutools against existing tools that handle the formats we discuss in Section 2.

5.1 `xugrep`

Industry: Currently, there are a variety of tools available to extract regions of text based upon their structure. The closest tool we have found to our design of `xugrep` is `sgrep` [18]. `sgrep` is suitable for querying structured document formats like mail, RTF, LaTeX, HTML, or SGML. Currently, an SGML/XML/HTML scanner is available but it does not produce a parse tree. Moreover, `sgrep` does not allow one to specify the context in which to report matches. Nonetheless, the querying model of `sgrep` is worth paying attention to.

If one is processing XML, XSLT may be used to transform and extract information based upon the structure of the XML. We have already mentioned libxml2's `xmllint(1)` [43] and its corresponding shell for traversing a document tree. Furthermore xmlstarlet has been around for a while and can also be used to search in XML files [17].

Cisco IOS provides several commands for extracting configuration files. For example, the `include` (and `exclude`) commands enable network administrators to find all lines in a configuration file that match (and don't match) a string. Cisco IOS also supports regular expressions and other mechanisms such as `begin` to get to the first interface in the configuration [8]. In contrast, our `xugrep` enables practitioners to extract matches in the context of arbitrary Cisco IOS constructs.

Windows Powershell has a `Where-Object` Cmdlet that allows queries on the properties of an object. An object may be created from a source file by casting it as a type (such as XML) [27].

Pike's structural regular expressions allow users to write a program to refine matches based on successive applications of regular expressions [25]. Our approach is different because we extract matches based upon whether a string is in the language of the supplied grammar.

Academia: Although Grunschlag has built a context-free grep [16], this classroom tool only extracts matches with respect to individual lines. Coccinelle [10] is a semantic grep for C. In contrast, we want our tool to have a general architecture for several languages used by system administrators.

As noted in Section 2, our `xudiff` complements research in network configuration management. For example, Sun et al. argue that the block is the right level of abstraction for making sense of network configurations across multiple languages. Despite this, however, they only look at correlated changes in network configurations in Cisco [34]. Similarly, Plonka et al. look at stanzas in their work [26].

5.2 xuwc

Our `xuwc` tool allows practitioners to count the number or size of a language-specific construct within an xupath result set.

Academia: The general implication of `xuwc` is that it will allow practitioners to easily compute statistics about structures within a corpus of files in language-specific units of measure. We were unable to find prior work that attempts to generalize `wc(1)`.

Industry: Certainly there are a number of word count tools used every day in word processors such as Word and emacs. These allow practitioners to count words, lines, and characters within a file.

There are also several programming-language utilities to compute source-code metrics. The Metrics plugin for Eclipse allows one to count the number of packages, methods, lines of code, Java interfaces, and lines of code per method within a set of source files [22]. Vil provides metrics, visualization, and queries for C#, .NET, and VisualBasic.NET and can count methods per class and lines of code [41]. Code Analyzer is a GPL'd Java application for C, C++, Java, assembly, and HTML and it can calculate the ratio of comments to code, the lines of code, whitespace and even user-defined metrics [11].

Finally, Windows Powershell has a nice CmdLet called Measure-Object that allows people to gather statistics about an object [27].

5.3 xudiff

The goal of our `xudiff` is to produce a general-purpose tool to compare multiple versions of a file in terms of a specified structure and to report changes relative to some parent structure. Our tool is unique because we seek to build a tool that can do a structural comparison of files in general. In fact, we could use our `xudiff` to compare Cisco IOS configuration files as well as C source code, Troff, JSON, or XML vocabularies.

Industry: The SmartDifferencer, produced by Semantic Designs, compares source code in a variety of programming languages in terms of edit operations based on language-specific constructs [13]. Unfortunately, the SmartDifferencer is proprietary. Finally, TkDiff [37], available for Windows, improves upon line-based units of comparison by highlighting character differences within a changed line.

Academia: The various components of our hierarchical diff tool use and improve upon the state-of-the-art in computer science. Computing changes between two trees is an instance of the tree diffing problem and has been studied by theoretical computer science [3]. Researchers have investigated algorithms such as subtree hashing, and even using XML IDs to align subtrees between two versions of a structured document and generate an edit script [7, 9]. Zhang and Shasha [48] provide a very simple algorithm for solving edit distance between trees that we currently use in `xudiff`.

Furthermore Tekli et al. in a comprehensive 2009 review of XML similarity note that a future research direction in the field woiuld be to explore similarity methods that compare "not only the skeletons of XML documents . . . but also their information content" [36]. Other researchers have looked at techniques to compare CIM-XML for compliance [31], XML trees for version control [2], and Puppet network configuration files based upon their abstract syntax trees [40]. Recently, our poster that proposed xutools [42] was cited in the context of differential forensic analysis [15].

5.4 xutools in general

Our xutools operate on parse trees so that practitioners can process texts in terms of high-level language constructs.

Industry: Augeas [19] is similar in spirit to our tools as it focuses on ways to configure Linux via an API that manipulates abstract syntax trees. This library includes a canonical tree representation, and path expressions for querying such trees. The goals of Augeas and xutools are complimentary, Augeas provides an API to manipulate configuration files safely and xutools extends existing text-processing tools so that practitioners can operate and analyze a variety of texts in terms of their high-level structures.

6 Future work

Our xutools set the stage for a variety of future research directions and tool development beyond the scope of this paper. First we discuss three ways in which we want to improve and extend our xutools. We then will describe two second-order services built on top of our xutools to demonstrate the potential broader impact of our research.

6.1 Improvements and extensions

First, we want to extend the library of grammars on which our current xutools operate. In Section 4 we discussed three strategies to build up the grammar library for standard languages. For non-standard languages we want to consider how one might construct a grammar to recognize strings *similar* to a patch. As discussed in Section 2

this would allow us to extract code that is similar to a known vulnerability.

Second, we want to collect more feedback on our currently-implemented xutools and then re-implement them in C with a more efficient parser. Once this is done, we can benchmark our xutools against traditional Unix tools.

Third, we want to think about extensions to current xutools, new xutools, and how both interact with traditional Unix. As mentioned earlier, we could think about `head` and `tail` as extensions of `xugrep` in which we only report structures within a given range of nodes on our parse trees. Traditional `grep(1)` has an `--invert-match` option, the analogue of which in `xugrep` would also be quite useful. Other practitioners have suggested that a context-free `sed(1)` might be useful. We also have been thinking about how Unix pipelines and existing Unix tools interoperate with our xutools. Our `-l` option in `xugrep` for example, allows us to map newline-escaped strings in context-free languages to a set of lines that other Unix tools can use.

6.2 Second-order xutools services

Even if a system administrator has no interest in working directly with XML, C, Java, or some other configuration format on the command line, our xutools can still provide useful functionality. We now provide two examples to illustrate our claim.

A Structural Difference Engine for Version-Control Systems: If we used `xudiff` as a difference engine for version-controlled configuration files, then we could provide useful debugging information to a system administrator when used in concert with bug reports. In essence, `xudiff` gives us an easy way to create a heatmap for the most volatile regions of configuration files and this is a useful tool for debugging. In LISA 2011, Workshop 8: Teaching System Administration stated that the parts of the system that break down historically are the first places to look for bugs. Our tools would allow practitioners to pinpoint the regions of configuration that were changed when the bugs first were reported. We also think such an application of our tools would help organizations understand legacy networks acquired via mergers.[6]

Measuring the Evolution of Terms-of-Service Policies: An important part of evaluating a web-based service is to understand its terms and conditions. These terms and conditions are described in a service's terms of service agreement. Casual readers, however, often do not read these terms, but rather agree by using the service. Despite this, enterprises and individual consumers need to be able to compare terms of service so that they can reliably evaluate the risks of using a service. The *Electronic Frontier Foundation (EFF)* recognizes the importance of understanding changes to security policies and so built a terms of service tracker, the TOSBack [14].

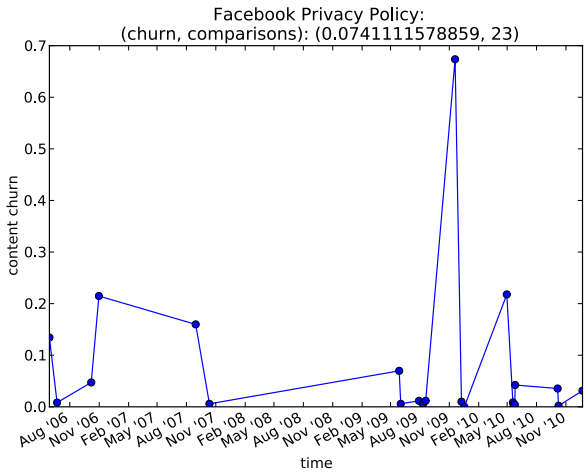


Figure 4: The structural evolution of Facebook Privacy Policies recorded by the EFF TOSBack from May 2006 until December 2010. We notice a spike in November 2009 when Facebook introduced major changes to their privacy policy that received enough attention to merit a subsection in Wikipedia's article: Criticism of Facebook [12].

As a proof-of-concept, we downloaded 24 Facebook policies recorded by EFF's TOSBack between May 5, 2006 and December 23, 2010. Using a simple metric based upon the Zhang-Shasha tree edit distance [48], we were able to quantify and thereby visualize the evolution of the Facebook privacy policy. In Figure 4, we notice a that our change metric (content churn) reaches a maximum on November 19, 2009. This policy revision was significant enough to merit an entire subsection in the following Wikipedia article: Criticism of Facebook. The article states that in November 2009, Facebook proposed a new privacy policy and new controls. These changes were protested and even caused the Office of the Privacy Commissioner of Canada to launch an investigation into Facebook's privacy policies [12]. This initial pilot study suggests that our xutools-based change metrics seem to be able to pinpoint important events in the "big-picture" history of a terms-of-service policy. Moreover, we can use our `xudiff` tool to "drill-down" and find specific policy changes.

7 Conclusions

Structured-text file formats break many of the assumptions upon which Unix text-processing tools were based. We have designed and built xutools, specifically `xugrep`, `xuwc`, and `xudiff`, to process texts in language-specific constructs. *In effect, we have extended the class of languages upon which Unix text-processing tools operate*. These generalizations are directly motivated by real-world problems faced by network and system administrators, software engineers, and other IT professionals who demand tools to process structured-text file formats.

8 Acknowledgments

The authors would like to thank Doug McIlroy for his encyclopaedic knowledge about Unix, his encouragement and advice. We would like to thank Tom Cormen, for his encyclopaedic knowledge about algorithms and insisting on a more rigorous mathematical model for our work. We would like to thank Rakesh Bobba and Edmond Rogers for their work to apply these tools in the domain of the power grid. We would like to thank Sergey Bratus for his insights into workflow patterns in Unix. We would also like to thank Kurt Seifried, Paul Schmidt, and many other practitioners for their real-world feedback. Finally, we would like to thank our LISA shepherd, Mike Ciavarella. This work was supported in part by the TCIPG project from the DOE (under grant DE-OE0000097). Views are the authors' alone.

9 Availability

Our xutools are available under the GPLv3 at `http://www.xutools.net/`. Our repository contains several use cases as well as relevant literature suggested by those we met as a result of our poster presentation at LISA 2011.[7]

References

[1] Aho, A. V., Sethi, R., and Ullman, J. D. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Massachusetts, 1988.

[2] Apel, S., Liebig, J., Lengauer, C., Kastner, C., and Cook, W. R. Semistructured merge in revision control systems. In *Proceedings of the Fourth International Workshop on Variability Modeling of Software Intensive Systems (VaMoS 2010)* (January 2010), University of Duisburg-Essen, pp. 13–20.

[3] Bille, P. A survey on tree edit distance and related problems. *Theoretical Computer Science 337*, unknown (June 2005), unknown.

[4] Bratus, S., Locasto, M. E., Patterson, M. L., Sassaman, L., and Shubina, A. Exploit programming: From buffer overflows to weird machines and theory of computation. *USENIX ;login:* (December 2011), 13–21.

[5] Burnard, L., and Bauman, S. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*, 5 ed., 2007.

[6] Caldwell, D., Lee, S., and Mandelbaum, Y. Adaptive parsing of router configuration languages. In *Internet Network Management Workshop, 2008. (INM 2008)* (October 2008), IEEE, pp. 1–6.

[7] Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., and Widom, J. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)* (June 1996), ACM, pp. 493–504.

[8] Cisco IOS configuration fundamentals command reference. Retrieved September 19, 2012 from `http://www.cisco.com/en/US/docs/ios/12_1/configfun/command/reference/frd1001.html`.

[9] Cobéna, G., Abiteboul, S., and Marian, A. Detecting changes in XML documents. In *Proceedings of the 18th International Conference on Data Engineering* (February and March 2002), IEEE, pp. 41–52.

[10] Coccinelle: A program matching and transformation tool for systems code, 2011. Retrieved November 11, 2011 from `http://coccinelle.lip6.fr/`.

[11] Codeanalyzer. Retrieved May 17, 2012 from `http://sourceforge.net/projects/codeanalyze-gpl/`.

[12] Criticism of Facebook. *Wikipedia: The Free Encyclopedia* (2012). Retrieved September 19, 2012 from `http://en.wikipedia.org/wiki/Criticism_of_Facebook#November.2FDecember_2009`.

[13] Designs, S. Semantic designs: Smart differencer tool. Retrieved May 16, 2012 from `http://www.semdesigns.com/Products/SmartDifferencer/`.

[14] TOSBack — the Terms-Of-Service Tracker. Retrieved September 19, 2012 from `http://www.tosback.org/`.

[15] Garfinkel, S., Nelson, A. J., and Young, J. A general strategy for differential forensic analysis. In *Proceedings of the 12th Conference on Digital Research Forensics (DFRWS '12)* (August 2012), ACM, pp. 550–559.

[16] Grunschlag, Z. cfgrep - context free grammar egrep variant, 2011. Retrieved November 11, 2011 from `http://www.cs.columbia.edu/~zeph/software/cfgrep/`.

[17] Grushinskiy, M. XMLStarlet command line XML toolkit, 2002. Retrieved May 15, 2012 from `http://xmlstar.sourceforge.net/`.

[18] Jaakkola, J., and Kilpelainen, P. Using sgrep for querying structured text files. In *Proceedings of SGML Finland 1996* (October 1996), unknown, p. unknown.

[19] LUTTERKORT, D. Augeas–a configuration API. In *Proceedings of the Linux Symposium* (July 2008), pp. 47–56.

[20] MCGUIRE, P. Subset C parser (BNF taken from the 1996 international obfuscated C code contest). Retrieved May 16, 2012 from http://pyparsing.wikispaces.com/file/view/oc.py.

[21] MCGUIRE, P. pyparsing, 2012. Retrieved September 19, 2012 from http://pyparsing.wikispaces.com.

[22] Metrics 1.3.6. Retrieved May 17, 2012 from http://metrics.sourceforge.net/.

[23] OPPENHEIMER, D., GANAPATHI, A., AND PATTERSON, D. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)* (March 2003), USENIX Association, p. unknown.

[24] PARR, T. ANTLR parser generator. Retrieved May 17, 2012 from http://www.antlr.org/.

[25] PIKE, R. Structural regular expressions.

[26] PLONKA, D., AND TACK, A. J. An analysis of network configuration artifacts. In *The 23rd Conference on Large Installation System Administration (LISA '09)* (November 2009), USENIX Association, p. unknown.

[27] Windows PowerShell. Retrieved February 3, 2012 from http://technet.microsoft.com/en-us/library/bb978526.aspx.

[28] RANCID - Really Awesone New Cisco Config Differ, 2010. Retrieved December 1, 2010 from http://www.shrubbery.net/rancid/.

[29] RELAX NG home page. Retrieved May 17, 2012 from http://www.relaxng.org/.

[30] Researchers expanding diff, grep Unix Tools, 2011. Retrieved May 17, 2012 from http://tech.slashdot.org/story/11/12/08/185217/researchers-expanding-diff-grep-unix-tools.

[31] ROUTRAY, R., AND NADGOWDA, S. CIMDIFF: Advanced difference tracking tool for CIM compliant devices. In *Proceedings of the 23rd Conference on Large Installation System Administration (LISA '09)* (October and November 2009), USENIX Association, p. unknown.

[32] RUBIN, P., AND MACKENZIE, D. *wc(1) Manual Page*, October 2004. Retrieved May 16, 2012.

[33] SIPSER, M. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, Massachusetts, 2006.

[34] SUN, X., SUNG, Y. W., KROTHAPALLI, S., AND RAO, S. A systematic approach for evolving VLAN designs. In *Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM 2010)* (March 2010), IEEE Computer Society, pp. 1–9.

[35] SUNG, Y.-W. E., RAO, S., SEN, S., AND LEGGETT, S. Extracting network-wide correlated changes from longitudinal configuration data. In *Proceedings of the 10th Passive and Active Measurement Conference (PAM 2009)* (April 2009), unknown, pp. 111–121.

[36] TEKLI, J., CHBEIR, R., AND YETONGNON, K. An overview on XML similarity: Background, current trends and future directions. *Computer Science Review 3*, 3 (August 2009), 151–173.

[37] TkDiff. Retrieved February 3, 2012 from http://tkdiff.sourceforge.net/.

[38] UNKNOWN. *diff(1) Manual Page*, September 1993. Retrieved May 17, 2012.

[39] UNKNOWN. *grep(1) Manual Page*, January 2002. Retrieved May 17, 2012.

[40] VANBRABANT, B., JORIS, P., AND WOUTER, J. Integrated management of network and security devices in it infrastructures. In *The 25th Conference on Large Installation System Administration (LISA '11)* (December 2011), USENIX Association, p. unknown.

[41] Vil. Retrieved May 17, 2012 from http://www.1bot.com/.

[42] WEAVER, G. A., AND SMITH, S. W. Context-free grep and hierarchical diff (poster). In *Proceedings of the 25th Large Installation System Administration Conference (LISA '11)* (December 2011), USENIX Association, p. unknown.

[43] xmllint. Retrieved May 16, 2012 from http://xmlsoft.org/xmllint.html.

[44] W3C XML Schema. Retrieved May 17, 2012 from http://www.w3.org/XML/Schema.

[45] XML Path language (XPath), 1999. Retrieved May 15, 2012 from http://www.w3.org/TR/xpath/.

[46] The LEX and YACC Page. Retrieved May 17, 2012 from http://dinosaur.compilertools.net/.

[47] ZAZUETA, J. Micro XPath grammar translation into ANTLR. Retrieved May 16, 2012 from http://www.antlr.org/grammar/1210113624040/MicroXPath.g.

[48] ZHANG, K., AND SHASHA, D. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Jounal of Computing 18*, 6 (December 1989), 1245–1262.

Notes

[1] http://nvd.nist.gov/download.cfm#CVE_FEED

[2] Thanks to Tanktalus for the Slashdot post.

[3] Our qualitative evaluation does not include performance benchmarks against traditional Unix tools, we see this as future work for after we get practitioner feedback on our tools and re-implement them in C. Once we have implemented them in C, we can do a fair comparison between our xutools operating on files in terms of lines and traditional Unix tools.

[4] Thanks to David Lang and Nicolai Plum at the LISA 2011 poster session.

[5] In fact, the notion of managing and measuring changes to multi-versioned texts over time is based upon our previous work in the Classics at Holy Cross, Harvard's Center for Hellenic Studies, and the Perseus Project at Tufts University.

[6] According to Doug McIlroy, this idea is reminiscent of a visualization of change activity for switching code at Bell Labs. This visualization was possible because every change was manually registered in their source code control system. `xudiff` enables one to summarize such changes directly from the source.

[7] We received a lot of interest in our LISA 2011 poster [42], including worldwide press coverage of our work. See http://www.cs.dartmouth.edu/~gweave01/grepDiff/Press.html for details.

Managing User Requests with the Grand Unified Task System (GUTS)

Andrew Stromme Danica J. Sutherland Alexander Burka Benjamin Lipton
Nicholas Felt Rebecca Roelofs Daniel-Elia Feist-Alexandrov Steve Dini Allen Welkie
Swarthmore College Computer Society
staff@sccs.swarthmore.edu

Abstract

As system administrators who are also full-time students, we aim to minimize the time we spend approving and carrying out standard tasks that comprise much of our day-to-day work. The less time required for these repetitive tasks, the more time we have available to provide new and exciting services to our community.

To facilitate the automation of this process, we have created the Grand Unified Task System (GUTS), which consists of a small core (a web interface and task executor) that unites task request processing for a range of modular services. This design allows for enhanced security and makes the system easily understandable and extensible, especially to new administrators. The Python backend provides deep integration with standard UNIX tools; the Django-based frontend provides a web interface friendly to both users and administrators. These design decisions have proven successful: deploying GUTS in production allowed us to dramatically reduce our response time for approving tasks, reach a much larger portion of our potential user base, and more easily support a diverse array of new services.

Keywords: infrastructure, project management, security, teaching system administration, web

1 Introduction

Since 1993, the Swarthmore College Computer Society (SCCS) has provided shell, mail, and web services to students, alumni, and other users affiliated with Swarthmore College. Because SCCS is run entirely by student volunteers on a small budget, both server resources and system administrator time are quite limited. To make efficient use of these resources, users receive access to most SCCS services only upon request. However, manually setting up user accounts, mailing lists, and databases on a case-by-case basis is both repetitive and excessively time-consuming. An ideal automated solution minimizes the amount of time that system administrators spend performing repetitive tasks while ensuring the security and supportability of the underlying systems. The Grand Unified Task System (GUTS) answers both of these questions through via its comprehensive web interface and deep integration with existing system tools. It replaces an older SCCS-designed system that performs a similar purpose, the Los [9] automated task request and approval system.

GUTS provides a full vertical solution and includes a submission and administration system for user requests, a website frontend for users, and a library of backend scripts that perform the actual tasks. An example of one such request is user creation, which involves a number of individual steps but is presented as a single task to the user. We have a policy of administrator approval for user creation tasks, but many other types of user requests are handled automatically and immediately by GUTS, without any interaction from the administrator (for example, password change requests). Having an administrator check over major requests helps both with security against malicious requests and with ensuring that various subjective policies that are hard to enforce automatically are followed. It is crucial that the system for handling requests makes review and approval of individual requests as easy as possible so that the administrator's time can be spent on reviewing the details of tasks rather than fighting with the system.

Users submit tasks to GUTS via a convenient web interface based on the Django [4] framework. Upon visiting the GUTS website, the user sees a listing of the available services in the form of tiles on the *SCCS Dashboard*, shown in Figure 1. After selecting the task, the user then encounters a form for entering data pertaining to the task. The data is type-checked and verified before further processing occurs. For tasks which do not require administrator approval, the user receives a visual report of the status of the task and often an accompanying email confirmation. If the task does require moderation, GUTS

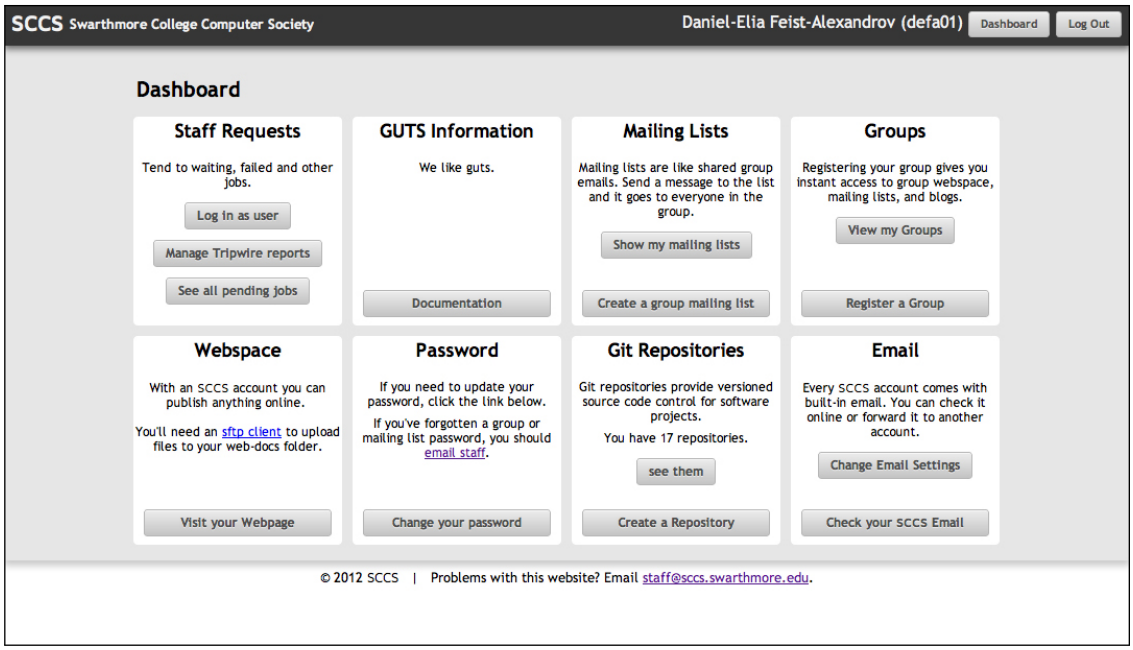


Figure 1: The SCCS Dashboard serves as the landing page for GUTS and provides users with an overview of the available tasks they may request. Each tile represents a self-contained pluggable GUTS service. Non-administrators do not see the *Staff Requests* or *GUTS Information* tiles.

sends an email to the administrators notifying them that there is a task awaiting approval. After the administrators review the task, the requesting user will receive an email detailing the status of their request.

At present, almost all of our system administration tasks follow this model. Through GUTS, users can request new accounts, change account passwords, set up email forwarding, and create and manage mailing lists, source code repositories, and groups.

This report first details how existing systems serving similar purposes do not satisfy our needs (Section 2). We then give an overview of the design of GUTS (Sections 3-5), and discuss its successes thus far (Section 6). We finally outline some of the lessons learned during the development process (Section 7) and identify some features of the design critical to its success (Section 8).

2 Related Work

Because managing large systems is traditionally labor-intensive, many different types of tools have been created to assist system administrators in performing their jobs.

One such type of tool, exemplified by Cfengine [2] and Puppet [11], automates the task of maintaining a large group of systems, keeping each machine as close as possible to a policy for the files and processes that should be present. This allows system administrators to avoid performing repetitive tasks such as software updates on multiple machines, and also allows the system state to be automatically repaired if an incident or user action causes it to deviate from policy. Though these tools are very useful for simplifying the administration and improving the robustness of a large installation of systems, they do not immediately address the problem of responding to user requests, which is the main cause for administrative action at small sites like SCCS.

Another category of tool is the ticketing system, used to log user requests and related actions by administrators. Some examples are Request v3 [8] and RT [1], both of which can automatically create tickets for email requests and manage tickets via email, command-line, or web interfaces. These tools keep track of the status of requests from users, so that the administrator is free to concentrate on satisfying these requests. Additionally, they allow reporting to track trends in requests and highlight problems. GUTS serves a similar purpose in our organization, but in addition to managing communication with users about certain types of requests, it also handles the execution of requested tasks.

A third category of tool simplifies administration by providing a graphical user interface for common administrative tasks, intended to make it easier for system administrators to configure their machines. One example of such a system is Webmin [3], which enables admin-

istration of many services on UNIX machines via a web interface. Webmin has a selection of modules that handle the administration of various services, each of which can configure that service on a selection of UNIX-like systems. The developers of Webmin have also created a tool called Usermin [12] that allows users to manage features of their own accounts via a graphical interface. GUTS combines these roles by providing a clean user interface for several features provided by the SCCS servers, both administrator- and user-initiated. In addition to providing a friendly interface to these features, GUTS also enables users to request administrator approval for actions for which they do not have sufficient privileges.

Finally, GUTS is the successor to Los [9], improving on the earlier SCCS effort in many respects. Both GUTS and Los support the automation of task approval and execution, but Los requires that administrators approve tasks through either a custom command-line tool or a GPG-signed email sent to the Los server. In contrast, GUTS provides a simple, easily-accessible web interface for one-click task approval. Since users interact with the same interface, effort spent on improving the frontend of GUTS benefits both users and administrators. GUTS is also much easier to extend than Los. While Los is written in a mix of languages and uses an inflexible, homegrown XML-based method to define tasks, GUTS code is pure Python, task definitions are simply functions, and task logic and presentation can be packaged together into modular services using the popular Django app model.

3 GUTS Core Components

At a high level, the core of GUTS combines a framework for automating system tasks with a convenient website that exposes these tasks to users. Built around this core are a number of services, each bundling together one or more tasks into a single modular package with a shared frontend interface.

The GUTS project core consists of the `libguts` library which contains the services framework and other helper code, the `gutsd` backend server which exposes registered services and functions, and frontend clients (most notably the `gutsweb` website) which use these registered functions to perform system actions. Figure 2 gives an overview of GUTS components, which the following subsections describe in detail.

3.1 GUTS Library

The `libguts` library provides a rich set of tools that allow programmers to easily write both GUTS system functions and the frontend interfaces that will call these functions. With `libguts`, the frontend programmer can call GUTS functions as if they were normal Python functions, even though there are a number of layers and transformations behind the scenes. In the following sections we describe first the semantics of GUTS functions, including how to register and call them, and second the structure of GUTS services, which organize these functions into modular packages.

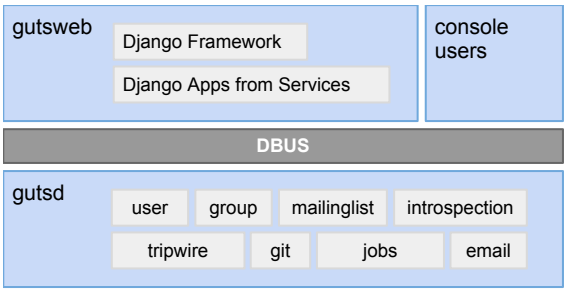


Figure 2: Architecture of GUTS components. Everything above the D-Bus line runs as unprivileged users while everything below runs as the root user. `gutsd` automatically loads and extracts the registered actions from each of the installed services, while `gutsweb` loads everything else contained in each service.

3.1.1 Functions

A GUTS function is a single task that can be performed on the system, such as creating a user, creating a group, removing a repository or modifying a mailing list. Functions are registered via one of the `libguts` Python decorators. The GUTS functions run as root within the `gutsd` process, so they can access any necessary parts of the GUTS job database or the running server as needed (for example, the `user` actions access the live LDAP database, and copy files around the server). All other code runs in the context of the calling program (in our case, `gutsweb` which runs as an unprivileged user). Because GUTS functions run with elevated privileges, we must ensure that there are no security holes and that parameters are checked for compliance before performing any system action. GUTS allows registered functions to provide type information and automatic parameter checking. The `libguts` library includes a number of checkers which verify commonly-used parameters. For example, one included checker ensures that a parameter is a valid UNIX group name.

A GUTS function is a normal Python function that has been registered with the GUTS daemon by a decorator from `libguts`. These functions fall into three types:

- *Info functions* have no side-effects on the system when executed. Examples include functions that list groups to which the user belongs or return metadata

for a source control repository.

- *Actions* are functions with side-effects on the system, such as a function that creates a user, removes a repository, or adds a user to a mailing list. Actions automatically add jobs to the database when run. Some actions are *delayed*, which means the parameters are added to the jobs database when run, but not executed until an administrator has approved the action.
- *Meta functions* perform some internal-to-GUTS task, such as listing the loaded services or running a delayed job. An augmented context is passed to meta functions, which includes the list of services and registered functions for the current GUTS instance.

3.1.2 Services

A GUTS installation is implemented and extended through the use of *services*. A service consists of a set of registered GUTS functions packaged inside a Django app that implements the logic and presentation of the service's web interface. An example of a GUTS service is the `git` module, described in depth in Section 5. In short, this service provides a collection of web pages for creating, editing and exploring Git repositories hosted on SCCS infrastructure. The Django-based website functions call out to GUTS functions to perform the actual heavy lifting of repository creation and modification. Services can interact with any existing system executables and libraries and are thus a powerful way to implement website and system functionality through GUTS. The SCCS GUTS installation includes services such as `git` (for Git repositories), `user` (account creation and management) and `mailinglists` (mailing list administration). The `git` service includes actions such as `create_repository`, `remove_repository`, and `modify_repository`.

We expect that services will often be written by programmers with little knowledge of internal GUTS systems. Because of this, we designed the services architecture to avoid boilerplate code as much as possible. A service leverages many of the elements of a standard Django app, including a settings file with service-specific settings, a url-routing module, view functions (that typically call GUTS functions) to display the relevant pages on the website, and template files for those views. Besides this, the service also contains the system-level code that implements the GUTS functions that the service provides. In this way, the service folder is a completely self-contained definition of everything related to that service (with the exception of any system executables or libraries it uses). The organization of this folder will be familiar to Django programmers. Figure 3 shows the elements of a service.

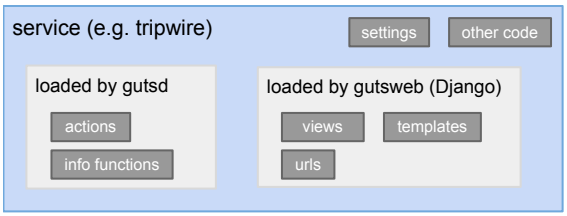


Figure 3: The anatomy of a GUTS service. A service is a Django app that also contains registered GUTS functions. Most of the service is loaded by `gutsweb` to power the website, but the actions are loaded by `gutsd` so that they can be run with superuser privileges.

Due to the self-contained nature of the service folder, GUTS services are modular and pluggable (though, if desired, they can have interdependencies where one GUTS services may use another's actions). When `gutsd` starts up, services are automatically discovered and sent to `gutsweb`. Several of the services that we have implemented are described in Section 5, as examples of functionality that can be plugged into GUTS.

3.2 GUTS D-Bus Daemon

The `gutsd` daemon provides access to GUTS services over D-Bus [5], an open source system for inter-process communication. The daemon runs with superuser permissions and registers itself on a known D-Bus interface. On startup it automatically detects services and loads their registered functions. It then presents these functions and overall GUTS introspection information over the D-Bus interface.

Security for `gutsd` rests on the use of a context object and the sender's username as provided by the D-Bus function `dbus_bus_get_unix_user()`. The context object is used to provide additional information to functions and parameter checking functions, such as the originator of the request (website or command line) and the identity of the user performing the request. On request submission, `gutsd` receives the context alongside the request parameters (both JSON-encoded) and uses it for validation. While `gutsd` gives the GUTS website's Unix user account trusted access and allows it to perform functions on behalf of any user, other users accessing GUTS from the command line are only allowed to make requests on behalf of their own user account, which `gutsd` enforces by checking the sender information from D-Bus.

While we have chosen D-Bus as our inter-process communication framework because of its wide deployment and security features, there is nothing integral to

GUTS that uniquely depends on D-Bus. Any other communications framework with the ability to securely identify message senders could be easily used instead. A framework without that ability could also be used by first adding an authentication layer on top of the framework. In our environment the D-Bus communication happens entirely internally to a single machine, but if messages between GUTS clients and the GUTS server are sent in an untrusted network environment, an additional layer of encryption (such as SSL) should be used to protect against snooping on function parameters (which can contain sensitive material such as passwords) as well as man-in-the-middle attacks.

3.3 GUTS User Interface

The main user interface to GUTS is the `gutsweb` frontend, a simple website with a design influenced by the modularity of the underlying service framework. Each GUTS service is allotted a tile on the dashboard (Figure 1) that describes the service and links the user to its own pages. These subpages can either provide further information or present forms for submitting a request.

For example, the mailing list service tile offers the choice of either viewing one's mailing list memberships or creating a new mailing list, as shown in Figure 4. Clicking the "Show my mailing lists" button leads to an overview of the user's mailing lists subscriptions, which including an indication of any administrative privileges for those lists. List administrators can simply click on the respective list to be taken to a new page where they can add or remove list members. Creating a new mailing list is just as easy; the corresponding button takes the user to a form for entering basic information about the new list and submitting the request for approval.

The allotment of one tile per service allows users to easily survey all the available tasks, while preserving the bundling of related tasks into clearly distinguishable services. It also presents a straightforward way to add the frontend hook for a new service; the developer can simply design a new tile and insert it into the dashboard, requiring minimum effort in working around the frontends for existing services.

3.3.1 Administrator Interface

When a user submits a request for an action that requires administrator approval, all staff members receive an email that contains the parameters of the requested action, a link to the new job on the administrative web interface, and a notification of the current status of the job. This is implemented by embedding in the email an HTML image tag pointing to a URL on the GUTS server, which will read "Executed", "Waiting for Admin", "Rejected" or similar, depending on the current job status. Because the image is downloaded when the message is viewed, administrators checking their email immediately see an up-to-date notification of the status. This notification prevents unnecessary trips to the web interface to approve already-executed jobs (a frustratingly common occurrence when SCCS used Los). Figure 5 shows a snapshot of the email, including the embedded image.

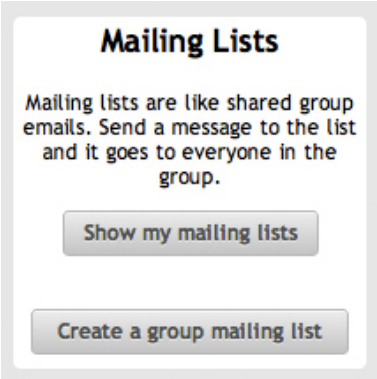


Figure 4: The mailing list tile. This tile is provided by the `mailinglists` service and provides access to commonly used tasks.

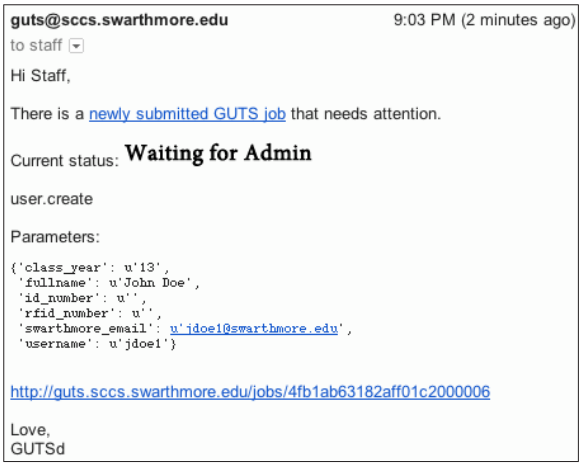


Figure 5: An example GUTS job notification email. Similar emails are sent out for all delayed actions so that administrators can approve them. The parameters dictionary has already been verified at this point, but it will be verified again once the action is approved.

Following the link in the email takes an administrator to the status page for the selected job. This page presents the history of the job and allows the administrator to approve or reject the job, or edit the parameters of the job before executing it. Each job is also accessible via the

"Pending Jobs" link in the "Staff Requests" dashboard tile, which leads to a page that can show not only pending requests but also a log of all previously executed jobs.

4 Security Model

From a security standpoint, it is important to ensure that user input is verified before it is used to make automated system-administration decisions. This provides protection against security vulnerabilities and prevents actions from running which would leave the server in an inconsistent state. This level of our security model is implemented with a layer of *parameter checker functions* that serve as a boundary between the interface-level code and the server-side functions. The code as implemented enforces our security guarantees while providing a convenient code interface for service programmers. It also serves as a common repository of validators for various data types, such as Unix users and mailing list names.

The input verification is all done on the server side. Similar to the Linux kernel, which provides a system call interface for userland to call into the kernel, the privileged code (in Linux's case, the kernel; in our case, the checker functions) needs to carefully check the validity of the parameters before doing anything. `gutsd` receives job requests over the Desktop-Bus (D-Bus) interface [5], and does not trust that input.

There are two kinds of verification which must be performed. The first is traditional input verification for web applications, in order to verify the input is correctly formed and to prevent database injection attacks. These are handled by individual parameter verifiers, which might check the format of a Unix username or verify that a flag coming from a dropdown list actually belongs to the set of legal options. After this basic verification, there are function-level checkers, which can reason about the relationships between parameters (for example, "Does this user have administrative permissions on this Git repository?") with full confidence that the parameters are valid for their individual data types.

To guarantee security, `gutsd` should check its inputs whenever they come from untrusted sources, and whenever the underlying system state might have changed. To this end, the verification typically happens multiple times during the lifetime of a call into GUTS. There are four ways to call into `gutsd`: (1) When an info function is invoked, the parameters are checked before the function is executed; these functions do not change system or database state. (2) When an action is invoked, the parameters are checked first, then the action is tried. If successful, the result is logged in the database, but if not a parameter error is returned to the caller (usually `gutsweb`). (3) A delayed action's parameters are checked, and if successful the server enters the request into the database and emails the staff. (4) Meta functions are like info functions, except that they are allowed to modify system state indirectly (e.g. by calling an action). Through these four interfaces, invalid data never enters the database, and data is verified both when originally submitted and immediately before being used.

Figure 6 presents a code sample from the `git` service (with some changes for brevity). This code demonstrates several parameter checkers and the beginning of a function which implements a GUTS action. This particular action can perform several functions related to the management of a Git repository (changing web visibility, descriptions, and user access). One of the parameters it must verify is a Unix user/group, so the action uses the `user-or-group` data type provided by `libguts` (logic within the function differentiates users and groups). The `permissions` and `visibility` arguments are verified to belong to a limited set of possibilities, while `name` verifies that a repository with the given name exists. Then, the action checker, `check_user_is_admin`, verifies that the user attempting to execute this action has the necessary privileges. It would be easy to create a `check_update_settings` function that called `check_user_is_admin` as well as performing any other required checks, if more checks were necessary.

The actual declaration of a GUTS action with specification of the verifiers is implemented using the Python function declaration syntax. A decorator provided by `libguts` interprets special default arguments of type `Param` to build up type information for each action parameter. This ties into one of our major design goals, which is to make implementing an action as simple as writing a Python function. The function `update_settings` in the code sample appears to be declaring a Python function with default values for its arguments. However, the decorators actually result in the creation of an object that automatically decodes its arguments from JSON, verifies them for correctness, and encodes return values, raising exceptions that GUTS understands if anything goes wrong along the way. All of this complexity is hidden from the service programmer.

Security Implications of `gutsweb`

Because `gutsweb` is a trusted client, a compromised `gutsweb` process could perform tasks on behalf of any user. However, the attack surface and potential damage are limited by the defined functions, and a compromised `gutsweb` process does not have root access. The frontend does not contain logic about user privileges; this is instead described and enforced in the validators for the functions themselves, preventing duplication of information and simplifying the implementation of `gutsweb`.

```
def check_user_is_admin(context, name):
    username = context.user.username
    repo = RepositoryGroup(settings.REPOSITORY_BASE_PATH).repository(name)
    if not (repo.user_is_admin(username) or
            any(repo.group_is_admin(g.groupname) for g in User(username).groups)):
        context.add_error("{0} is not an administrator of {1}".format(username, name))

def repo_name_check(context, name, params):
    repo = RepositoryGroup(settings.REPOSITORY_BASE_PATH).repository(name)
    if not repo.exists:
        context.add_error("Repository {0} doesn't exist".format(repo.name))

_permissions  = (None, "none", "read", "write", "admin")
_visibilities = (None, "public", "private")

@guts.action(service="git", checker=check_user_is_admin)
def update_settings(context,
                    name        = Param(checker=repo_name_check, required=True),
                    description = Param(                         default=None),
                    visibility  = Param(choices=_visibilities,   required=True),
                    entity_name = Param("user-or-group",         default=None),
                    permissions = Param(choices=_permissions,    default=None)):
   pass # GUTS action logic goes here
```

Figure 6: A code sample illustrating parameter checking for the `git.update_settings` action. A single GUTS action is defined which uses the GUTS Param domain specific language to define the function's parameters, complete with verification functions and defaults.

5 Case Studies for Implemented Services

User account creation User accounts were a core consideration when designing GUTS, because our accounts are non-trivial compared to those of many other sites. This consideration helped shape a number of GUTS features such as delayed actions, automatic parameter checkers and the split between `gutsd` and `gutsweb`. From the start we knew we needed GUTS to support delayed account creation, where the user signs up for an account and later an administrator approves it. Once the account is approved, GUTS needs to then create an entry in our LDAP system, copy skeleton files, and set up webspace, shell access and an email inbox. Doing all of this requires integration with a number of different tools (including LDAP, Postfix, shell commands, and Mailman) and therefore benefits hugely from the flexibility that GUTS functions allow. Writing functions in plain Python allows us to use pre-existing Python libraries as well as to call out to specific shell executables. The integrated delayed-action support ensures that checks are made both when the action is submitted and when it is finally run, in order to catch errors as early as possible, while also preventing errors that arise if the system configuration changes between submission and execution.

Git repository management In an effort to provide useful services to our users, SCCS previously created a set of tools for managing shared and private Git [10] repositories that are hosted on our server infrastructure. We make extensive use of Filesystem Access Control Lists (FACLs) to allow our users to create and use repositories, allowing private or shared read, write, or admin access as they wish. Users add, modify, and view repositories via our production GUTS website. This is implemented using a GUTS service which nicely integrates with our existing Git infrastructure which has a Python library interface. Info functions provide insight on existing repositories and action functions allow for changes to be made in a simple (from the service programmer's point of view) way.

Tripwire One of the tools we use to monitor the security of our servers is Open Source Tripwire [6], a file integrity scanner and reporting system. Tripwire is run daily and supplies the administrator with an overview of changed files in web directories and other high-risk areas. Originally the only way to approve or reject these

changes was through the shell. By writing a Python module that can parse and modify Tripwire's report files, however, we have now created a convenient web interface for approving these reports that integrates with the other approval systems we use every day.

Mailman One of the most popular services that we offer are GNU Mailman mailing lists [7], which are frequently used by many of the clubs and organizations at Swarthmore College. Since users are often not very comfortable or even aware of the Mailman administration interface, we decided to integrate that functionality into GUTS. Mailman provides a convenient Python interface to its list management systems, so this is fairly easy to accomplish. Now, users can comfortably create and administrate mailing lists directly from their dashboard, eliminating the need to leave the GUTS interface for common mailing list maintenance operations. For more advanced administrative tasks, the GUTS web interface provides a link to log directly into the Mailman web interface without having to use a Mailman administrative password: GUTS sets the login cookie used by Mailman before forwarding to the administrative page. We have also recently added to the Mailman interface an admin-side action which scans the mailing lists for invalid `@swarthmore.edu` email addresses, a constant problem for SCCS lists since graduating students rarely unsubscribe from their mailing lists before their email addresses expire. Although this could have been implemented as a standalone script, implementing it in GUTS meant that we already had convenient helpers in place to access our mailing lists and to check with the College's LDAP and SMTP servers for address validity.

6 Results and Reception

We made the decision to deploy GUTS in production and officially retire Los, the former task manager, after it became apparent that Los was not going to be able to keep up with the additional services that SCCS wanted to provide to its user base. The pivotal moment came when we wanted to automate the process of creating Git repositories for a user. This functionality would have been very hard to implement in Los, and thus it was decided to roll out GUTS somewhat earlier than planned. On October 10th, 2011, GUTS went live.

The results of the migration to GUTS were positive in all areas. Since GUTS made it possible to approve delayed tasks via the web dashboard, administrators could now tend to requests from any kind of web-enabled device, including smartphones. As a consequence, the median approval response time for delayed tasks dropped by a factor of 20 to 14.3 minutes for GUTS (mean 5.2 hours; based on 606 jobs between November 2011 and May 2012) as opposed to 4 hours and 54 minutes for Los (mean 59 hours; based on 1177 jobs between April 2004 and September 2011).

We also more than tripled the number of new user accounts per semester from the 2010-2011 academic year (using Los) to the 2011-2012 academic year (using GUTS). Although we have not attempted to measure the exact effect of GUTS, we believe a substantial part of this increase is from a higher conversion rate of students considering creating accounts, resulting from a greatly simplified signup process (1 page with 4 fields for GUTS; several pages for Los). We presently reach around one third of the incoming freshman class (our primary source of new users) every year, and expect this proportion to increase now that we can build on the infrastructure provided by GUTS to easily expand the services we offer and provide greater incentives for account creation.

But the greatest benefit of GUTS is that it acts as a unifying, straightforward platform for both SCCS and its user base: regular users are able to access most of the features an SCCS account provides with a single click from their GUTS dashboard, student group leaders can easily tend to their mailing lists and websites from the group dashboard, and administrators can quickly view, approve, correct, or reject tasks from the staff dashboard. Administrators can also access staff-specific areas via GUTS, such as the task for approving Tripwire reports, as well as built-in documentation on GUTS and a list of registered GUTS functions generated via introspection. This integrated approach has made life significantly easier for both the user and administrator, allowing the user to readily discover and take advantage of the services that SCCS offers, while lowering the hurdles for the administrator to develop, test, and roll out improvements.

7 Lessons Learned

As with all major projects, GUTS was a learning experience for those involved. This section offers a small window into our trials and tribulations in developing GUTS.

7.1 Mistaken Assumptions

There were several assumptions that we made during the development of GUTS that did not pan out once the service was launched. Some of these can be deduced from the types of email that we received right after launch. It became clear that the web interface needed to expose a lot of the functionality much more obviously. We still get many emails requesting password resets, even though this functionality has long been present in GUTS. We are working on improving the home page so that such simple functions are immediately apparent.

Also, as GUTS expands in scope, some new services that we want to integrate do not quite fit into the GUTS model. A case in point is the Tripwire module, which does quite a bit of processing on the server side in order to munge the report into a readable format and present it for staff approval in the web interface. If the report gets too long, this processing cannot complete before Apache decides that the process must have crashed and kills it. An expansion to our job model will allow us to mark certain jobs as asynchronous, so that they cannot lock up the main `gutsd` process.

7.2 Pipe Dreams

Of course, in the absence of time and resource constraints, there are certain aspects of the GUTS architecture that might be implemented differently. Many of these wishes have actually been resolved, as the internals of GUTS have been rather significantly overhauled since the system went into production. Making this work without disrupting the experience for our users requires discipline in testing all new changes before pushing them to the live system – as well as good error reporting (all staff members receive an email with a full stack trace whenever a user encounters a HTTP 500 server error). An automated test suite for GUTS would contribute greatly to reducing regressions, but would also require significant effort in mocking system interaction, due to the highly integrated nature of GUTS services.

Another facet of the system that might be overhauled if we had the expertise is authentication. We are quite satisfied with the security model that we have implemented, and all connections are secured with SSL so man-in-the-middle attacks are not a concern, but the only defense currently in place against impersonation is the Django authentication system.

7.3 Development Challenges

We had our fair share of hurdles to be overcome during the development process. The first of these was in design. The major reason for not writing GUTS earlier was the lack of a good security model. The core `gutsd` process has to run as root, in order to execute tasks (and even for some harmless activities, such as querying Mailman), but we certainly can't have the entire web server running as root. This common-sense security requirement is directly at odds with the purpose of GUTS, which is to allow users to execute tasks with sysadmin intervention only when necessary. In the past, Los solved this problem by restricting the user-facing parts. The web interface could only submit tasks, and the backend relied on the security of GPG-signed email for communication. For GUTS, we wanted tasks to be nearly full-fledged web apps, written in pure Python, which required devising a method for securely separating the user and server layers. The solution, discussed earlier in Figure 2, was to strictly partition privileged code behind a D-Bus interface. With D-Bus we can specify policies so that only the GUTS web interface's user can connect, for example, and only explicitly exported functions are accessible. Getting this to work for our site also involved setting up Apache MPM-ITK to run the GUTS web interface as a special user; since we allow users to write their own CGI scripts, this is how `gutsd` ensures it is talking to `gutsweb` and not another rogue Apache process.

Another significant driver of design decisions was ease of service writing. Ideally, a programmer familiar with Python and Django should be able to write a service without knowing about the internals of GUTS. Each service folder, as discussed previously, looks like a Django app. Of course, there is a fair amount of magic required to keep up this appearance. The `gutsweb` interface gathers parts from all of the services to make one Django app. On the other side, `gutsd` exports various functions from each service, based on decorators. We also take advantage of Python's default argument syntax to implement the parameter checking layer. Preventing these abstractions from breaking down was a major challenge in our development process.

Related to leaking abstractions, a design concern which came into play at the end of the process was the realization that GUTS could in principle be used at sites other than SCCS, and so there should be a clear separation between "core" functionality and anything SCCS-specific or essentially optional. Defining and enforcing this boundary was difficult and required reorganization of some of the GUTS internals.

Working through these challenges in the development process taught us a lot about the internals of Python, Django, and how to integrate these layers into a working system. The result, while the product of a somewhat non-linear development process, is a successful system that satisfies all of our key design goals.

8 Conclusion and Future Work

GUTS provides an integrated environment for user and administrator interaction with SCCS resources. Though its original goal was simply to speed up handling of administrative tasks so that a small group of volunteer system administrators could effectively serve the entire college community, GUTS is extensible enough that it has become the default platform SCCS uses for providing new user services of any kind. Several features of GUTS contribute to its applicability and ease of use.

First and foremost, GUTS includes a seamlessly integrated privileged execution model. Each service is sim-

ply a set of Python functions that the system calls as needed. The system also guarantees that the inputs to those functions are correctly formatted, and marshals interprocess communication over D-Bus between the unprivileged user interface and the privileged GUTS core. This makes it very easy to add new privileged services and design user-friendly interfaces to interact with them.

Secondly, GUTS streamlines the interaction between user and administrator, using an optionally-delayed execution model. Tasks that require administrator approval are performed via the web interface just like other tasks, but executed only after approval is received. Furthermore, the users and the administrators interact with the system through the same web interface. The GUTS job model is sufficiently general that administrative tasks such as approving delayed jobs are simply services that GUTS provides to the staff members. GUTS can be used as an interface to its own internals.

Lastly, our security model for services is designed around permissions that are afforded on a user or group basis. Parameter checkers can easily be implemented that enforce privilege separation for individual tasks. This means that the privileges required for an action are fully dictated by the service that implements it; there is no requirement that all services share a privilege model.

Although it is successfully serving many of our needs, GUTS is still an evolving system and has much room for improvement. The most immediate path for improvement is to add new services and extend those already supported. For example, we are currently working on implementing a service that allows users to create and manage MySQL databases, and one that lets users easily update their profiles on the SCCS-run campus photo directory.

To support more types of services, we would like to make it safe to create long-running jobs, which currently interfere with other jobs due to the single-threaded nature of `gutsd`. This can be solved by providing a facility for asynchronous jobs, which would return a response to D-Bus immediately, and then perform their work in the background. Managing these jobs should be relatively easy, since GUTS already has the ability to check on the status of jobs in the system.

A further way to improve the usefulness of GUTS will be to increase the flexibility of the user interfaces to the system. Some services, such as the `git` module, are generally used alongside command-line tools, and would therefore be convenient to access through dedicated command-line tools themselves — so we can avoid a trip to the web interface just to create a Git repository. We plan to implement a comprehensive command-line GUTS interface to fill this gap. Also, since many of the SCCS administrators approve jobs on their smartphones, a mobile-specific version of the web frontend would make that process even easier.

9 Acknowledgements

This work relies upon the invaluable contributions of several free software projects, most significantly Python, Django [4], and D-Bus [5]. SCCS is supported through equipment funding and space provided by the Student Council of Swarthmore College. We also receive network resources from the College's Information Technology Services Department.

10 Availability

GUTS is free software (released under the GNU General Public License), available from the project homepage at `http://sccs.swarthmore.edu/projects/guts/`.

References

[1] BEST PRACTICAL SOLUTIONS LLC. RT: Request tracker. http://bestpractical.com/rt/.

[2] BURGESS, M. A tiny overview of Cfengine: Convergent maintenance agent. In *Proceedings of the 1st international Workshop on Multi-Agent and Robotic Systems* (2005).

[3] CAMERON, J. Webmin: a web-based system administration tool for UNIX. In *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC)* (2000).

[4] DJANGO SOFTWARE FOUNDATION. Django: A web framework for the Python programming language. http://djangoproject.com.

[5] FREEDESKTOP.ORG. Software/dbus. http://dbus.freedesktop.org.

[6] KIM, G. H., AND SPAFFORD, E. H. The design and implementation of Tripwire: a file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security* (New York, NY, USA, 1994), CCS, ACM, pp. 18–29.

[7] MANHEIMER, K., WARSAW, B., AND VIEGA, J. Mailman - an extensible mailing list manager using Python. In *Proceedings of the 7th International Python Conference* (1998).

[8] RHETT, J. Request v3: A modular, extensible task tracking tool. In *Proceedings of the Twelfth USENIX Large Installation System Administration Conference (LISA)* (1998).

[9] STEPLETON, T. Work-augmented laziness with the Los task request system. In *Proceedings of the Sixteenth USENIX Large Installation System Administration Conference (LISA)* (2002).

[10] TORVALDS, L., AND OTHERS. Git - fast version control system. http://git-scm.com, 2006.

[11] WALBERG, S. Automate system administration tasks with Puppet. *Linux J. 2008*, 176 (Dec. 2008).

[12] WEBMIN. What is Usermin? http://webmin.com/usermin.html.

Bayllocator: A proactive system to predict server utilization and dynamically allocate memory resources using Bayesian networks and ballooning

Evangelos Tasoulas
University of Oslo
Department of Informatics
evanget@ifi.uio.no

Hårek Haugerund
Oslo And Akershus University College
Department of Computer Science
harek.haugerud@hioa.no

Kyrre Begnum
Norske Systemarkitekter AS
kyrre.begnum@nsa.no

Abstract

With the advent of virtualization and cloud computing, virtualized systems can be found from small companies to service providers and big data centers. All of them use this technology because of the many benefits it has to offer, such as a greener ICT, cost reduction, improved profitability, uptime, flexibility in management, maintenance, disaster recovery, provisioning and more. The main reason for all of these benefits is server consolidation which can be even further improved through dynamic resource allocation techniques. Out of the resources to be allocated, memory is one of the most difficult and requires proper planning, good predictions and proactivity. Many attempts have been made to approach this problem, but most of them are using traditional statistical mathematical methods. In this paper, the application of discrete Bayesian networks is evaluated, to offer probabilistic predictions on system utilization with focus on memory. The tool Bayllocator is built to provide proactive dynamic memory allocation based on the Bayesian predictions, for a set of virtual machines running in a single hypervisor. The results show that Bayesian networks are capable of providing good predictions for system load with proper tuning, and increase performance and consolidation of a single hypervisor. The modularity of the tool gives a great freedom for experimentation and even results to deal with the reactivity of the system can be provided. A survey of the current state-of-the-art in dynamic memory allocation for virtual machines is included in order to provide an overview.

1 Introduction

Computers and ICT (Information and communications technology) undoubtedly is emerging more and more into our lives. A Statistical research (2007) provided that we would have 1 billion personal computers by 2008 [18]. This was confirmed by another research in June, 2008 [5] and both of them foresee that this number will be doubled by 2014-2015.

The adoption in the Enterprise increases as well as ICT increases corporate productivity [1], but the companies obviously need to use non stop running servers. This leads to increased hardware purchase costs, as well as peripheral costs and environmental damage coming from the power consumption of the servers and the cooling facilities [9]. If one considers that most of the time (>70%) in typical deployments the servers are idle, the cost to operate underutilized servers is significant [9]. Furthermore, if a server requires more computing power at peak load and not for more than a few hours every day, either better hardware has to be acquired, or more servers in a load balanced topology have to be added. As an outcome, even more power consumption and idle time is added to the infrastructure. Except that, human presence and intervention is needed to take care of the hardware changes.

A solution to the described problem can be given using server consolidation and Virtualization technology. Since the virtual machines have no physical boundaries, they maintain great flexibility to manipulation of their consumable resources and extend the options to provide high Quality of Service (QoS) using scripts or other soft technologies. Probably the most famous technique to address performance issues and underutilization, is the live migration between different hypervisors [2, 12, 4], but it needs further infrastructure like the existence of a common storage device (SAN/NAS) and more than one physical hosts - hypervisors [4]. Small businesses (SMBs) might not even have this additional infrastructure or spare servers to use live migration. An alternative solution could be to use the dynamic resource allocation ability of virtualization to dynamically change the resources allocated to the running virtual machines on a single hypervisor

at the right time in order to achieve optimal performance.

Dynamic CPU and live memory scaling is provided by mainstream virtualization technologies like KVM, XEN and VMware [8, 15]. CPU is a time-shared resource so it can be easily shared among the virtual machines and priority can be given by the scheduler to the most important virtual machines [8]. Memory, on the other hand, is harder to share since a memory page reserved by one virtual machine, cannot be released as long as it is in use. Common tactics used for memory sharing and scaling are achieved by memory overcommitment. Memory overcommitment builds upon the assumption that all virtual machines residing in the same hypervisor do not need to consume 100% of their available memory at all time. Following this assumption, more memory than the total memory available in the hypervisor is assigned to the total number of virtual machines and some memory addresses are used by more than one guest, usually not at the same time.

Ballooning is a common memory overcommitment method which is used in this paper. Ballooning needs a driver to be installed in the guest operating system. The driver communicates directly with the hypervisor and acts on demand. When the hypervisor needs more memory for a guest, it will choose a virtual machine which is the "victim" or the "donor" and the balloon driver will "inflate" in this virtual machine, trying to allocate memory pages from it and give them back to the hypervisor. Then the hypervisor will "deflate" the balloon of the guest who is the "beneficiary" and needs to get the claimed memory, making available these memory addresses to it (see figure 1). One disadvantage of ballooning, is that even if the virtual machine is trying to collaborate, the balloon driver might not be able to recover the memory requested by the host fast enough to avoid performance degradation [8, 13]. To avoid or reduce this undesirable behavior, proactivity is needed. A tool which is able to learn from repeatable events, predict virtual machine memory load beforehand and act, would help to solve the disadvantage of real time ballooning to a large extent.

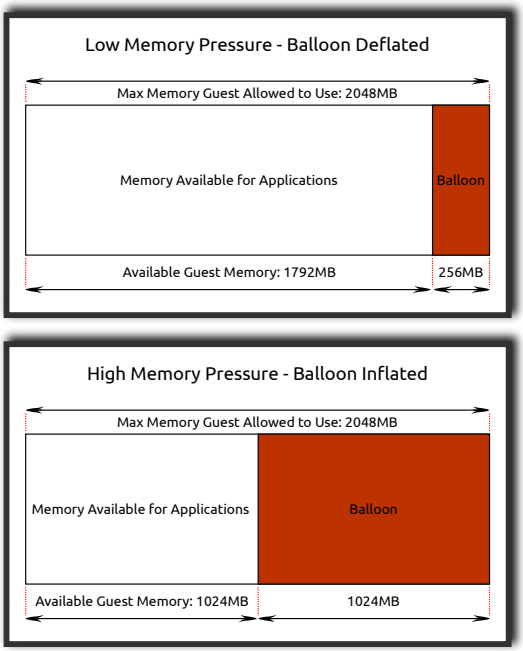


Figure 1: How ballooning works

The scope of this paper, is an attempt to create a prototype self learning, predictive system for dynamic memory allocation in virtual machines running in a single KVM hypervisor using Bayesian networks. A Bayesian belief network is a probabilistic graphical directed acyclic model, indicating the conditional dependence in a set of random variables [3]. Bayesian networks are popular to probabilistic artificially intelligent decision support systems for their good ability to learn from new observations, perceive and plan ahead [7, 10, 6]. A Bayesian network first needs to be trained by a person who is knowledgeable in the environment it will have to work with. Nodes of random variables (chosen by the trainer) have to be created and their directed relationship influence needs to be defined. The Bayesian network will then provide probabilities for the event represented by a node to happen, given the probabilities of the surrounding nodes. More nodes for more accurate results can be added to the Bayesian network if there is evidence of interference from other sources, and the current ones can adapt to behavioral changes of the system giving accurate results as more knowledge is absorbed. This means the longer the system works, more confident results it will provide.

The layout of this paper is as follows: In the related work section an overview of similar scoped papers and their difference from this approach are explained. The methodology section comes to give an explanation of the methods used to perform the experiments, which

is followed by the results, discussion, future work and a conclusion.

2 Related Work

Research in the area of virtual machine dynamic resource allocation is highly active, as this is the key to achieve proper resource utilization, environment friendliness, cost reduction and increased profits by physical resource oversubscription. There are many papers found addressing the same topic, but a brief survey with the most closely related work with similar scope as this paper is gathered in this section.

2.1 Adaptive Control of Virtualized Resources

Adaptive Control of Virtualized Resources in Utility Computing Environments [11]

This system is a quite simple approach based on control theory, for dynamically allocating virtualization resources in a single hypervisor to succeed in CPU QoS. The experiment comprises input data from the test virtual machines. An actuator in the control system controls the CPU scheduler of the hypervisor, to assign resources to the virtual machines so that they will not exceed 100% utilization and performance decrement. They only focus on CPU utilization.

2.2 MEmory Balancer (MEB)

Dynamic Memory Balancing for Virtual Machines [19]

MEmory Balancer (MEB) is a software which dynamically monitors the memory usage of virtual machines, it will then calculate its memory needs, and periodically reallocates memory to the virtual machines.

MEB has an estimator and a balancer. The estimator will build a Least Recently Used (LRU) memory pages histogram for each virtual machine and it will also monitor their swap space usage. Then the balancer runs with an interval and adjusts virtual machine memory, based on the information given by the estimator and the available host resources. Ballooning is used for the memory management from MEB.

2.3 Memory Buddies

Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers [17]

Memory Buddies, is a project which will try to maximize the efficiency of the page or memory sharing feature available in hypervisors, between multiple hypervisors. The page sharing feature will only merge memory pages of virtual machines running in the same hypervisor, so in a data center with multiple hypervisors, memory sharing opportunities may be lost because the guest machines holding identical pages are located on different hosts.

The contribution of this paper is a memory fingerprinting technique to identify guests with high memory sharing potential. Then it will use live migration to move these hosts and shutdown or start on demand hypervisors not in use to trim down operational costs by reducing the energy consumption. Their evaluation shows a 17% increase of the virtual machines running in a data center.

2.4 Overdriver

Overdriver: Handling Memory Overload in an Oversubscribed Cloud [16]

Ultimately, the target of this project is to maximize competitiveness and profitability of cloud providers by oversubscribing customers. Since most of the physical resources are leased using virtual machines, oversubscription means that the cloud provider sells more subscriptions, or more virtual machines to their customers than what its infrastructure can actually handle if all of them would run at peak load at the same time. The coherence is that the peak load for each customer is largely transient (up to 88.1% of overloads last for less than 2 minutes) and not at the same time for all of them. Overdriver focuses on memory oversubscription as memory is not largely oversubscribed in practice like CPU, because memory overload leads to swapping and consequently to severely degraded performance.

Existing methods to handle memory overload use mostly migration to another physical machine that can handle the memory requirements, but virtual machine migration is a heavyweight process needs also high network usage, best suited to handle sustained or predictable overloads. They propose a new application of network memory to manage overload giving it the name “cooperative swap”. This will take swap pages and store them to memory servers over the network. Then they present Overdriver, the system that it will respectively choose between VM migration and cooperative swap to manage either sustained or transient overloads. Overload will create a probability profile for each virtual machine to decide after how much time the coming load is considered to be sustained load for the specific guest. When the increased memory load comes, it will first

use its cooperative swap feature and if the load surpass the sustained threshold set for this guest (based on the probability profile), a live migration will be executed. Overdriver is a reactive tool to avoid excessive memory load.

2.5 Managing SLA Violations

Dynamic Placement of Virtual Machines for Managing SLA Violations [2]

In this paper, the authors introduce a management algorithm for dynamic resource allocation in virtualized environments using live migration. Their elemental goal is to meet the Service Level Agreement (SLA) the company has with its customers, while reducing the operational costs from the data centers of the company. The algorithm in this work will measure logged data for each of the virtual machines and forecast the future demand. It will then remap guest virtual machines to different hosts. Their forecasting technique involves some statistical analysis to determine the type of resource usage of the guests and then classifies them to three categories:

1. Guests without variability or periodic behavior. (No need to migrate)
2. Guests with slight variability and periodic behavior. (Potentially good guests for migration)
3. Guests with strong variability and periodic behavior. (Highest migration potential)

From those three categories the second and the third category of virtual machines are potential candidates for live migration. The first one shows sustainable resource usage so it does not need to change host often.

The next step is the calculation of available physical resources needed to handle the predicted load and start or shutdown non needed servers to reduce costs. This method is a proactive probabilistic method like the one proposed in this paper but they use different statistical tools, while their experimental studies are only focused CPU utilization which is known that it is much easier to handle since it is a time shared resource.

2.6 VMCTune

VMCTune: A Load Balancing Scheme for Virtual Machine Cluster Based on Dynamic Resource Allocation [20]

This paper proposes VMCTune, a tool that monitors the resource utilization of virtual machines and their hosts. Then it uses dynamic resource allocation for virtual machines running on same hypervisor to achieve better local resource utilization and if the QoS cannot be met, it uses live migration for virtual machines among different hypervisors to achieve global load balancing. The tool focuses in Paravirtualized machines and offers a reactive solution dealing with CPU, memory and network bandwidth. Paravirtualized machines are easier to handle their resources in comparison with the Fully virtualized ones, as they are aware that they run in a virtualized environment.

VMCTune has several tools to monitor, log the data, schedule the resource allocation or issue a live migration if needed and a command line tool by which the user can monitor the status of the virtual machines and control the host.

Improving of scheduling algorithm of the live migration is the key work mentioned as their future research work in this paper, as the one used is a very simple best-fit algorithm. For example, if a virtual machine needs more resources and it cannot fit to the current hypervisor, it will be transferred in a different one with the available resources. There will be no attempt to squeeze resources from other hosts to achieve the best possible utilization.

3 Methodology

In the next section the design of Bayllocator will be presented.

3.1 System Design

In order to improve working efficiency and extendability, the system is designed in a modular fashion. As illustrated in the conceptual system design in figure 2, everything runs on top of a single physical machine which is the hypervisor. The hypervisor is split into two logical partitions. First is the space available for the virtual machines (left part on the figure) and the latter (right part on the figure) is the space available for the administration of the virtual machines.

Data generation is taking place on the virtual machines. Data collection resides on the administrative part of the system together with the prediction system. A script collects the data through the virtualization layer (the dotted separation line) and stores it in a database. The prediction then is made using as the data from the database as input and the dynamic memory allocation mechanism will reallocate the memory using the virtualization layer.

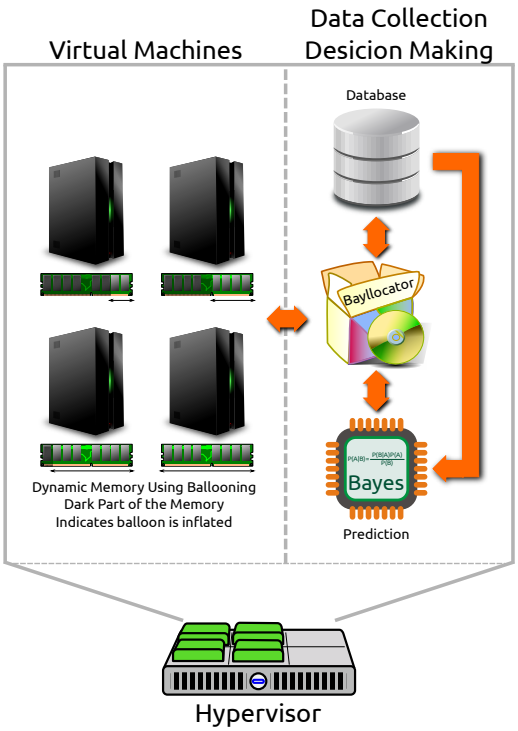


Figure 2: Conceptual system design

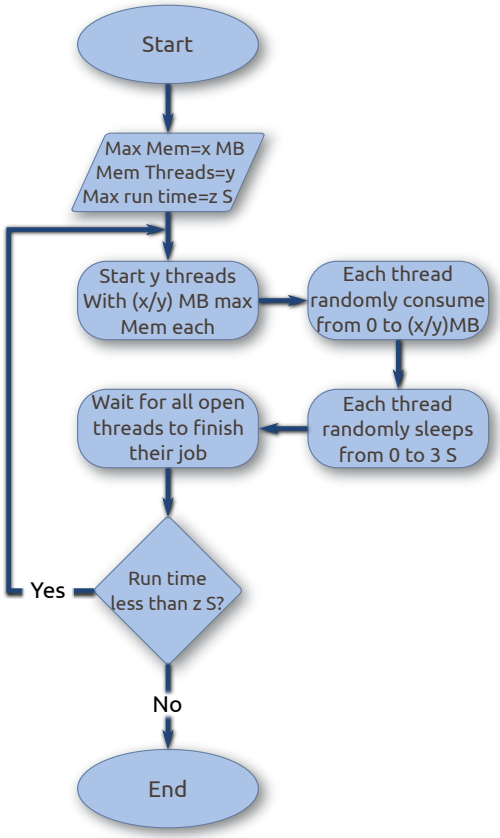


Figure 3: fakeload.pl - Script to generate data

3.2 Data Generation

Controlled random data are generated and mostly used to carry out the experiments. Data generation is a good starting point to explore something new. Generated data has a unique feature. It can be controlled so that one can make prior assumptions and expectations need to be confirmed by the results.

For data generation, a script is created and a flow chart in figure 3 explains its working logic. The design of this script was made with the ambition in mind to create random data with controlled randomness by modifying the variation. The script will generate data given 3 parameters:

1. Max memory to occupy (given in MB). This parameter will set the maximum memory the script will try to claim. The passed memory is a rough estimate.

2. Max threads to create. This parameter will start the given number of random consuming memory threads and all of them together will consume a maximum amount of memory given by the first parameter.

3. Max runtime of the script. This will control the time length the script will be generating data.

After the data are collected and stored in a database, rolling averages are created to smooth the randomness even more.

3.3 Data Collection from Real Servers

Real life data from two servers in production (nexus and studssh) have also been collected in parallel with the generated data. Nexus is a webserver, while studssh is an ssh server for students, with many logged in users. A simulated prediction based on this data is run in the end, which gives an indication of how good the designed Bayesian network and its predictions work on real systems.

3.4 Dynamic Memory Allocation

Dynamic memory allocation driven by the Bayesian predictions is the outcome of this paper and short term prediction is the main goal. A memory allocation script is created and it tries to reallocate the memory of the virtual machines 5 minutes in advance before the expected memory demands come.

3.5 Experiment Design

The infrastructure schematic already explained briefly in figure 2. On top of this, a test environment is being built for the experimental part of the project. Some virtual machines are created, and high memory load is generated and collected on each one of them. Simple Bayesian networks are tested and in the end a prototype (Bayllocator) is created to put everything together. With the final system built, a simulation is running against the real data collected. The observed memory utilisation should follow the predictions closely to be considered successful, and since ballooning is implemented any slowdowns introduced by heavy memory utilisation should be trimmed off.

4 Results

4.1 Bayllocator

Bayllocator is the tool created to accomplish the task in this paper. Some clarifications of the principles this program operates are given as bullets below and a very detailed flow chart explains visually the entire operation steps can be studied in figure 11. Bayllocator will:

- Make a prediction of how much memory each of the virtual machines will need and set this amount of memory plus some predefined percentage (default 15% more).
- Ensure that the virtual machines do not get less or more than the minimum and maximum limits set.
- Make sure that the hypervisor will not swap any memory. In the case of high memory pressure, it is preferred that the virtual machines start swapping, instead of the hypervisor which is responsible to handle them all.
- Distribute fairly (according to a percentage of the needed memory of the total predicted for the virtual machines) any excessive hypervisor memory to the virtual machines. If there is more memory than needed available, it can be used for disk caching purposes so it will boost guest performance.
- Claim memory fairly from virtual machines if they need more than the total amount allowed to get to avoid hypervisor swapping. In this case some of the virtual machines might start swapping under pressure.

It is important to note from the list above, that even though the predictions are made on a per-VM basis, Bayllocator acts with all VMs in mind, trying to distribute the memory as fairly as possible based on individual predictions.

4.2 Predictions

The Bayesian prediction is being made with an R script which will be given as command line arguments the states of the known nodes (evidence), the name of the node which the answer is to be provided and the file with the discrete values to learn the CPTs (Conditional Probability Tables) from. The answer is a two-column comma separated output with the first column showing the possible state and the second column the probability (a percentage) to observe this state for the given inputs. In the example on the following code block, the Bayesian network is asked "what is the expected memory when it is Monday, 07:55-07:59 and the currently observed memory of the guest system is between 100MB and 200MB". The answer is 14.8% to get 100MB-200MB, 24.3% to get 200MB-300MB and so on.

```
$ QueryNet.R /etc/bayllocator/TestServer1.dat \
> FMU Monday h07 m55_59 mem_100_200
mem_100_200,0.148
mem_200_300,0.243
mem_300_400,0.257
mem_400_500,0.226
mem_500_600,0.091
mem_600_700,0.026
mem_700_800,0.009
```

4.3 Choosing the Expected Memory

The result from the Bayesian network apparently contains more than one probability (the previous output of the script has more than one row), so a single number has to be chosen as the expected memory. To make a fair decision, all of the given results are used to calculate this number. First the mean value of the discrete states memory will be calculated and then a single number given the probabilities of all of them as illustrated in the following equations.

$$
\begin{aligned}
mem\_100\_200 = (100+200)/2 &= 150 \\
mem\_200\_300 = (200+300)/2 &= 250 \\
mem\_300\_400 = (300+400)/2 &= 350 \\
mem\_400\_500 = (400+500)/2 &= 450 \\
mem\_500\_600 = (500+600)/2 &= 550 \\
mem\_600\_700 = (600+700)/2 &= 650 \\
mem\_700\_800 = (700+800)/2 &= 750
\end{aligned}
$$

$$
\begin{aligned}
p_1 = 150 \cdot 0.148 &= 22.2 \\
p_2 = 250 \cdot 0.243 &= 60.75 \\
p_3 = 350 \cdot 0.257 &= 89.95 \\
p_4 = 450 \cdot 0.226 &= 101.7 \\
p_5 = 550 \cdot 0.091 &= 50.05 \\
p_6 = 650 \cdot 0.026 &= 16.9 \\
p_7 = 750 \cdot 0.009 &= 6.75
\end{aligned}
$$

$$
\sum_{n=1}^{7} p_n = 348.3MB
$$

4.4 Chosen Bayesian Network

Many Bayesian networks with simple nodes related to the memory and date were tested, but the following one affected by current memory and date proved to be more successful (see figure 4) and its results for the generated data are presented in figure 5, while simulations on real data with the same network are taking place in figures 6,7,8. The Bayesian network nodes abbreviations are expanded in table 1 and the quantized discrete states used for training and querying the Bayesian network can be seen in table 2. From these 5 nodes, the W, H, M and CMU are evident nodes (known) for the prediction to be made, and the FMU is the query node which gives the answers.

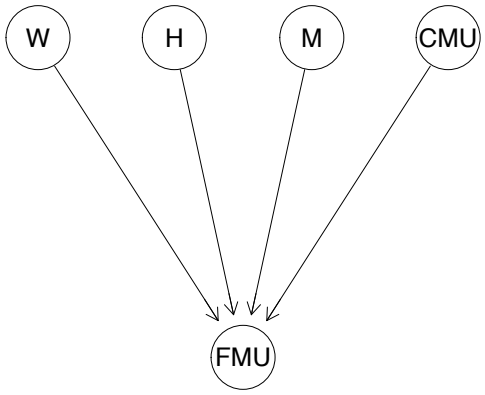


Figure 4: Bayesian Network affected by current memory and date.

Bayesian Network Node Labels Abbreviation Expansion Mapping	
W:	Day of the Week
H:	Hour of the day
M:	Minutes
CMU:	Current Memory in Use
FMU:	Future (Predicted) Memory to Use

Table 1: Abbreviation expansion mapping of Bayesian network node labels.

Discrete States Used	
W	Bool: 0 for weekend, 1 for the rest of the days
H	24 states: 00 - 23
M	12 states: 00_04 - 55_59
CMU	41 states: 100_less, 100_200, 4000_greater
FMU	41 states: Same as CMU

Table 2: Bayesian network node quantized discrete states used for training and querying.

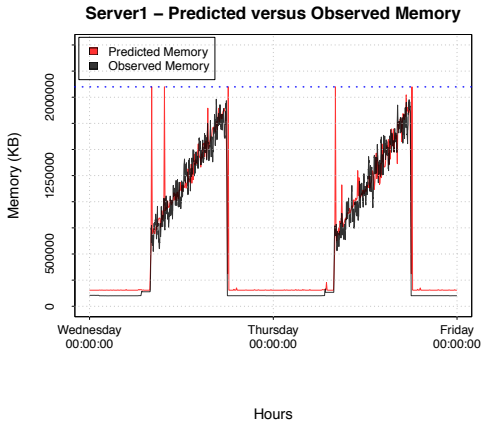


Figure 5: Predictions plotted against the observed data.

The red line in figure 5 shows the predictions which comes 5 minutes earlier and the black line is the actual data observed. There is a large variation in the observed data and this is related to the randomly generated data, but as illustrated the prediction follows quite well.

Some of the predictions often reach a certain wrong value in y axis, marked with a horizontal dotted line in figure 5. This value is exactly 2099200 KB and it is an indication that the system does not know how to answer for the given evidence. When the Bayesian network gets an unobserved combination of evidence, then all of the probabilities of the query node are equal. Because the discrete states defined for the memory nodes CMU and FMU are 41 (from mem_0_100, mem_100_200 to mem_4000_greater every 100MB), the probability for each state is $1/41 = 0.024390244$ which gives this num-

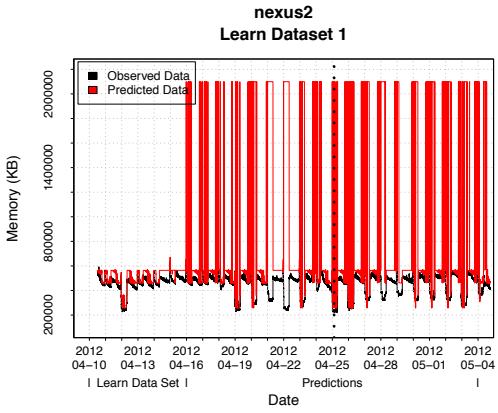


Figure 6: Learning Dataset from April, 10th 2012 to April 16th 2012. Predictions right after the learning dataset till the end of the graph.

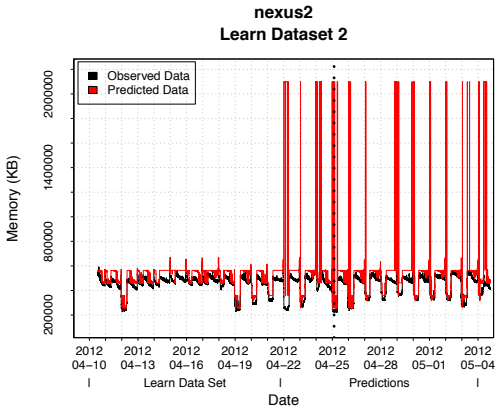


Figure 7: Learning Dataset from April, 10th 2012 to April 22nd 2012. Predictions right after the learning dataset till the end of the graph.

ber as expanded in equation 1.

$$\sum_{n=100,4000} n \cdot \frac{1}{41} + \sum_{n=1}^{39} (n \cdot 100 + 50) \cdot \frac{1}{41} = 2050MB \quad (1)$$

$$2050MB \cdot 1024 = 2099200KB$$

4.5 Prediction Simulation on Real Data

Three simulations ran with different training datasets in **nexus2**. The first learning dataset for nexus2 is from April 10th, to April 16th (figure 6), the second from April 10th, to April 22nd (figure 7), and the third one from April 10th, to April 25th (figure 8). All of the three simulations ran have missed predictions (2099200KB values as explained earlier), but it is obvious that when the system is trained with a larger dataset it fails much less. A comparison of the common predictions for all of the simulations (after the vertical dotted line) shows that for the first training set, 808/2774 predictions failed (29.1%), for the second 122/2774 (4.4%) while for the third only 76/2774 which makes up the 2.7% of the total predictions.

If one takes a better look to the predictions of nexus2, will notice that the predictions follow steps. This is because the training datasets are based on discretized and not continuous variables, so as the predictions. If the memory in the system is 501MB or 600MB, both of these observations are encoded as the memory state mem_500_600 and they represent a memory state of 550MB in the reverse operation.

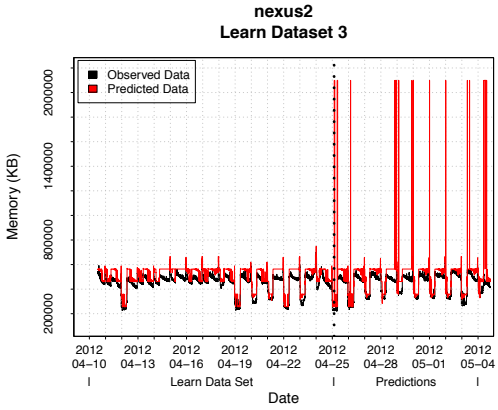


Figure 8: Learning Dataset from April, 10th 2012 to April 25th 2012. Predictions right after the learning dataset till the end of the graph.

4.6 Ballooning Evaluation

A simple evaluation was made to see the effect of dynamic memory allocation. Two operations were running in test server 1 and their execution time was measured. As can be seen in figure 9, the execution time of a PHP script running on an apache 2 web server would take as much as 50 seconds under heavy memory load (between 08:00-18:00), while it would not take more than 3 seconds under normal circumstances. When ballooning is initiated, the operation looked like normal at any time. Same observations can be made for figure 10, where a secure copy operation initiated from an external machine to test server 1. The reason for this slowdown before ballooning, is the heavy memory swapping on hard disk. Hard disk is many orders of magnitude slower than the

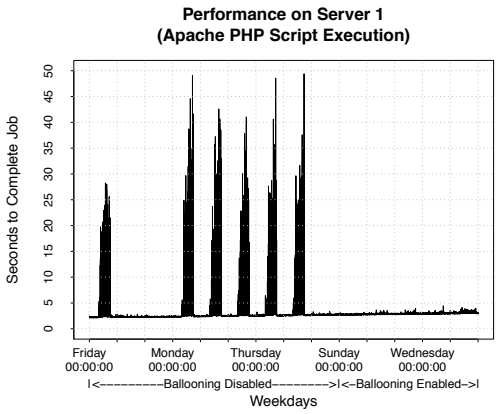


Figure 9: Server Performance

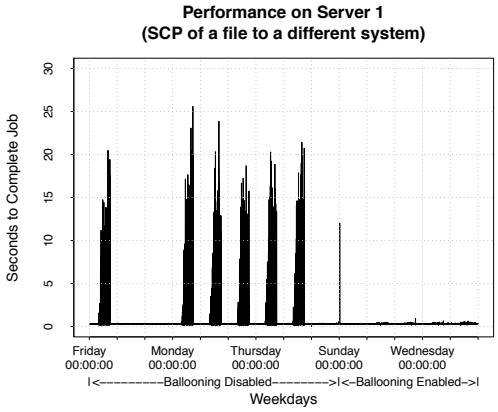


Figure 10: Server Performance

memory. To avoid this kind of slowdowns and use the most out of the physical hypervisors, the dynamic memory allocation is necessary.

5 Discussion and Conclusion

The competence of Bayllocator resides in the prediction method to predict system utilisation in a non traditional way, using Bayesian networks. The focus is mainly in memory prediction and appliance of dynamic memory allocation based on this information to improve sharing efficiency. Due to its nature, memory is not easy to share as the time shared resources like CPU or Disk I/O, becoming an obstacle to server consolidation, green and low cost ICT. Highly active ongoing research tries to predict system utilisation and use this information for dynamic resource allocation, but most of them use traditional statistical methods to approach the problem. The traditional statistical methods might involve rigorous mathematics and they usually are single purpose, or hard to adapt them to different needs. Bayesian networks are strongly associated with artificial intelligence. Moreover, after an extensive literature survey it does not seem that effort to apply the use of Bayesian networks on server utilisation prediction exists. Bayesian networks differ significantly from the traditional statistical methods, and their big advantage is that they are modular directed acyclic graphs, composed of nodes representing variables related to the system they work with, and changing of these nodes might give totally different results and use cases. Furthermore, even the output from traditional methods can be used as the input in a Bayesian network, and more than one of them can be combined. The modular design gives flexibility and allows the easy combination and alteration of different parameters affecting the system.

The prototype software is made in Perl, which is calling an R script for the Bayesian predictions. This modularity gives the flexibility to change the prediction method at any time and evaluate the guest performance under different circumstances. The lack of wide purpose collected data from real servers prevented from further experimentation with the causal nodes affecting the final memory predictions. Each change to the Bayesian network needed time to be tested and make assumptions on how well it works, but when one gets used to work with Bayesian networks, there is a broad spectrum of applications this method can be applied, and certainly computing utilisation prediction and decision making is one of them. The date related variables used in this paper are not the best choice because the date is not the true reason behind high memory activity. They had to be used though, because of the lack of representative collected data. The true reasons are probably the number of logged in users and the number of running processes or network connections which are factors an expertise in the field knows, or is able to suspect. This is why the trainer must be knowledgeable to the subject where the Bayesian networks needs to work with. The date might be an indirect causal node since it affects the number of connected users in a server (for example if it is a weekend less people might work on a server). Not well directed nodes and excessive number of states decrease the learning efficiency of the Bayesian networks, and as a result the training period needs a longer time.

6 Future Work

In this paper, an implementation of simple discrete Bayesian networks and their ability to predict memory utilisation was evaluated. Since the nodes represent discrete states for continuous variables, there is a loss of

information as illustrated by the introduced imaginary steps of the prediction in figure 8. Hybrid Bayesian networks which include both discrete and continuous nodes to achieve higher prediction resolution would be the first task for the future work list. Moreover, the focus of this paper was in the predictions and proactivity, which are followed blindly by Bayllocator. Proactivity solve some problems of reactivity which it might be slow to sudden large memory changes, but reactivity is mandatory in a dynamic resource allocation system because if the prediction is wrong, then the system will perform very inadequately. Implementation of reactivity which will be ruled by predictions is another key improvement for Bayllocator. Design of a more sophisticated Bayesian network with more accurately chosen nodes is also under work.

7 Acknowledgements

This paper is work made for a master thesis in system administration at the University of Oslo and Oslo and Akershus University College. Therefore, we would like to express our gratitude to the institutes provided the test equipment, and the people involved directly or indirectly.

8 Availability

The scripts and prototype software created for this project are based on Perl and R statistics language. They can be acquired upon request by email. Further details can be found in the accompanying MSc thesis [14]

References

[1] ABRAMOVSKY, L., AND GRIFFITH, R. Ict, corporate restructuring and productivity.

[2] BOBROFF, N., KOCHUT, A., AND BEATY, K. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on* (2007), IEEE, pp. 119–128.

[3] CHARNIAK, E. Bayesian networks without tears. *AI magazine 12*, 4 (1991), 50.

[4] CLARK, C., FRASER, K., HAND, S., HANSEN, J., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2* (2005), USENIX Association, pp. 273–286.

[5] GARTNER, I. Press release: Gartner says more than 1 billion pcs in use worldwide and headed to 2 billion units by 2014. http://www.gartner.com/it/page.jsp?id=703807, Jun 2008. [Online; accessed 17-February-2012].

[6] KERSTEN, D., MAMASSIAN, P., AND YUILLE, A. Object perception as bayesian inference. *Annu. Rev. Psychol. 55* (2004), 271–304.

[7] KOLLER, D., AND MILCH, B. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior 45*, 1 (2003), 181–221.

[8] KUKREJA, G., AND SINGH, S. Virtio based transcendent memory. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on* (2010), vol. 1, IEEE, pp. 723–727.

[9] MEISNER, D., GOLD, B., AND WENISCH, T. Powernap: eliminating server idle power. *ACM SIGPLAN Notices 44*, 3 (2009), 205–216.

[10] NEAL, R. *Bayesian learning for neural networks*, vol. 118. Springer Verlag, 1996.

[11] PADALA, P., SHIN, K., ZHU, X., UYSAL, M., WANG, Z., SINGHAL, S., MERCHANT, A., AND SALEM, K. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review 41*, 3 (2007), 289–302.

[12] SCHMID, M., MARINESCU, D., AND KROEGER, R. A framework for autonomic performance management of virtual machine-based services. In *Proceedings of the 15th Annual Workshop of HP Software University Assocaiton. Hosted by AI Akhawayn University in Ifran, June* (2008), pp. 22–25.

[13] SCHOPP, J., FRASER, K., AND SILBERMANN, M. Resizing memory with balloons and hotplug. In *Proceedings of the Linux Symposium* (2006), vol. 2, pp. 313–319.

[14] TASOULAS, E. Bayllocator: A proactive system to predict server utilisation and dynamically allocate memory resources using bayesian networks and ballooning. *MSc thesis, University of Oslo, http://www.duo.uio.no/sok/work.html?WORKID=164621* (2012).

[15] WALDSPURGER, C. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review 36*, SI (2002), 181–194.

[16] WILLIAMS, D., JAMJOOM, H., LIU, Y., AND WEATHERSPOON, H. Overdriver: Handling memory overload in an oversubscribed cloud. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2011), ACM, pp. 205–216.

[17] WOOD, T., TARASUK-LEVIN, G., SHENOY, P., DESNOYERS, P., CECCHET, E., AND CORNER, M. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2009), ACM, pp. 31–40.

[18] YATES, S., DALEY, E., GRAY, B., GOWNDER, J., AND BATIANCILA, R. Worldwide pc adoption forecast, 2007 to 2015. *Forrester Research Report* (2007).

[19] ZHAO, W., WANG, Z., AND LUO, Y. Dynamic memory balancing for virtual machines. *ACM SIGOPS Operating Systems Review 43*, 3 (2009), 37–47.

[20] ZHOU, W., YANG, S., FANG, J., NIU, X., AND SONG, H. Vmctune: A load balancing scheme for virtual machine cluster using dynamic resource allocation. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on* (2010), Ieee, pp. 81–86.

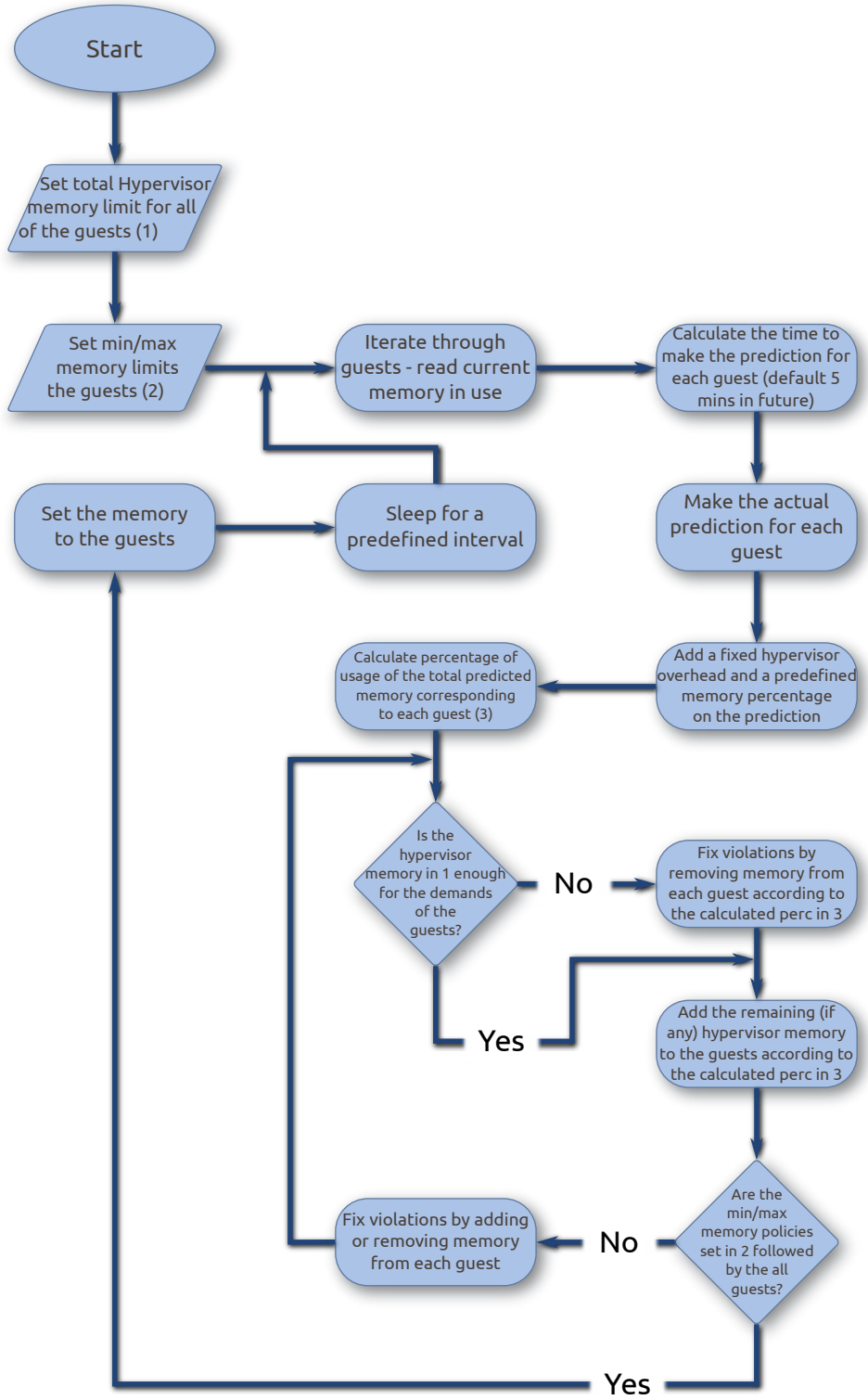


Figure 11: bayllocator.pl: Main script to dynamically allocate memory using the predictions

A sustainable model for ICT capacity building in developing countries

Rudy Gevaert
Ghent University
Rudy.Gevaert@UGent.be

Abstract

System administrators are often asked to apply their professional expertise in unusual situations, or under tight resource constraints. What happens, though, when the "situation" is a foreign country with only basic technical infrastructure, and the task is to build systems which are able to survive and grow in these over-constrained environments?

In this paper we report on our experiences in two very different countries – Cuba and Ethiopia – where we ran a number of ICT projects. In those projects we assisted local universities to upgrade their ICT infrastructure and services. This included skills and process building for local system administrators.

Based on our experiences we formulate a model for sustainable ICT capacity building. We hope this model will be useful for other organizations doing similar projects.

1 Introduction

Universities are one of the many actors in international development programmes across the world. They actively participate in fields such as health care, food security and ecological management to name a few.

Many of these development programmes use Information and Communication Technology (ICT), but there are hardly any ICT-only programmes[1]. ICT is seen and actively promoted, by the Organisation for Economic Cooperation and Development (OECD), as a way to achieve the Millennium Development Goals[2].

In this paper, we report on our experiences in organizing ICT-for-development programmes. One of our objectives is to build sustainable ICT capacity in our partner's institution. This paper highlights some of our activities over the last ten years. We also formulate a model, which we hope may be of assistance to others providing similar aid.

We begin by establishing background and context (Section 2). Currently running programmes in Cuba and Ethiopia are described in the following sections (Section 3, 4). In each of those sections we first sketch the programme and continue with a discussion of the ICT-specific aspects (Section 3.2, 4.2). Section 5 proposes a model, based on our experiences, for other organizations using ICT in their development programmes. In Section 6 we give our conclusions and share our plans for the future (Section 7).

2 Background

Before we can report on our activities it is necessary to have some background information regarding the financing and coordination of our programmes.

The programmes we have been involved in were organized by the Flemish Inter University Council for University Development Cooperation (VLIR-UOS). Since early 1998, VLIR-UOS has been the responsible actor for the Belgian government, on behalf of the Flemish universities, for all university cooperation for development[3]. The Belgian federal government finances VLIR-UOS through the ministry of development cooperation (DGOS). From the received funds, 60% is required

[1] According to OECD figures (Table 19. Aid by Major Purposes in 2010, http://www.oecd.org/dac/aidstatistics/47452930.xls, accessed 9 August 2012), development aid is largely targeted at *Social and Administrative infrastructure*. There is no specific tracking of development aid of ICT programmes, although Communication is included under the more general infrastructure heading *Transport and Communication*. For European institutions that's 39.7% of Socical and Administrative infrastructure aid, compared to 6.4% for Communication aid (Belgium is 28.6% versus 0.7% respectively).

[2] See http://www.oecd.org/development/20611917.pdf, p1 "Introduction" and p2 "How can ICTs be integrated into development programmes?"

[3] Alternatively, the Commission Universitaire pour le Développement (CUD) is the responsible actor on behalf of the universities in the Walloon region of Belgium.

to be spent in the developing world.

The key role of VLIR-UOS is to act as a facilitator for the programmes. That is, the universities propose programmes, that can be executed after approval by VLIR-UOS and DGOS. VLIR-UOS is responsible for the general policy, selection, observation and evaluation of the executed programmes. VLIR-UOS is accountable to the Federal government for the use of the funds.

VLIR-UOS provides the Flemish universities several different frameworks to use for development cooperation programmes. Some programmes are short, lasting only one year and hope to achieve regional impact. Other programmes focus on long-term goals and aim at a country-wide impact. These programmes last ten years, but can be extended up to fifteen years[4].

In terms of duration and available budget, the largest framework available is the *International University Cooperation* (IUC) programme. The main objective of the IUC programme is to:

> Empower the local university as institution to better fulfill its role as development actor in society.
>
> This objective is to be attained through the implementation of a coherent set of interventions, guided by the strategic plan of the partner university, aimed at improving institutional policies and management and the quality of local education, research and societal service delivery.
>
> *Source* `http://iuc.vliruos.be/index.php?language=EN&navid=507&direct_to=Objectives_and_features`

The IUC programmes takes 30% of VLIR-UOS' budget. Currently there are ten active IUC programmes spread over several countries: Cuba, Ecuador, Ethiopia, Kenya, Mozambique, Peru, South Africa, Suriname and Tanzania.

Another framework that can be used for university development cooperation is the *IUC crosscutting*. The key idea is to organize theme-based and programme-wide initiatives, that is, initiatives which "cut across" and benefit multiple areas. For instance, ICT capacity building is one of the main themes. Another crosscutting programme is the *North South South (NSS) Cooperation programme*. The objective of this programme is to improve the process of institutional capacity building through collaboration with other partners running a IUC programme. The VLIR-UOS website `http://iuc.vliruos.be` contains more information about these and other programmes.

[4] After the tenth year the participation becomes competitively assessed and budgets are more limited.

3 Case study: ICT capacity building in Cuba

In 2003 VLIR-UOS started an IUC programme with the university of Santa Clara, Cuba. The university, Universidad Central Marta Abreu de las Villas[5] (UCLV), is located in the most central region of the country. Santa Clara (see Figure 1) is a typical Cuban town that isn't overrun with many tourists. The burial place of Ernesto Che Guevara, located at the outskirts of the city, attracts its share of tourism but few tourists stay overnight in Santa Clara.

Next, we will first give an overview of the whole IUC programme at UCLV. Following that, we will describe our experiences in the ICT infrastructure part of the programme.

3.1 Programme context

The programme's focus is not limited to ICT infrastructure. There are many different projects:

- ICT infrastructure.
- ICT in education.
- Library development.
- Capacity building for communication in English for academic purposes in international collaboration.
- Institutional development.
- Improving the quality of graduate and postgraduate education and research programmes in plant and animal sciences.
- Strengthening undergraduate and graduate education in pharmaceutical sciences.
- Environmental education and development of clean technologies.
- Strengthening research and postgraduate education in computer sciences.

Development cooperation in Cuba, in the broadest sense, is in some cases very similar to development cooperation in other developing countries. In other cases it is *very* different.

As a first example, historically Cuba has had the highest rates of education and literacy in Latin America, both before and after the revolution. All education, including University education, is free to Cuban citizens[6]. As a consequence we were able to work with very skilled people.

As a second example, the US embargo against Cuba makes normal trade and foreign investments nearly impossible. In other words, forget about easily buying hardware from the US. Even if Miami is only 366 km from

[5] `http://www.uclv.edu.cu`

[6] The United Nations Statistics Division `http://data.un.org` reports an adult literacy rate of 100% for Cuba, compared to 30% for Ethiopia

Figure 1: Map of Cuba, with the location of our project in Santa Clara

Havanna, Cuba's capital. The embargo also limits access to information. Downloading software from servers located in the US isn't always possible. For instance, popular code hosting sites like Sourceforge and Google Code have blocked access for Cuban IPs.

As a final example, the government system is based on socialist principles. The largely state-controlled planned economy influances decision making from top to bottom. It leads to situtions where people are willing to do things, but simply can't because the system doesn't allow it.

In this paper we will not further discuss the differences between socialism and capitalism. We have decided to accept the situation and make the best of it.

3.2 The ICT infrastructure project

After ten years of active collaboration with our counterpart in Cuba our programme will close in 2013. This makes it possible to identify the key results of the ICT infrastructure project on a long term basis. The ten year programme is split into two phases: phase one was from 2003 to 2007 and aimed at capacity building; phase two is from 2008 to 2013 and aims consolidation and valorisation. In the following subsections we are going to describe these achievements.

3.2.1 Establishment of a solid network infrastructure and associated services

During the first phase the university backbone was expanded to connect all buildings of the university. The deployed backbone consisted of 7 km of optical fiber. The fiber was connected by the university staff. To make this possible the programme purchased the necessary equipment (fiber verification kit, splicing kit) so future deployments could be handled too. It also gives the staff the possibility to fix fiber cuts.

At the same time the necessary services like e-mail (Exchange, Postfix, Amavis and ClamAV), dial-in[7], VoIP (Asterisk), instant messaging (Jabber), web hosting, file serving (DAS with Samba and Windows file servers), monitoring (Nagios, Munin) were implemented. To use these services each user requires a centralized account.

In each faculty a computer lab was installed. The computers were received through the Close The Gap project[8].

During the second phase, emphasis was placed on improving and optimizing the ICT infrastructure. The storage system was expanded up to 40 TB. Redundant paths were added to the network, and wireless access points were installed. Because of the annual thunderstorms and hurricanes the electrical grounding system was improved and lightning protection was installed.

To comply with governmental policies the migration to Free Software[9] technologies was started[10]. By the end of the programme the Exchange e-mail system will be replaced (Dovecot and RoundCube webmail) and the

[7] Broadband access is not easily available in Cuba, which makes dial-in modem access the most common way for Internet access from home.

[8] Close the Gap is an international not-for-profit organization that helps bridging the digital divide by offering cost-efficient high-quality used IT-equipment to projects in developing countries. Socio-educational programs such as schools, hospitals and other projects focusing on improvement of educational and information facilities can ask for support from Close the Gap. `http://www.close-the-gap.org`

[9] Here we do not mean free, as in free beer, but as in freedom.

[10] In 2007 the Cuban government decided to move from proprietary to Free Software.

Active Directory server will run together with a Samba4 Directory server.

3.2.2 E-administration system developed

In the first phase the software development team implemented a student administration system in Java. They were able to use Ghent Universities student administration system as a starting point. The Cuban staff made the necessary changes to make the software work according to the Cuban higher education policies. Next to that, they implemented a system to manage university wide ICT accounts.

The implementation of a unified management system for administrative and academic purposes was started during the second phase. Part of the process involved implementing a enterprise messaging service bus (JBoss) that connects the different information systems at the university.

3.2.3 Internet access improved and managed

Cuba was first connected to the Internet in 1996 and currently has about 379 Mbps downstream and 209 Mbps upstream. The Internet bandwidth at the university is 2 Mbps, which is shared by 3300 computers. To compare, a residential user in Belgium has a download speed of 30 Mbps. At the time of writing there is a undersea cable (10 Gbps) between Cuba and Venezuela, but it is not yet in use. It's assumed that the cable will be in use "soon". Which has been rumored for the last two years.

Because of the growing amount of computers that need Internet connectivity our programme focused on increasing the bandwidth availability and usability. This was done through negotiations with the government and through managing the scarcely available bandwidth. A caching proxy was installed and quota on the amount of bandwidth available for each user was enforced. To manage the quota a web application in the Symfony PHP framework was developed.

3.2.4 Establishment of a central data center

During the first phase (year $1-6$) of the programme a small server room was installed. The server room hosted a couple of desktop computers that were functioning as servers. At the start of the second phase (year 7) the construction of a new and dedicated data center room was started. The data center was provisioned with HVAC, a UPS and a generator. The desktop servers were replaced with rack mountable servers. In the next to the last year of the programme (2012) Ghent University donated IT equipment that was recently taken out of service. The container that was shipped by boat contained rack mountable servers, blade chassis' and servers, 42U racks, in-row coolers and devices for environmental monitoring.

This equipment is now installed in a new data center room at the university. However not all equipment is being used. Currently the campus power grid is under provisioned and not capable of serving three phase electric power in the new data center. Another problem is that the in-row coolers make use of a centralized water based cooling system. Currently we can not afford to purchase the chiller to cool the water. We were aware of these problems when we shipped this material, but our partners insisted[11] on shipping it. They are now looking at ways to get the power grid upgraded. We are at the same time looking for a decommissioned chiller or another alternative.

3.2.5 Human capacity building in software engineering, system and network administration

In order to build the necessary human capacity we organized several training courses in Cuba and Belgium. Every year several Cuban ICT staff visited Ghent University for several weeks. During these visits the staff worked with our guidance on their projects. When the staff returned to Cuba, they organized training for their co-workers so the gathered knowledge was shared.

The importance of these visits can not be underestimated. It gives the Cuban staff the necessary exposure to current and upcoming technologies. By connecting with peers in the field of system and network administration and software engineering they can learn new ways of problem solving. Whilst at the same time we can learn from them too. Another advantage of these exchange visits is that they can use unrestricted and fast Internet access. This makes researching ('googling') far more effective.

Several training courses were also organized in Cuba. For instance, training on data center design and management was held when the university received the donation of servers and data center equipment. We also helped with the design of the network. Training on web-development with the Symfony PHP framework was organized to increase the productivity of the web developers and to move to agile web development. To manage their servers DevOps practices were introduced and a short training course on configuration management with Puppet was given.

It is important to mention that the Cuban ICT staff didn't need much training. They are very skilled and adapt very easily to the new technologies we introduced. During our visits it was sometimes sufficient to give a short introduction on a particular topic, which they

[11] Which isn't that unreasonable, you never know which funds or opportunities will arise in the future.

would study and implement by our next visit. It is amazing what they can achieve considering the difficult conditions. They are so used to their system that they know how to manage the system for their benefit.

Our experiences in ICT for development aren't limited to Cuba. In the next section we will discuss the programmes we are running in Ethiopia. Similar to this section we are going to start with providing the necessary context of the two programmes before going into the ICT related projects.

4 Case study: ICT capacity building in Ethiopia

VLIR-UOS has two IUC programmes in Ethiopia. The first programme is with Mekelle University[12] (MU) and was started in 2003. In 2007 a second programme was started with Jimma University[13] (JU). In Figure 2 you can see that Mekelle is located in Tigray, the Northern province of the country, while Jimma is located in the Southern part of Ethiopia.

Before we report on the ICT specific part of the programmes we will first give an overview of the other projects in each programme.

In contrast to the programme in Cuba (Section 3.1) these programmes have a more multi-disciplinary approach as can be seen in the next subsection. In the following subsection we report on our experiences.

4.1 Overview of the MU-IUC and JU-IUC programme

The overall objective of the MU-IUC programme is to contribute towards sustainable livelihood in the Tigray region. The projects are:

- Enhancement and optimization of ICT usage.
- Upgrading of the library services.
- Cluster support service.
- Enhanced crop production through improved irrigation water management and water-saving techniques.
- Socio-economic research for sustainable rural livelihoods.
- Ecological integrity and sustainable management of standing waters.
- Land degradation and rehabilitation at the scale of the Geba catchment.
- Appropriate farm technology for vertisol[14] management.

[12] `http://www.mu.edu.et`.

[13] `http://www.ju.edu.et`.

[14] A Vertisol is a soil in which there is a high content of expansive clay that forms deep cracks in drier seasons or years. `http://en.wikipedia.org/wiki/Vertisol`

In comparison, the JU-IUC focuses on the impact of the Gilgel Gibe hydro-electric dam in terms of human and animal health, ecology and agronomy. Joint research is undertaken in different disciplines in the Gilgel Gibe area to improve the life quality of communities. Furthermore, research and educational capacities of Jimma University academic staff will be extended. The programme has the following projects.

- Zoonotic and animal diseases.
- Child health and nutrition.
- Environmental health and ecology.
- Epidemiology and modeling.
- Soil fertility.
- ICT and library.

In what follows we will report on our experiences of the ICT projects in Mekelle and Jimma. We can do this because there is a high degree of overlap between the needs regarding ICT.

4.2 The ICT infrastructure projects

To appreciate the problems we solved it is necessary to describe the situation in Jimma when the programme started.

The Internet access at JU in 2007 was 4 Mbps. There were a couple of Sun Blade 1000 work stations (512 MB RAM, one UltraSPARC III CPU) running Solaris 8 that were used as proxy server (Squid), DNS server (Bind) and web server (Apache). There were also two Sun Fire v880 machines that weren't even in use! The core network switches were two Cisco Catalyst 6500 devices[15]. What made the situation problematic was that only one person was running the ICT for the whole university. He also didn't really understand how to configure any of the above software.

For example, from time to time 'the Internet broke'. The only solution was to wait for him to fix the problem. He would then fix the problem, till it happened again. Actually, the problem was that there was no log rotation on the Squid log file so it would fill up the partition and make the system stop working. His solution was to delete the log file each time which didn't prevent the problem from happening.

Given the fact that the person in charge was unwilling to share his very limited expertise with junior staff (two bachelors) he was in a very powerful position. He couldn't be replaced, so it seemed, because nobody had the knowledge to keep the system running nor did anybody else have the root passwords. Nobody, except him had the key to the server room.

MU had exactly the same hardware was JU. In fact, all the bigger Ethiopian universities had the same hardware.

[15] To put this into perspective, Ghent university used in 2007 only a couple of the 3500 series.

Figure 2: Map of Ethiopia with the location of our programmes in Jimma and Mekelle

This was because the World Bank donated a lot of IT equipment in the beginning of this decade. MUs staff tried very hard to manage their infrastructure, but they also had only one person with some skills.

To summarize the situations; in both universities there was a working small WAN with very limited and unreliable services.

Before we continue with reporting on the ICT projects we remind that the programmes in Ethiopia consist of two phases. Phase one (2003 – 2007) and phase two (2008 – 2012) in Mekelle. Phase one (2007 – 2011) and phase two (2012 – 2016) in Jimma.

4.2.1 Apply bandwidth management and optimization techniques

During the first phase of the JU-IUC programme there was a lot of attention for bandwidth management and optimization. The key objective was to provide reliable and as quick as possible Internet access.

To achieve this the following actions were taken. During the first three years all the Sun hardware was replaced with rack mountable servers running Debian GNU/Linux. This meant that the junior staff needed to be trained in basic system administration: handling servers, installing Debian, setting up software by following HOWTOs, reading manual pages, using mailing lists and forums, ... During several training sessions in Belgium and Ethiopia the staff was trained in DNS, Squid, Apache, Dansguardian, MRTG, Collectd and Icinga.

At the end of the first phase the JU ICT staff had successfully implemented their own authoritative DNS server[16]. The setup makes use of internal and external zones and uses a couple of slave DNS servers. The staff also implemented a caching Squid proxy server that applies the following bandwidth management and optimization techniques:

- Time based ACLs. E.g. Facebook and Youtube aren't allowed during office hours.
- Limit the bandwidth usage per IP with Squid delay pools.
- Use Dansguardian with ClamAV to do content filtering and anti virus checking for the Windows

[16] At the time, only Addis Abeba university was the only other Ethiopian university with their own authoritative DNS server. Ethiopian telecom is very reluctant to hand over control. Getting permission from them was more challenging than setting up the DNS itself.

computer labs

To save bandwidth when installing software local mirrors for Debian and Ubuntu were set up.

4.2.2 University e-mail system created

During the first phase of the JU-IUC programme a local e-mail system was set up. As there wasn't any real server hardware available, a desktop PC was installed with Debian GNU/Linux, Postfix, Amavis, Courier IMAP and Sqwebmail. The next year a rack mountable server was installed and the data was migrated to the new system. Over the last five years the number of mailboxes grew from 249 to 2343.

The successful implementation of the local e-mail system prove to be very important for the whole IUC programme. Next to saving bandwidth it made communication between the staff easier. Users enjoyed using the local webmail system, as they were previously waiting many minutes for GMail or Yahoo mail to load. Using the local e-mail system also meant that when Internet access was down people could still send e-mail to each other locally. When Internet access returned, e-mail from abroad would gradually be delivered into their inbox.

Very important for the adoption of the system was that everybody in the JU-IUC programme was "forced" into using the university e-mail system. The coordinators from Belgium and Ethiopia said they would only reply to e-mail messages if sent from the university e-mail system. Having a critical mass proved to be very important. It ensured that the ICT staff was aware of the importance of the system. In the MU-IUC programme management couldn't be convinced of this, which meant a much slower adoption. Because the staff involved in the JU-IUC programme was so happy with their official e-mail accounts, other staff members soon followed.

At the end of the first phase the staff of JU and MU jointly organized an international training course for other VLIR-UOS partners on the installation and administration of an e-mail server. This training built capacity at the other partners, but more importantly it built additional capacity in their own institutions. Other staff members were given the opportunity to learn from their colleagues, who created an official training program that they could follow. It also required the trainers to fully understand their own system, as they were going teach about it.

4.2.3 Creation of a user management system

When the e-mail system was set up the account creation and management needed to be done on the command line. This was error prone and account creation was slow. Only one person was trusted to create the e-mail accounts. When this person was away from the office no accounts were created.

The HR office and the registrar office couldn't provide an up to date list of the staff or students. This meant that importing user information wasn't possible. This made us create a web interface to manage the e-mail accounts.

To create the web interface, a one month training period was organized in Belgium. Two staff from JU and MU undertook training which focused on developing web applications with the Symfony PHP framework. The objective was to learn how to use the framework, and to create a web application for managing their institutions' user accounts. A pre-built web interface could have been used too, but then the capacity to create web applications would not have been built.

As the primary users, the JU and MU staff knew what was important in the design and how it would be used. They also made sure that the interface was available in English and Amharic, their official language. They designed the application interface to reflect Ethiopian practices. For instance the naming practice is totally different. In Ethiopia children get a given name followed by their father's given name and their grandfather's given name.

From the beginning, the system was designed with the future in mind. Instead of focusing on only e-mail accounts, the database model left room to add other types of accounts such as file server and Internet access accounts, ... The idea was that the system would be expended during the following years.

After this training the developed system was ready to be deployed. It could be expected that when the trainees returned to their institution they would get the system up and running in no time. However, this almost never happens. This was foreseen – at the end of the training I went to Ethiopia to help them deploy the web interface.

It was expected that the system would be expanded over the coming years. Unfortunately, this didn't happen straight away. Of the four participants, two started their masters study, one was given other tasks at the university and the last one no longer wanted to program.

This put us in a awkward situation. A lot of money was spent training the four staff. The training produced a web interface that was useful, but it also had some bugs and could not be maintained. We expected that the trainees would train other staff on the framework and application. Given the situation, this wasn't going to happen soon. We needed to build more capacity on using the PHP Symfony framework.

To achieve this a second training course was necessary. However, for the training to take place the four staff who had completed the first training course needed to extend the web application. This demonstrated they still cared and could take ownership. The second training

course was organized in Ethiopia and participants from other IUC programmes (South Africa, Tanzania, Kenya, Mozambique, Zimbabwe) were also invited. As a result of this second training course, more staff of JU and MU were trained, leading to real ownership of the web application.

After five years, several other web applications were developed and used. At each university there are a now a number of staff who really understand the framework, and are able to maintain and develop applications using agile software development techniques. The biggest challenge that still remains is how to further distribute the knowledge of the framework to the new staff.

4.2.4 Capacity building in Free Software technologies

As a consequence of the previous activities, a lot of capacity was built in the use of Free Software technologies. At programme inception, the ICT staff weren't even aware of what Free Software was available, and how it could be used. The introduction to software that could be freely shared, modified and used was very important for the programme. The time lost looking for cracked software and passwords could now be invested in studying how to actually use the software. Free Software also made it possible to share the software legally with other universities in Ethiopia.

The projects built serious capacity in using and applying Free Software to solve problems. Staff members were confident to use additional software. For instance one staff member set up a VOIP system with Asterisk. Another staff member set up NeDi as an alternative to Cisco Works, to aid in the management of the network. The ICT helpdesk started using the OTRS ticketing system. Currently they are in working on implementing OpenLDAP and Samba4.

5 A sustainable model to build capacity in ICT

The experiences in the above case studies are only some of the stories that can be shared. More important are the things that I learned the last six years when I was member and leader of several ICT projects in Cuba and Ethiopia. Below is a model to do development cooperation in ICT. Development organizations or volunteers willing to implement ICTs can use the model when developing their programmes.

5.1 Bandwidth is limited, manage it!

When doing projects in developing countries you should be aware of the limited bandwidth. The only sustainable way to handle this problem is to implement bandwidth management and optimization (BMO) techniques.

Increasing the bandwidth by paying for a faster Internet connection will ease the pain in short term. There will come a day that your partner will have to pay for the bandwidth themselves. If they haven't got the funds they will fall back into their initial situation, that of a slow and unmanaged Internet connection. On the other hand, if they are able to increase the available bandwidth they will use it more cost effectively.

The most common BMO technique is the implementation of a local caching proxy server. The proxy server can then enforce certain policies of who can download what and when. Next to that, the implementation of local services is very important. By offering local services you can save bandwidth and decrease the page loading time. Local DNS, e-mail and websites are the basic Internet services that should be run on the local network.

Unfortunately, what can be cached or offered locally is limited. Social network sites and streaming video sites consume large amounts of bandwidth. Here it is important to define the policies on what is allowed and when. For instance, blocking Youtube and Facebook access during office hours is a fair thing to implement.

Setting up local software mirrors can save bandwidth too. It's very easy to set up a mirror of a GNU/Linux distribution. In situations where the initial sync takes too long or is impossible, bringing a USB hard disk with the mirror on is almost always faster. Another possible solution is to provide a service similar to Youtube on the local network.

Focusing on technical aspects is important, but it should not be the only objective of your project. In the next subsection we describe the importance of building human capacity.

5.2 Capacity building is more important than infrastructure building

It's very easy to go in overdrive and buy a lot of expensive equipment when doing ICT projects. Many times this is done in good faith, but unfortunately this doesn't lead to the best results. In a worst case scenario, the equipment isn't even used. To prevent this, it is crucial to start building human capacity from the beginning. Organizing training that leads to actual implementation will create awareness for the deployed solution. At the same time, this training can be used to identify the most motivated and more expert staff.

In development cooperation it's common to use the *train the trainer* principle. This is a great way of creating capacity, but it's not easy. Many projects succeed in training the trainer, but then fall short of getting the trainers to successfully deliver training. Expecting that

trainers will take the initiative to train others, has only a small success rate. A successful way of overcoming this is to organize the initial training, coupled with followup training *given by the new trainers*. If the trainers know in advance that they will be training others soon after their own training, they will be more motivated. It will also give their own project more visibility and establish them as leaders amongst their peers.

When selecting topics for capacity building it's important to work demand driven. Enforcing certain solutions will not lead to success. It's necessary to identify the problems and to work on fixing them. You create owner ownership of the solution this way. In the end, this is a formula for success.

Training can be delivered locally or abroad; the choice of location depends on a number of factors. Local training, that is, in the countries where aid is being provided, allows you to teach more people. This is primarily due to cost – international airfares are expensive! However you need to be prepared for frequent power cuts, trainees arriving late or not at all, slow Internet connection, old computers, and more. You need to be prepared for everything; a good start is making sure the training can be given off-line. Delivering training abroad gives the trainees the chance to break out of their normal environment and get exposure to technologies that are difficult to demonstrate "at home". This gives trainees the opportunity to see what they are working towards. For instance, you can repeat endlessly how things are organized in a modern data center, but it's more effective to take someone to a modern data center to see how it operates.

When organizing training, it's important to consider that not all participants will actually implement what they learned. There are many reasons for this; the selection of the participants is not always done correctly, for instance you can end up with a network technician following a software engineering training; or participants don't always get the permission to implement when they return to their institution. Another reason why learned skills aren't implemented is the high staff turnover. People easily change jobs. It is very demotivating when you've just trained several people to find out they they left the institution. It also happens that they go back to school for their masters study. Some staff will abuse the gathered knowledge. They will explicitly not share their knowledge, because they see it as an advantage over their colleagues. When doing projects and training people it's something you can only live with. The only real chance you have is to put your eggs in many baskets.

Our experience over multiple training courses is that capacity building can be successfully achieved as follows:

- Train a broad group of staff in a certain topic.
- Identify those who grasped the topic.
- Let the identified people implement the solution.
- Organize a second training where you retrain them, they now become experts, together with other staff.
- Let the experts give training to people outside of the project so they get very confident and know the topic very thoroughly.

In the next subsection we go into detail why building this capacity isn't easy, but is very rewarding.

5.3 The level of knowledge is limited, but the people aren't!

Working together with local people will be challenging. Many of the people in IT weren't computer users when they grew up. In fact, they could have grown up without access to electricity or clean water. Most of the young people will have a Bachelor degree. Unfortunately you can not compare their bachelor, or master, degree with a European or US degree. For instance, it's very normal to finish a programming course without writing code on a computer! This isn't uncommon, particularly if there aren't enough computers at the university. The education system is still heavily based on reproducing theory and less on practice. Basic computing skills, such as touch-typing, are poor because users have had much less exposure to computing systems.

Sometimes it is necessary to explain things many times, each time a bit differently. The language barrier definitely causes problems, even in countries where English is the official teaching language.

It is also important to adapt to the local culture. Sometimes failures aren't admitted or the truth is not told because individuals fear punishment, or their pride is hurt.

The previous examples are only a few of the issues that will arise. Even given their sometimes-limited knowledge, the people you will work with are capable of doing big things. When given the chance they will certainly try to take it. But due to external circumstances not everything leads to success. Which makes it very important to have a good project management.

5.4 Management

In developing countries the awareness of ICT is very low. Therefor it's important that the management supports your project. You need to make sure they give the good example by using the ICT systems you implement.

Beware not to be used as a the milch cow. Even bigger organizations in developing countries can afford to buy computers and other office materials. If project money is used to buy flash memory and ink toners priorities aren't defined right. An exception would be Cuba, where even buying paper sheets is difficult.

When working with the local people it's crucial to build up a team of honest and hard working people. Because you are the foreigner you automatically attract people who will want to take advantage of you. For example, when a training course is organized there will be a lot of candidates. It's then important you can trust your partner to select the right attendees. If your partner can't make the right selection you could end up with a room of participants attending the training solely to obtain a certificate, with no intention of using their new knowledge. If your partner puts himself always between the selected participants, it's time to look for a new partner or modify the selection process.

Development cooperation is big business. It happens that donors are working next to each other. Or even worse, knowingly implementing different or similar technologies. Local partners don't always inform the parties concerned as they are worried one donor would withdraw their cooperation. Another consequence of the "too-many-donors-syndrome" is staff attend multiple courses, hopping from training to training, without implementing anything. If you can establish a good relationship based on mutual respect, your local partner will surely inform you about other activities going on and assign people to do specific tasks.

It is necessary to have continuous leadership. In Mekelle there were six different local project leaders and four different Flemish project leaders. This lead to loss of long term vision and made follow up almost impossible. In Jimma the first six years the leadership was constant. Only in phase two the Flemish project leader was changed. Continuous leadership makes it possible to keep the objectives of the project aligned with the objectives of the local partner.

5.5 Purchase of IT equipment is difficult

The implementation of ICT requires buying computers and servers. In most developing countries you can forget about next business day support. Many vendors have local resellers but they can't be used for support. From experience, we can say that the resellers have very limited knowledge of the equipment they sell. Desktop computers are easily available in most developing countries, except Cuba. Servers and networking gear are also available, but not the high end ranges and the choice in specifications are limited. When you place an order you can't be sure that you will get it. The main advantage of purchasing locally is that your partner should know how to order. If he doesn't, the process can be tried to see how it works. This creates the local procurement capacity that is necessary for ongoing sustainability.

Another option is to buy through European or American resellers and to ship the goods internationally. However, it's important to take into account delays when importing the goods. Without proper preparation this could easily take several months. Many countries have a very high import tax that can easily double or triple the price. So it's important to know that some countries lift the import tax if you can prove that your project is donating for relief aid purposes.

When purchasing IT equipment it's also good to purchase several items at once. Buy five or ten servers at the same time. Once the project gets going you can't afford to wait another half year (or longer) for your next server to arrive. By buying more hardware than you need the local staff can experiment with the extra hardware. The extra servers can also be used as spare parts.

If licensed software is necessary, think on the long term. Will your partner be able to pay the renewal of the license when the project stops? Even with funding, will your partner actually be able to make the purchase? The chances are very slim that they will have a credit card to make the purchase online. It's very important that the software can be bought locally and that the purchasing process is tested.

Also check the requirements to operate the equipment you buy. For instance, when purchasing a high end UPS make sure the data center actually has a three phase power circuit. Another example is when expending the network make sure your partner has the necessary interface converters so he can use the new equipment immediately.

Once the equipment arrives on site it's important to follow up on the usage. It happens that the equipment is put in *the store* for bureaucratic reasons and then is left in the store forever and forgoten about. Frequently equipment isn't installed correctly: rail kits aren't used, only one power supply connected, ... By personally checking and training the local people to mind these details will eventually pay off.

To achieve maximum ownership it is important your partner pays and orders the equipment as soon as possible. It will put the ownership with your partner very early in the project. Otherwise your partner may abandon the deployed setup. He could wait for the next donor to take over, or to implement something different.

5.6 Small is beautiful

A mistake made frequently is to over-engineer solutions. It is better to start with a small and easy solution that can grow gradually. When implementing you must make that your partner can follow what you are doing. Once implemented, you must be able to step back and honestly say if your partner is able to maintain the solution. Of course, it's better if your partner can do the setup himself.

When starting with smaller and easier projects you can

involve less trained staff and build their capacity. You should value these people the most. They will take every chance they get and will stay longer in the project. In developing countries it's not uncommon that only the senior people are given training, you must take initiative to prevent this.

In bigger projects it can be necessary to rely on a third party for certain aspects of the project. When going down this road it's important that your partner is trained by the third party too. Be aware that consultants will abandon the project when their contracts finishes. If all knowledge is still with the consultant and none has transferred to your partner no progress was made.

5.7 How to handle project follow up

During the lifespan of the project it's important to constantly follow up on the progress of the project. This can be done by requesting reports. However, initially don't expect to get reports voluntarily. It's better to request reports and install a culture of reporting. Far more better than requesting reports is to visit the project in person. Personal relationships are more appreciated than impersonal e-mails. It also gives the staff more opportunity to raise questions and get guidance. With one visit a year you can't achieve much, two visits are the bare minimum.

Another way of following up is to use social networks. Become friends with everybody in the project. Sure, you will have to ignore a lot of uninteresting messages, but you will at least know when something happened. Creating a Google group after a training that connects all the participants is a great way to create a support channel for the participants.

6 Conclusion

Our experiences in supporting ICT projects in Cuba and Ethiopia lets us present a sustainable model for ICT-for-development. To be able to build a sustainable ICT project it is necessary to take into account the following points:

- The available bandwidth in developing worlds is limited. Implementing a caching proxy and other bandwidth optimization techniques is necessary.
- An ICT project in the developing world should focus on creating human capacity. A project that only invests in hardware and software will fail. Frequent ongoing training is the way to handle high staff turnover. This training should be hands-on and should lead to actually implementing a learned technology. The training and implementation should be given equal attention.
- The local people are very eager to learn. Their educational background is not perfect but they make up with enthusiasm and dedication. If possible, let your partners get a degree in a ICT-related field.
- It's important to have a good relationship with the local management. They know the 'ins and outs' of the system. You should be able to rely on them to select candidates for training. It is necessary to get your project's objectives aligned with your partner institutions' objectives. This way you will work together and the local people will take ownership of the solution.
- The purchase of hardware or software is not easy. Purchasing abroad or locally each have their own advantages and disadvantages. In either case, provide the necessary spare parts as vendor support is very difficult to arrange. Near the end of the project, make sure your partner can purchase the necessary hardware or software without your involvement. This will enable them to continue providing services when your programme has completed. Choosing Free Software over proprietary software makes it easier for your partner to continue, and permits sharing of solutions.
- Do not make solutions overly complicated; keep solutions simple so the local people can implement without your ongoing input. Your help shouldn't be necessary to set up and manage any solutions.
- Project follow up is very important. Two visits per year are the minimum. By personally visiting the project you will get a feeling for the problems and burdens of deploying ICT the in developing world. This makes it possible to adjust the project's objectives.

For the three ICT projects, we can also make several conclusions.

The ICT-infrastructure project in Cuba is very successful. The services offered and their quality resemble very well what we expect from ICTs in the "more developed world". The university has it own storage system, they make use of a central identity management system, several websites are online, the staff is using a reliable e-mail system, the network and system services are monitored, ...

The ICT programme in MU was not very successful. Lack of continuous leadership in Belgium and Ethiopia is the main reason. Next to that, MU didn't have a clear vision on how to use ICT in their institution. Because of this, when the staff implemented a service it was not given the necessary attention and credit. This made the staff unmotivated. It is only in the last three years that

we have seen a real improvement. This is primarily because of continuous leadership on the Flemish side of the ICT project. In MU there are now a couple of motivated staff that were able to create some small but important changes: BMO best practices were implemented and a local e-mail system was set up. Unfortunately they will soon leave the university to get their masters degree which will again leave a knowledge gap in the ICT office.

Although the programme in JU is not yet finished and phase two has just started, we can already see its accomplishments and impact in the country. The necessary capacity was created to manage the network and systems. A big shift was made to Free Software to offer several services: a local e-mail system, several Drupal websites, web application development in the PHP Symfony framework, Samba fileservers, knowledge of Debian GNU/Linux... A direct effect of this is that JU won in 2010 and 2011 the countries ICT-cup which acknowledges the superiority of the university ICT office in the country. Outside the university the ICT office of JU was contracted to establish the ICT infrastructure in Semera University[17]. The ICT office has also provided training to other governmental bodies throughout the country. One of the most important reasons for this success was the synergy that is in place between the Flemish project leaders and the management of Jimma university. They fully understand they need to work together on a basis of mutual respect and as peers.

For myself I can also make several conclusions. Doing development cooperation in ICT has changed me in many ways. First of all, I have learned to handle unforeseen situations — *Don't Panic* — more easily and to make the best of it. Secondly, I've been exposed to different cultures which made me more tolerable. The best technical solution isn't the most sustainable solution. I've learned to give attention to the personal feelings of the team members and to be more compassionate. Thirdly, I've experienced that gentle diplomacy can accomplish more than enforcing solutions.

7 What's next?

Given the great success of the IUC programme in Cuba VLIR-UOS will start a new programme in 2013. The objective is to multiply the built capacity in other big institutions. For the programme with MU in Ethiopia there are no plans regarding ICT. The programme with JU will continue for another five years. In that programme we will focus on making the ICT services high available, the design and construction of a data center and the set up of a university wide storage system.

[17] Semera University is a new university which is located in the north-east of Ethiopia. `http://www.su.edu.et/`

The biggest challenge in Cuba will be on how to collaborate effectively with the other institutions. It's easy to talk about collaboration, but effectively doing so is something very different. Another issue we will hope to solve is the limited bandwidth between the institutions. In Ethiopia the biggest challenge for MU will be to stand on their own feet. JU will be facing the difficulty that many of their senior staff will leave the university to start their masters study. It's now up to the university and the JU-IUC programme to attract new staff and to train them. While concurrently increasing the offered services and their availability.

8 Acknowledgments

I thank VLIR-UOS for funding the projects in Cuba and Ethiopia. I'm very grateful to Prof. Seppe Deckers, programme coordinator MU-IUC, and to Prof. Luc Duchateau, programme coordinator JU-IUC. Both have been very inspirational in how they manage their projects. I express my sincere gratitude to Mike Ciavarella who shepherded this paper. He helped me change this document from a "brain dump" into a readable form. His suggestions and motivation helped me bring this task to a good end.

Teaching System Administration

Steve VanDevender
University of Oregon

Abstract

For the past twelve years I have taught a one-term college-level class introducing students to the discipline of system administration. I discuss how the class was created, the considerations that went into designing the class structure and assignments, student outcomes, how the class has evolved over time, and other observations on teaching. Links to detailed course materials and other resources are provided.

1. Introduction

In 2000 I began teaching an 8-week class titled ''Introduction to System Administration'' once each year during the summer session for the University of Oregon Computer and Information Science department. This class has been popular with students and also attended by University staff and community members who were already working as system administrators as a training opportunity.

Designing and running this class was a challenge since I initially lacked formal training or experience in teaching a college-level class. Careful thought and some experimentation went into developing a set of assignments that emphasized the combination of technical and non-technical topics suitable for undergraduates who might have no previous system administration experience.

By describing my experiences with creating and teaching this class, I also hope to inspire others with an interest in education to get involved, while also giving them some practical suggestions and a realistic idea of the challenges.

2. How did I get into this?

I attended the first System Administration Education workshop at LISA '99. At the time it looked like an interesting alternative to taking a tutorial. It turned out to be even more interesting when the attendees, a mixture of experienced educators, people interested in developing training for their workplaces, and inexperienced people like me, had widely varying views on education techniques and what should be in a system administration curriculum. Some of this came from a lack of a widely agreed-upon definition of system administration itself. Later workshops that I have attended continue to struggle with these issues, partly because the technology of system administration and its role in organizations and society continues to change rapidly.

At the workshop Mark Burgess was also passing around a draft of a system administration textbook [Burgess] he was writing. I got a chance to skim through it and was pleased that unlike many other general books on system administration, it really was structured more like a textbook than a collection of how-to topics.

Although I didn't go in to the workshop with the ambition to become an educator, I came out at least hoping to encourage my local computer science department to develop a class in system administration. When I first suggested this to the head of the department, her response was ''But that would be so . . . practical.''[1] This was not that surprising given the theoretical bent of our computer science department.

Later a colleague and I were invited by the professor teaching their operating systems class to speak to her students, and at the end of the class I asked her if the department would be interested in developing a system administration class. She said ''That would be great! Will you teach it?'' My stunned reaction was a qualified ''yes'', and that I would have to clear it with my manager. As my position description actually included teaching as an option, my manager gladly approved and I was shortly contacted by the summer session coordinator to start setting up the details of the class for the 2000 summer session.

3. Developing the class

My computer science department gave me a great deal of independence in developing the class and its content. The professor who championed the creation of

[1] She later claimed she was kidding. I don't remember that being obvious at the time.

the class offered her advice to avoid being too ambitious (which I agreed was wise), but otherwise there were few requirements or even recommendations for content nor was I required to submit material for review or approval.

The department was able to provide me a small lab of computers that could be dedicated to my students while my class was in progress, which was a major influence on my assignment design.

I had a number of obvious constraints that also affected my design:

- The class lasted only eight weeks. Although summer session classes had four or five lecture days a week, comparable to an 11-week 4-credit-hour class with three lecture days a week during the regular school year, the amount of material I could cover was limited and I would have to focus on basic, essential topics. I titled the class ''Introduction to System Administration'' to try to establish the right expectations.
- That first year there were 10 lab computers available for up to 30 registered students, so students would have to share computers. Since I wanted to emphasize themes of communication and teamwork, I could actually use this to my advantage.
- The lab computers didn't necessarily come with usable operating system installations (they were used by other classes than mine) and would have to be have installations created.
- Lab access was limited, so I wanted to make the computers remotely accessible as soon as possible. On the other hand I didn't want to put a lot of vulnerable machines on the network, so some basic security issues had to be covered early, even though security is often considered an advanced topic.
- I did not want the class to be a system administration ''boot camp'', based partly on the advice not to be too ambitions. While I wanted class work to accurately reflect real system administration in some ways, I also wanted the class to be accessible to students without previous experience and who weren't necessarily interested in system administration as a career.
- I would be responsible for all assignment grading and other management of course work. Even if I could have obtained a teaching assistant, I wasn't sure having one would be useful given the amount of extra coordination required. Therefore I had a strong incentive to keep assignment evaluation as simple as possible.

I also had several notions of teaching philosophy based on my own experience as a student.

- I had hated assignments that were poorly specified and had unclear evaluation criteria.
- I knew that different students had different preferred modes of learning, like reading, interacting with teachers and other students, and hands-on work. Therefore I provided information in each of these modes: textbook readings, lecture material and classroom discussion, and hands-on assignments.
- I wanted to create a class that provided underlying principles and a framework for knowledge, rather than just training students how to perform specific technical tasks.
- I wanted to emphasize important non-technical aspects of system administration such as effective documentation, communication, and service to a user community.

4. Class description

Complete on-line materials for my class for the years 2004 onward are available in [VanD] including the syllabus, full assignment descriptions, and lecture material. Here I will focus on the considerations that went into designing the assignments and other course content, and how the assignments have developed over time based on my experiences.

The overall sequence of my system administration class involves students working together in small groups to install and maintain computer systems and perform a variety of basic system administration tasks. I also want to allow students an opportunity to pursue a topic of their own interest and learn how to manage their own system administration projects, so the last two weeks are dedicated to a final project that student groups get to design and implement themselves with guidance from me.

4.1. Week 1: Class setup

The university's registration policies allow students to drop classes without fee penalties or add classes without special permission from the instructor through the first week of the term, so I have found that enrollment isn't really stable until the second week. I use that first week to provide general background for the class, introduce students to the lab, and to have students form groups. Usually students are good about doing this on their own so I don't have to assign

students to groups.

Because group work is required in the class I explain my expectations for how groups should work. Groups should try to divide work evenly among members as much as possible. Someone who is expecting to be absent for an extended period should let their group and me know, and work out how to handle that absence with their group. I took an idea from the LISA '99 education workshop for evaluating group contribution, and ask group members to privately send me estimates of the relative contributions of all group members with each assignment. This tends to highlight cases where someone really wasn't contributing equitably since the other group members tend to agree when that happens. However, if there does appear to be a problem, I won't assign unequal credit to group members for assignments until I can meet with the entire group and discuss the situation with them, to make sure it really is a problem with someone not contributing, and not some other sort of internal friction.

4.2. Week 2: System Installation

Before anything else, the computers used by groups need an installed operating system. I decided to try a dangerous experiment the first time I ran the class and allowed students to choose their OS distribution themselves, as long as it was freely available for educational use. I hoped that this would give people in the class indirect experience with a variety of OS distributions to see that all could be made to work. This turned out to be successful and I've continued to give students that choice, although it requires me to maintain basic knowledge of the various Linux and BSD distributions and their peculiarities. This typically results in a mixture of different Linux distributions (Ubuntu, Debian, and Fedora being the most popular recently) and one or two FreeBSD or OpenBSD installations.

However, the real goal of this assignment is to introduce students to creating repeatable, documented procedures and some basic concepts of version control. One group member does an initial installation and writes an installation document for it, and every other member of the group is required to do at least one OS installation using and revising documentation created by the one before, using RCS to record each revision to the install document. I chose RCS as the version control software because it is widely available, demonstrates basic version control concepts clearly, and, unlike some more modern version control systems such as CVS or Subversion, requires no extra infrastructure to use and can be applied to individual files as needed. Later assignments also require use of RCS for change tracking.

The material each group hands in is the RCS file containing their installation document and its revision history, which I grade based on whether each group member contributed at least one revision, and on the comprehensibility and completeness of the documentation.

4.3. Week 3: Network installation and updates

The tasks in this assignment are determined by a practical considerations for use of the lab. Limited lab hours provide students with little time or flexibility for physical access to their lab computers, but if their computers are on the network they can work on later assignments remotely at any time and place with network access. However, it's necessary to minimize the vulnerability profile of the computers before they are put on the network, since usually security updates aren't conveniently available to them until after that has been done. That means disabling all non-essential network services, which for our purposes leaves only `sshd`.

In order to determine how to disable network services on their systems, students need to know how those services are started. The first part of this assignment involves investigating how all the running processes on their computer are started by exploring their `init` configuration and scripts (or corresponding `upstart` or `systemd` configuration in more recent OSes), which usually directly indicates how an unwanted network service daemon can be disabled. This also gives students a motivation to dig in to their systems and understand how system initialization works.

Before the class started I removed the network cables for the lab computers. Once students believe they have disabled all network services other than `sshd`, I personally audit their computer to ensure that it is safe to put on the network, and give them a network cable at that time. This also gives me an opportunity to help them create and test their network configuration. The amount of effort required to ensure secure installations has decreased over time, as distributions have converged on secure-by-default policies so the number of services that are running that need to be disabled has decreased, and `sshd` is almost always installed by default or quickly installed as a package. However, some distributions also have become more dependent on network-based installation, so sometimes only a very basic installation can be made from CD or other media, and cannot be fully completed until the

computer is on the network.

After this students have to investigate which security updates are available and relevant for their OS, install them, and document which ones were installed, to assure me that they actually were. They are also required to state a security update policy that they will follow for the rest of the class, such as checking for and installing updates once a week.

Although personally auditing group computers keeps me fairly busy during this week, it has paid off in that I have had very few security problems. Those few that have occurred have been issues like weak account passwords or insecure application installations for final projects. The early emphasis on security also helps encourage students to think about it as a fundamental part of everything else they do.

4.4. Week 4: Network services

The main tasks for this assignment are to install current versions of Sendmail and Apache compiled from source. This is possibly the most technically difficult assignment of the class. When I first developed this assignment, I had certain reasons for requiring source installs. Installing software from source distributions was more common in 2000. Because everyone was using different OSes, some might have current packaged versions of these programs that could be installed with minimal effort, while others would have to build them from source anyway, so making everyone build them from source meant everyone spent a similar amount of effort.

Things have changed a lot since then. More distributions now have reasonably current prepackaged versions of Sendmail and Apache with workable default configurations. Software management using packages is now much more stable and popular. However, one reason I continue to require building these packages from source is to make students confront the issues of software installation and configuration. Installing a package with a working default configuration is usually easy. Doing from-source installation gets students to understand what package managers actually do behind the scenes. Binaries and configuration files have to be placed properly and the configuration has to be examined and appropriate customizations applied.

Since students need a C compiler and the right set of development libraries in place, most still have to also install a number of packages. Apart from Sendmail and Apache themselves, I encourage students to install packages for everything else they need.

The lecture material for this assignment, besides providing background on how software is built from source code, also explains the SMTP and HTTP protocols, and issues relating to service management. A lecture on the problem of email spam has been popular.

Evaluation of this assignment is practical and results-oriented. Groups have to demonstrate that their computer can send and receive email and serve web pages from the document root and from their own user accounts. They also have to provide the primary Sendmail and Apache configuration files, RCS logs tracking any customizations, and their installation notes.

4.5. Week 5: Account management

For this assignment, students have to create accounts for everyone in the class, including me, on their systems. The methods they may use to accomplish this goal are left unspecified, although the two most common approaches are to either have each account holder pick a login name and initial password, or to use a list of all class members to assign precreated login names and passwords.

By not dictating any preferred method of organizing account creation, I usually see a number of approaches. Sometimes this also results in creative or naive approaches that can be problematic. Once, given a preprinted slip that said my password was `$password$`, I asked aloud in class whether that group had assigned everyone in the class the same password (they gave me a rather embarrassed look — it turned out they had). Another group asked everyone to PGP-mail them a requested username and password, and helpfully included a `---PGP PRIVATE KEY---` block in the mail they sent to everyone.

The first time I handed out this assignment, I had somewhat unwisely failed to establish how I was going to evaluate it beforehand. What I ultimately decided to do was evaluate each group based on the proportion of their users who had requested accounts who were able to successfully log in to their accounts on the group's system. One of the problems with this was getting everyone to both put in the effort to request accounts on all the other computers and report whether their accounts worked. I do some cross-checking to determine whether the reason someone didn't get an account was because they didn't request it, didn't work with a group to report and resolve problems with it, or because the group responsible for creating it didn't put in the effort to get it working. Noticing that there were consistent problems each year with a few people not providing reports about the status of the accounts they requested, I eventually made a policy that any

individual who did not send me a report of which accounts they had requested and which they had working would not get credit for the assignment.

While it might be argued my chosen evaluation method could be unfair, although I am careful to watch for signs that anyone might be gaming the results, it is the best way I have been able to think of to emphasize that interaction with a user community should be based on providing good service. Based on my early experiences with this assignment I have since been careful to emphasize that the difficult part of this assignment is communicating effectively with everyone else in the class and being responsive to reported problems. Also, I advise that a simpler approach like pregenerating accounts tends to work better than a complicated approach like trying to write a web-based signup system. Since each person in the class has to interact with all the other groups to get accounts, everyone gets a sense of the tradeoffs of the different approaches to handling the assignment.

The other component of this assignment is having groups write a basic use policy for their system, as a way of encouraging students to think about issues in policy creation and enforcement. I don't expect students to write legally airtight acceptable use policies, but I do want them to understand common kinds of inappropriate computer use and what they might have to do as sysadmins to monitor and enforce compliance with policies.

4.6. Week 6: Logging, access control, scripting

This assignment gives students some basic experience with analyzing log data, experimenting with access control for their computers, and writing a simple log processing script that is to be run from cron. This is combined with lecture material on more advanced security topics.

Evaluation of this assignment is also very practical and simple. Students must provide certain specified log events, such as the pairs of Sendmail log lines showing incoming and outgoing mail being delivered, and logs showing that access control was used to permit or deny access to a service. They also have to write a simple script that mails them selected events from their system logs once a day when run via cron.

This assignment is the least complicated one. It's something of a break before the final project.

4.7. Weeks 7-8: Final project

In order to give students an opportunity to explore a system administration topic of their choice and learn some important principles of project planning, they get to design their final project themselves. I require that their design address five of the things I consider most important when doing any system administration work and planning projects:

(1) Concrete goals

(2) Specific benefits to their user community

(3) Security considerations

(4) Estimated effort of implementation and maintenance

(5) Documentation for implementation and use

A project proposal that specifies their concrete goals and outlines questions that cover the other topics is due early in week 7 to encourage some advance planning. I strongly recommend they discuss their ideas with me so I can make sure their project is realistic, since they sometimes propose extremely ambitious projects thinking that those will be better received. However, I tell everyone that projects will be evaluated based on how well they think about and address those topics, and not on how ambitious or complicated it is. The completed project, including a write-up discussing the five topics above based on the group's experience in implementing their project, is due at the end of week 8.

Examples of some notable final projects include:

- Replacing Sendmail with qmail, while preserving compatible user functionality.
- Building a honeypot system and successfully investigating the behavior of someone who broke into the honeypot environment.
- Building a NAT gateway system that provided dynamic IP allocation and firewalling for clients behind the NAT boundary.

4.8. Other elements of the class

The weekly assignments are each worth 10 points, while the final project is worth 30 in total. I originally did not grade the final project proposal separately from the final hand-in, but found it was hard to get students to cough up the proposal. Once I remembered my own student days, I realized that from a student's point of view, "if it's not worth points, it's not worth doing", made the proposal worth 10 points, and got a lot more handed in complete and on time.

Although I did not do this officially in the first few years of the class, later I held in-class discussions

for four of the assignments (week 2, 3, and 5, and the final project) each worth 5 points. This gives students an opportunity to share information with each other, and also conveniently rounds out the point total for the class to 100, making grading computation slightly easier.

In week 7 I hold an activity called "System Emergency Day!" (another idea I got from an early LISA education workshop) where I cause simulated outages to student systems by various means, such as doing `chmod 700 /`, renaming an important but non-critical system shared library, disabling some important system service at boot time, or modifying network configuration settings. All are chosen to impair system functionality without causing data loss and without requiring complicated effort to fix (at least if the correct underlying cause is found). Students then have to diagnose the problem, often with some hints from me to help keep them on track and keep the exercise from running too long. I have found that students enjoy this activity greatly, and some have even suggested doing more of this during the class, although the amount of effort required by me and the students is such that I usually spread it over two days so that I only have to track a few student groups through the exercise each day.

5. Observations and ideas for the future

Developing some 30 hours of lecture material (50 minutes a day, 4 days a week for 8 weeks) was my biggest challenge the first year, especially since initially I didn't have much intuition for how much material fit into a 50-minute class period. Having at least an idea of what needed to be covered as background for each assignment, I created an outline of everything I wanted to talk about, and got an idea of what fit as I went along. It's still an imprecise art, since student questions and in-class discussions affect how much time is available to cover each day's material, and I still often have to postpone or rearrange things as I go.

Later I expanded the outlines into readable prose that I posted as lecture notes on the class web site, to make them more usable to students as reference material or for catching up on missed lectures. I also use these as slides in class, although there is some question about whether it's useful to present that kind of prose to people while speaking. I have also tried to freshen and update the lecture material to reflect changes in current practices and my own thinking about certain system administration issues.

While I initially required Burgess's textbook [Burgess] (although I did not directly use its exercises), over time I found most students tended to be reluctant to buy it, especially as textbook prices have increased. I still recommend it and also provide information on the course web site [VanD] about a few other textbooks for reference and background, but do not require its purchase any more.

Enrollment for the class has tended to reflect overall enrollment in the computer science department. Demand for the class was highest during the first four years, when it often filled up during registration. The higher maximum class sizes in those years were an attempt to meet demand, although the year with 38 enrolled students was especially difficult.

year	enrolled	maximum	groups
2000	23	30	9
2001	23	30	9
2002	38	40	11
2003	28	30	11
2004	14	20	7
2005	14	20	7
2006	17	20	8
2007	12	20	5
2008	11	20	11
2009	14	20	6
2010	17	20	7
2011	10	20	5
2012	17	20	7

During the busiest years groups were often four people, and I got consistent feedback that many of those groups felt too big and that dividing assignment work among that many people left them without as much to do. As enrollment decreased I have recommended people form groups of two or at most three which seems to provide the best division of labor. In 2008 with only 11 students I somewhat relaxed the group work requirements, but found that it was hard to get the amount of collaboration I wanted to see when people were largely working on their own.

I haven't changed the overall structure of the class much since I created it. There have been changes to address problems students experienced with the assignments, but I've been reluctant to make major changes to a largely successful plan. However, I do feel some of the material is becoming a bit dated, and I have considered whether to try to bring in more modern topics like virtualization or configuration management. When thinking about how to integrate those, I've been reluctant to try to squeeze more into what is already a busy class, and also felt that some advanced topics

would be out of place in an introductory class, like trying to teach calculus to pre-algebra students. Adding a sequel class seems like the best way to teach those more advanced topics.

Having a dedicated lab for the class has greatly influenced its structure and the types of assignments I offer. Without a dedicated lab I would probably have to create something like a virtual lab, where students could load up their own virtual machine images so there would be no (or at least less) dedicated hardware. However, this would require restructuring many of the assignments that assume systems will be online all of the time (in particular the Sendmail/Apache and account creation assignments).

6. Conclusion

The development of system adminstration as a profession requires bringing new people into it, and having more and better opportunities for people to learn about the profession will recruit people beyond the self-identified and self-taught. Ideally there should be collegiate degree programs focused on system administration, but that still requires finding people to develop a curriculum and and create and teach the classes.

There's also some debate over how much and how well system administration can be taught. Being an effective system administrator involves independent thought and creativity, qualities that are typically thought of as hard to teach. Most system administrators are still self-taught, and most haven't reflected on how they learned important ideas and how those ideas could be taught to others.

Elizabeth Zwicky's talk from LISA 2006, ''Teaching Problem-Solving: You Can and You Should'' [Zwicky], makes a strong case that problem-solving can be taught, particularly in the context of system administration, and provides excellent advice for the would-be teacher. I'd say that a listener can generally replace ''problem-solving'' with ''system administration'' in her talk and her points remain valid. Perhaps her most important point is to believe that system administration skills can be taught, and that education experts don't find that idea controversial. Basic skills can be taught in well-understood ways, and more sophisticated skills can be developed by giving students a safe environment for experimentation and the motivation to learn through a progression of more difficult problems.

The benefits of teaching aren't just for students. It's sometimes said that you don't really understand something until you can teach it to someone else. Learning how to teach will help you understand your own profession better. Many of the skills that are essential to teaching are useful in any professional context, such as patience, good communication and interpersonal skills, and the ability to supervise and evaluate the work of others effectively.

Some of the circumstances that made it possible for me to get involved in teaching were just luck, such as having a job description that allowed for part-time teaching and finding an eager faculty sponsor to champion my idea for a class within our computer science department. Because my class is an elective, students take it because they have a genuine enthusiasm in the subject, and are much more willing to allow for the experimentation I found necessary to refine the class. However, none of this would have happened without my having the idea to try to teach in the first place. By showing you how it was possible to develop a small but successful class from scratch, I hope some of you will have the same idea and find your own opportunities.

References

[Burgess] Mark Burgess, *Principles of Network and System Administration*, 2nd ed., John Wiley & Sons Ltd, 2004.

[VanD] Steve VanDevender, ''CIS 399: Introduction to System Administration'' web site, `http://www.cs.uoregon.edu/Classes/index.php?course=cis399sysadmin`. Retrieved September 2012.

[Zwicky] Elizabeth Zwicky, ''Teaching Problem-Solving: You Can and You Should'', LISA 2006 Proceedings, `http://www.usenix.org/conference/lisa-06/teaching-problem-solving-you-can-and-you-should`. Retrieved September 2012.

Training and Professional Development in an IT Community

George William Herbert
Taos Mountain, Inc.

Abstract

This paper describes training and professional development activities at a mid-sized IT consulting firm over the last roughly 15 years. These activities have successfully engaged many of the consultants and provided significant career bonuses and advantages for the company. We present the types of activities, their effectiveness and success, and the evolution of professional development efforts over time. Many of these activities proved effective and valuable and are still in use, including annual skill reviews and development recommendations, regular organized training and discussion type events, training and materials reimbursements, and escalation support. Challenges and failures with other training and activities are described. Recommendations are made for other organizations' own professional development programs.

1. The Challenge

Training and professional development activities are a natural part of the IT community. Skills development and professional advancement are necessary for individual system administrator success and for organizational IT success.

Organized and planned professional development in an organization is a major organizational advantage for a number of reasons. It enables development within the organization rather than encouraging or forcing employees to leave to find their next more advanced role, which reduces turnover and hiring requirements. It provides employees with a morale boost, showing them that there is organizational interest in them as individuals. The very existence of an organized program is a discriminating factor which helps recruit high caliber talent in the first place.

In general, IT organizations have largely abandoned the practice of formal organized and funded professional development programs. It has fallen out of standard practice industry-wide. This paper will discuss a reasonably successful example of a program and hopefully encourage further spread of such programs in the field.

1.1. The Basic Setting

The author works for an IT consulting company headquartered in the Silicon Valley region. The company employs well over 200 technical IT staff in two major locations, in three business lines of IT consulting and outsourced services. This report focuses on the longest-standing business line and consultant base, those providing primarily technical consulting and staff augmentation roles.

These consultants are working by themselves or in small groups at a large number of geographically separate customer sites, the vast majority of which are in the greater San Francisco Bay Area.

The company has been in business for more than 20 years and has always had some level of professional development activity.

Peak consultant employment in the pre-dot-com-bust timeframe was over 800; today it is around 200. Technical job roles employed include UNIX system administration (35-45%), Windows administrator (25%), and lower and varying quantities of desktop administrators, database administrators, network administrators, project managers, IT managers, and IT SOX related experts. Skill levels range from junior to guru/architect level, though it is predominantly the equivalent of senior system administrators by respective industry standards.

This report will focus on efforts from 2001 through the current day. The author has some personal experience with activity before that but was not involved in organizing the professional development activity during those time periods. My internal employment in professional development related roles began in 2004 and continued with brief interruptions until present day.

1.2. Our Community

Our community is more than a workplace – we want to create a shared experience combining professional

work, mutual support, and social experiences. The nature of our day to day work, with consultants embedded in a large number of client workplaces, is divisive. The community has been an intentional response to create a unifying cooperative environment for the consultant workforce. The company has approached this with both social and professional development responses.

Social activities provide personal bonding and outlets, and are a major part of our organizational ethos.

Integrating professional development in the community has helped form a shared sense of professionalism and advancement, set expectations and encourage people to continue to grow and excel beyond learning directly on the job.

2. Early Efforts 1990-2000

Through the 1990s, the professional development efforts were focused in a number of areas:

- Professional development reimbursement (books, classes, certification examinations, etc)
- Industry expert presentations (roughly monthly basis)
- Peer mentoring
- Between-assignment "beach" training
- Staff management relationships
- Technical support network

Of these three efforts, the professional development reimbursement was the most used, industry expert presentations fairly well used, and peer mentoring was inconsistently used. At the time, statistics were not kept on attendance or utilization in an organized manner.

Professional development reimbursement at the time was primarily used to subsidize book purchases, prior to ebook availability and programs such as O'Reilly's Safari. A secondary contribution was training classes. A separate but related program provided reimbursement for certification exams (Sun system administrator, Cisco CCNA / CCIE, Microsoft MSCE, etc). A budget of approximately $1,000 per year per consultant was provided, though utilization was much lower.

Industry expert presentations were large, well advertised events at company HQ, which were generally open to the public as well as the internal community. The events were mostly large group, presentation-and-Q&A oriented events.

Peer mentoring was done on an ad-hoc basis.

Consultants "on the beach" between assignments were given pre-prepared learning plan / training curricula for self paced learning in the HQ office's training lab.

The company also established a clearly defined role known at the time as "Staff Manager", who were outreach HR staff who engaged with the field consultants (ideally at least once a month) to follow up on both work success and professional development efforts.

Finally, the company established an internal technical support escalation process. This was somewhat ad-hoc initially, but consultants could contact their staff managers or headquarters and be connected rapidly to other consultants who were pre-identified as subject matter experts in various areas. Assistance was made available in case of emergency, or even a consultant who found themselves in over their heads on a particular technology and needed someone to help them through a problem.

2.1. Early results

Professional development reimbursement was widely seen as highly useful. Most consultants were able to fit book purchases of the time within the available budget and occasionally stretch to cover part of a relevant class.

Industry expert presentations were well attended (~ 10-15% of total consultant base) and well liked. Drawbacks were that the events were spaced out significantly (monthly) and not as interactive as attendees might have liked, though Q&A sessions were largely effective.

Peer mentoring was inconsistently effective.

Self-paced beach learning programs were found to be difficult to keep current and relevant, hard to track progress on, and eventually determined to be somewhat ineffective.

Staff manager roles were found to be very effective, both for normal HR and management issues and for encouraging professional development.

The escalation support mechanism was found to be highly effective and useful. Backing up the consultants improved both client success and consultant professionalism and willingness to push their own technical comfort envelope.

3. Shaking things up – 2001-2007

After the dot-com bust, the industry was significantly unsettled. As the consulting company recovered from that downturn, one of its priorities was to rebuild the professional development activity and make it more effective and engaging to the consultant community. The consultant base was smaller than it had been, and the customer interests had moved to more senior level skillsets over time. Several key changes were attempted over time.

One change was that the existing role of senior technical consultants was expanded out from primarily interviewing and consulting to include more involvement in professional development. More engaged contact between the most senior consultants and the rest of the consultant base was seen as an easy and obvious win. This included several approaches, including one-on-one mentoring programs, group lunches where one or two senior consultants had a chance to meet and get feedback on professional development activities from mid-sized (8-12) groups of field consultants, more instructor type training programs, etc.

The roughly monthly expert presentations were changed (largely) to a weekly schedule, more interactive and smaller group activity now known as "Office Hours". This allowed for more interactive, round table type discussions as well as presentation type meetings with an expert speaker. Many discussions were led by those senior consultants.

Regular organized training classes were begun for more advanced concepts not amenable to short one-night sessions. These were usually 1-2 nights per week, for up to 4-6 weeks in a row, though other schedules (5 nights in the same week, both days in a weekend) were experimented with as well to accommodate instructor and consultant schedules. As noted above, senior consultants were the usual instructors.

Finally, as part of the annual review process, technical re-interviews of the consultants were made more standard and consistent, to establish new snapshots of people's skill development and professional level over time. Additionally, discussing professional development with the consultant and making recommendations (or for more aggressive consultants, validating that their own learning efforts were appropriate) was made a standard part of the process.

Prior programs such as professional development reimbursement, staff management support role, and the escalation support program were continued and reinforced.

3.1. Shakeup results

Strengthening and engaging the senior consultants was generally successful. More active mentoring programs were not as effective; consultant engagement turned out to be inconsistent and generally poor. Intermittent informal meetings to collect feedback worked somewhat well, but were eventually found to be logistically difficult to coordinate on a regular basis.

The shift to weekly more interactive presentations and round table discussions were initially highly successful. At times 20% or more of the total consultant base showed up for the weekly meetings, and the more interactive ones had very good participation from the audience. They helped foster an effective sense of community involvement.

Training classes were found to be highly effective. Though longer classes experience a predictable falloff in attendance, ultimate completion rates were good and knowledge transfer was found to be effective. Drawbacks were time commitment by the instructor and the students, but the classes were found to be valuable anyways.

Annual skills reassessments / reinterviews were found to be both a valuable data collection tool over time and a valuable venue for one-on-one feedback on professional development and training progress from senior technical consultants to the individual consultants. Establishing these as a standard, annual process for each consultant was highly valuable.

Unfortunately, quantitative records from this time period are missing or were insufficient for statistical results analysis.

4. Evolution in action – 2008 to current

In general, successful programs were maintained at that point. However, the world changes and shifts around

us, and success levels evolved for the varying programs.

The 2008-2009 timeframe brought significant turmoil with the major economic downturn. This affected this community and company as well as most other employers, though not as badly as the dot-com bust.

We anecdotally observed that a number of consultants became more interested in professional development during the downturn, focusing on their employability. Lack of detailed training records from 2008 and earlier has prevented statistical analysis of this effect, though we may be seeing some of the return to a more long-term normal level in 2009 to 2010 event attendance trends.

One change was that professional development reimbursement benefits were temporarily suspended during early 2009, along with most of the training budget. The informal office hours were maintained, but much other activity went briefly on hiatus.

During this time period, from 2008 through 2010, the company experimented with a vendor-provided computer-based training e-learning system. Adoption proved problematic and the training was not well maintained as technology and skills evolved.

Another change was that the company headquarters moved approximately 3 miles, from right next to a major freeway interchange to about 3 miles from the next exit up the freeway. This had a subtle but eventually significant negative effect on attendance of in-person training and office hours activity. With consultants spread over a large area (tens of miles) their ability to attend in-person events after work hours, and through evening traffic delays, has been a significant and increasing issue.

Also in this time period was the eventual adoption of real time internet video broadcast of many or most events so that consultants could follow the training or interactive events remotely. Though video conferencing and time-shifting proved somewhat useful, adoption rates have been only moderate, as ease of access to the recordings and broadcasts is impeding users.

One major gap is that the existing web presence for the professional development community has been identified as deficient. This makes it hard for consultants to locate schedules, information, and recordings of classes and events that have already happened.

Another change is that the average age of the consultant employees has been advancing steadily over time. Professional development has been fighting more and more for time with parent and family activities in particular.

A change to offer Safari ebook service subscriptions to any consultants who wanted them also happened during this time period and has been widely used by consultants who are actively pursuing personal professional development.

4.1. Late-breaking developments

In late summer 2012, after initial submission of this paper, review of the lower trending office hours attendance led to a refocusing of the professional development program away from that type of event and towards more distinct training class type events, even if they are shorter classes with only 1-2 class meetings.

This author and others are working to analyze what long-term trends may have affected the attendance and interest in the shorter "office hours" type events. Surveys of the consultant community and other analysis are ongoing.

5. Ultimate results

The professional development program is ongoing and actively engaging a significant percentage of the consulting employees. The engaged consultant base has anecdotally shown a consistently higher rate of professional advancement and learning compared to the unengaged consultant base, though it's not clear if that is a causal result or merely correlated.

6. Lessons learned and analysis

It is possible to build an organized professional development program in a large modern IT environment. It appears to energize at least the employees who are already inclined to learn and promote and advance themselves.

A number of things have failed in the professional development and training programs over time, or have evolved into less useful or non useful states. A willingness to review and examine success of projects and methods, and regularly invent new ones to fill in gaps, has been necessary.

It has been difficult to build an integrated ongoing statistical efforts / results database for the various programs, and quantitative effectiveness of the programs has been difficult to nail down. Efforts to build that are ongoing.

Sufficient anecdotal feedback has been received over time that the programs were felt to be worthwhile and a strong plus for the organization.

6.1. Measure as you go

For many years, performance metrics were not collected in an organized fashion as the program progressed. Though event attendance was often recorded dating back into the pre-2001 timeframe, those records were not organized and reliably tracked until 2009. Spending on total training related costs was available going back to the advent of the current financial records system, but subcategories of professional development reimbursement, training event logistics, course preparation work were not separated out in the financial records. Introspective research into scattered and inconsistent records has proven impractical.

Data that is not collected is extremely hard to recreate later; data that is not organized may be effectively lost by the time you go looking for it. Measure, and record what you are measuring.

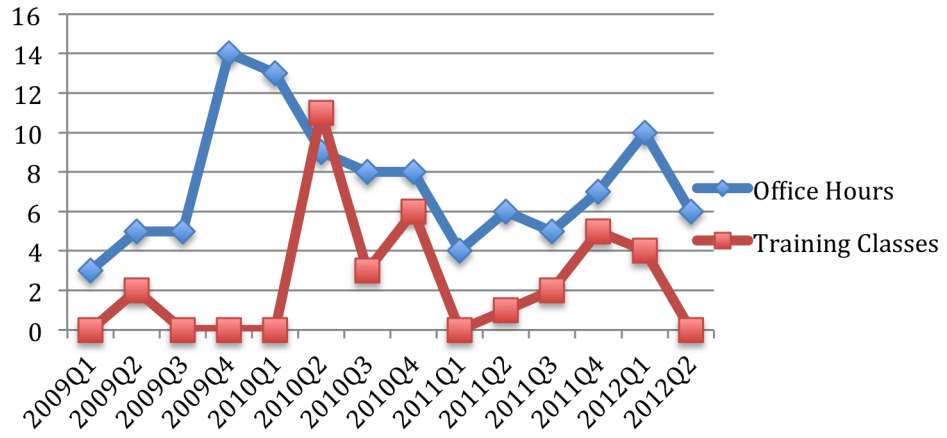
6.2. Tracking and interpreting event attendance

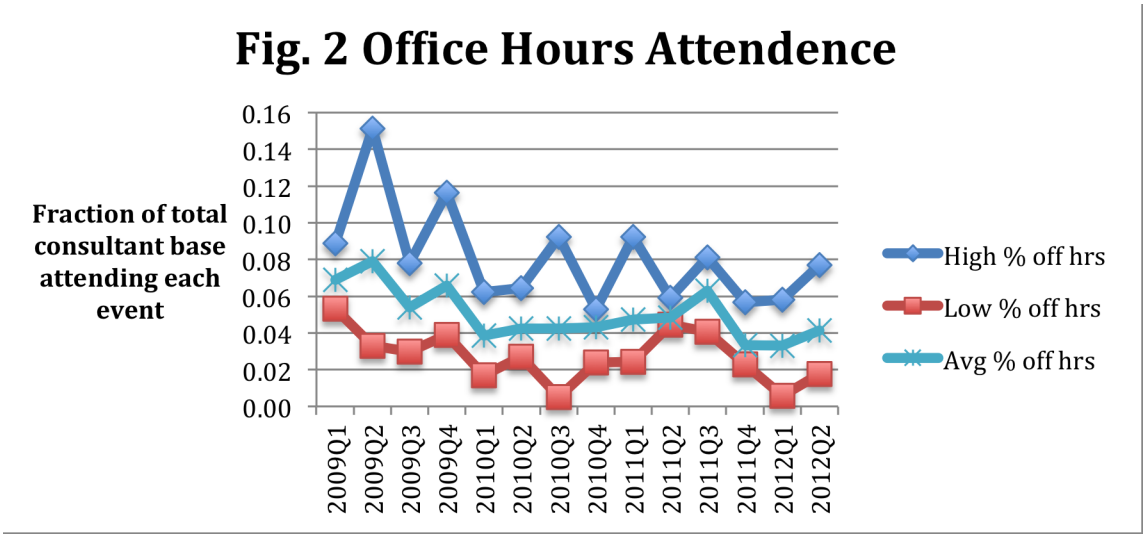
Event attendance data from 2009 to the present day (14 quarters) was reasonably complete, though individuals' personal activity was not analyzed in detail to date. In overall statistical terms, 137 events were held over those 14 quarters, in separate categories of "office hours" type one-session less formal meetings and training classes with more formal lesson plans and usually multiple class sessions. 103 office hours events were held, with 837 total attendees. 34 training class sessions were held with 305 total attendees.

Fig. 1 shows the total of each type of event, per quarter over the time period.

Fig.2 shows the average, minimum, and maximum attendance of each office hours event during the respective quarters, normalized to the total consultant employees during that quarter.

Fig. 1 Training and Office Hours Events





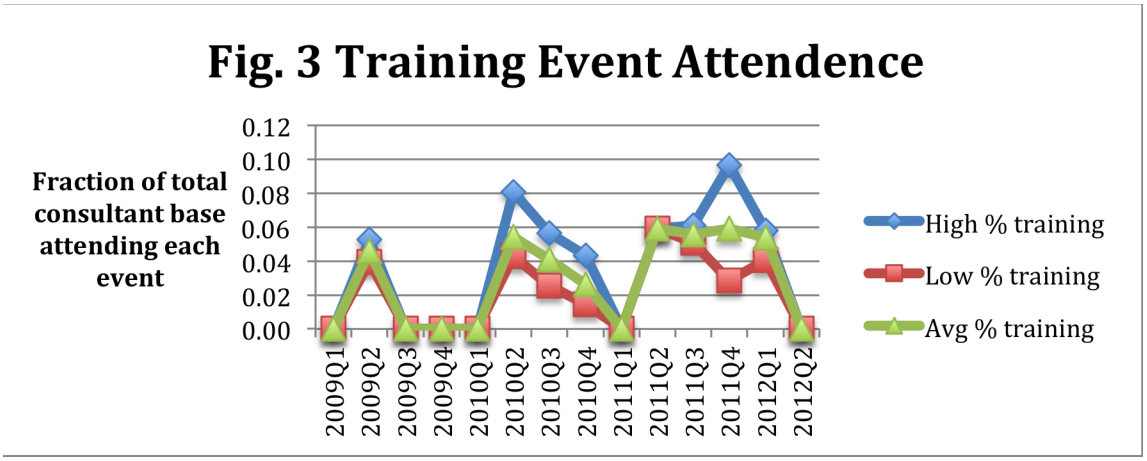


Fig. 3 shows the training event attendance during the respective quarters, again average, minimum, and maximum values normalized to total consultant employees.

The attendance data require some careful review, as they are both noisy and there were quarters in which we did not offer any training classes, either due to conflicts in schedule or due to intentional hiatuses

The beginning of the data overlap with the 2008-2009 economic downturn.

The office move described in section 4 of the paper is evident in the per-event office hours attendance drop-off from the beginning of 2009 (right after the move) through the beginning of 2010. The count of events per quarter was also significantly decreased that year (2008 generally resembled 2010 numbers, though detailed records are not identified at this time).

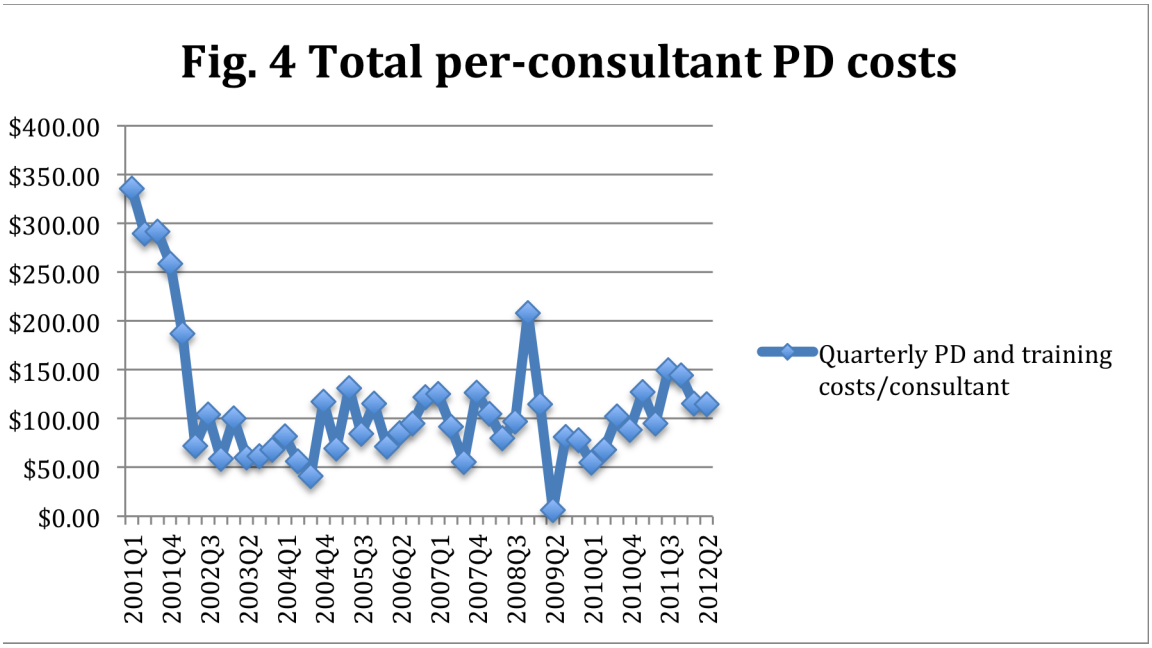
Due to both the move and the economic downturn, only 2 training class events were held over a 12 month period. The number of office hours events was also depressed during the first 3 quarters of 2009.

For informational purposes, the list of training and office hours event topics presented during 2010 is provided as Section 9 of this paper. This hopefully will provide ideas for others looking at their own diverse professional development programs.

6.3 Professional Development Budgeting

Fig. 4 shows the total quarterly per-consultant training and professional development spending from 2001 through the present day. Easily visible events include the middle of the dot-com bust, with employees rushing to spend professional development benefits before layoffs, and the temporary training and PD hiatus in early 2009 due to the recent economic downturn.

Visible in the data are a low-active level from 2002 through early 2004, with an average of around $75/quarter, followed by a more active period from 2004-2008 at an average of around $100/quarter (and an anomalous quarter whose spike's cause is unclear). Following the pause in 2009, a steady growth now flattening out at around $125/quarter is evident.



These costs include both training and professional development. Breakdowns of those were not available within the accounting data easily available.

Total spending per consultant of less than $500/year has proven to be enough for a functional professional development program.

6.4 Lack of Professional Development reimbursement utilization

The employee base has a base individual professional development reimbursement of $250 per quarter available for books and training and other professional development expenses. Factoring in other costs and training costs, the average utilization of this reimbursement appears to be a very small fraction of the available benefit.

From the 1990s, per-employee reimbursement was used more actively on book purchases. Now, with the Safari service available to staff on request, this may have reduced the demand for individual PD reimbursement benefits. However, the benefit is still provided for physical book purchases, books not on Safari, or for other training materials or courses.

6.5. Overall benefits of the Professional Development program

It is always desirable to have an easily and clearly measurable result of significant efforts. That has eluded our professional development program; results have always proved difficult to quantify.

Qualitatively, we believe the program has been of significant benefit to large segments of the community.

Approximately half of the consultants show up to one or more training events over the course of a year; over 14 quarters, total training attendance was 7 times the employee headcount, and if half the employees participated at least once then average attendance for active participants was about once per quarter.

Consultants who are active with the professional development programs anecdotally cite those programs as reasons for staying employed with the company, and as advantages for their career development and job satisfaction.

6.6 Gaps in our methodology

Our organization collected this data in the course of normal business operations and did not have a data-analytics driven quantitative approach to program results at the time.

The largest gap we are aware of is that while we measure professional development results in microscale – individual staff skill reviews, annual reviews, and professional and HR interactions – we have not consistently applied good quantitative methods to look at it in wide scale across the consultant base. We have intermittently tried surveys of the consultant base and working with our HR staff to quantitatively feed development results back into the professional development process. None of these have given satisfactory results over the long term.

We have overall professional development records for each consultant associated with our annual technical review interview / skills assessment process. Integrating that data with the participation data for the training and other professional development activities is an obvious next step for us, but has not been accomplished in time for this paper.

6.7 Open Problems

Our largest identified open problem with the program is in engaging the fraction of professional staff who have not yet chosen to participate in a fundamentally optional, personal time commitment required professional development program.

We have intermittently had moderate problems effectively communicating the events to some consultants; some of them tune out or ignore email notifications, and we are not always successful at leveraging face-to-face HR meetings with consultants to encourage attendance, though it is a goal.

Many more consultants, while aware of the program, either find it too inconvenient to participate or do not have enough career and professional development drive to value participating.

Inconvenience climbed as consultant average age has climbed out of the 20s, now in the late 30s. The fraction of consultants with families and in particular small children has noticeably increased over the last 10 years, though we have not quantitatively analyzed that yet.

We are also aware that our office move in 2009 introduced significant perceived additional inconvenience, adding another 10 minutes to travel time for most consultants.

Those staff who are content and not driven to actively learn and promote themselves are an ongoing motivational problem which is beyond the scope of this paper.

7. Applicability and recommendations

Our organization is not yours; our community is not yours.

7.1. Differences with typical organizations

The example organization is a consulting and staff augmentation organization, not a monolithic internal team. A number of approaches were oriented on handling a dispersed, harder to unify team. Many (most) organizations will find internal communications easier.

The example organization has tended towards older, more experienced consultants over time as the client base demands have evolved. Average IT organizations have a more normalized junior/intermediate/senior staff distribution. This both increases target opportunities for internal training and advancement of bright junior and intermediate staff and means that more range of training may be required. More junior and intermediate staff, and staff less highly selected for consulting professional skills, may be somewhat harder to interest and engage in active professional development training programs.

The example organization has more than 150 technical consultants in its main region. Most IT teams have smaller total staff, and larger teams tend to be dispersed in larger organizations. Our organization has a critical mass of senior consultants to sustain active training and professional development over long time periods. Yours may not, and available resources for programs like this may be more limited and precious.

7.2 Annual technical reviews

An annual technical skills review program is likely to prove valuable. The sample company has established a repeatable technical evaluation process for interviews and annual reviews; other organizations without that process may find it somewhat more difficult to assess skill improvements. Even if it is harder to do, talking to people regularly about what they've done and what they've learned, writing it down, and making suggestions for upcoming training and development helps engage the staff in development processes.

Our approach focuses on skill gains, professional capabilities advancement, experience, and recommendations for personal and company-supported professional development activity for the next year. Our skills assess-

ment is based on the same proprietary skills assessment process used in hiring; you will likely want to look at open source IT professional skill assessment and analysis methodologies.

7.3. General recommendations

Our program has been successful at some things and not at others. Understanding what's working and continuing to do it and explore more options work, over long time periods.

It is important to note in starting that well supported professional development programs will not take below-average employees satisfied with their current level and turn them into super-learners. However, they can engage more of the employee base, and provide more successful outcomes for employees who are engaged in the process. Every bit helps.

The first core recommendation is to organizationally value professional development, communicate that, and support it. The rest follows from that core value.

Some aspects of the sample company's practices should translate easily and successfully to any organization. Defining and providing a personally managed professional development reimbursement budget is still useful. Even if average utilization is low, the control and empowerment employees feel with this available have anecdotally been valued.

Programs such as the Safari books program, either as a common team benefit or as an individually purchased benefit from professional development reimbursement funds, are extremely valuable now. The ability to participate in conferences, paid training, and certification programs are significant benefits to actively self-advancing employees.

Identifying above average communications skills senior employees and empowering them and making them responsible to some degree for the learning and success of the rest of the employees is a powerful tool. If they are engaged in the process they can spread that energy and draw other more junior employees along with them. Building a community, publicly identifying and crediting community leaders, and encouraging employees to step up over time are all constructive actions.

Most organizations have an informal technical escalation process, within local peer groups. Establishing a formal one and identifying local or remote escalation resources can help reinforce employees' willingness to take technical and learning risks and step up more actively.

Establishing a regular seminar, training, or open discussion session time and format is a powerful tool and effectively engages many employees. Making it a lunch-provided midday program once a week has worked effectively in several organizations. An after-hours program will have less attendance but is still valuable.

For any program with activities, be they discussions or formalized training classes, strongly consider recording or webcasting the events, and making the recordings available on a single centralized intranet site. Delayed participation is still very valuable and can easily double the end value to the organization. Make sure that the site is sufficiently user-friendly to be usable and effective, however.

7.4. Measuring your Professional Development success

As an expansion on the annual review point, establishing a program to quantitatively measure engagement in and apparent results of the various programs in your organization is a valuable focus tool. Knowing what is not just popular, but was used most by people compared to their year-over-year advancement, is important to fine-tune your focus and efforts. Every time we measured something statistically or numerically we learned something. Every time we found new ways to poll our community members about the community and professional development, we learned something.

With that said, our organization – with a central staff dedicated over decades to this type of community and consultant base development – has still had problems both figuring out how to measure data and how to quantify and track results. This remains a hard problem.

7.5. Your IT Community

Every community is different.

Communities require critical mass of participants to sustain them. We have more than enough people who are at least socially involved (150) but have the problem that their jobs are spread across a 100-mile-wide area.

Staff at smaller organizations may find open organizations, such as local LISA chapters, Linux User Groups, various new IT group meetups, and other communities are more effective than attempting to create your own

local community. We included professional development and community social activity in our internal organization; you may separate out some professional development activities to internal-only within your smaller organization, while connecting socially and for other professional development with larger public communities.

8. Additional research

Further data mining is in progress into our historical records.

We would like to broaden event attendance data over more of our history of this effort, though records status is currently unclear.

It would clearly be valuable to individually compare personal success at our company and employee happiness with participation in professional development activities. Initial and evolving technical skill / seniority level data is already collected, and our HR / consultant management teams collect significant consultant feedback, but it has never been reviewed or categorized to derive success data for the professional development program. Employee retention and employment duration, client project success, employee happiness can be correlated with seniority and participation in professional development activities.

This is a much more involved statistical and data analysis problem and will hopefully be the subject of a future paper.

9. Sample training / office hours topics

These topics were used for either training courses or office hours presentations and discussions during 2010.

Facilities Management
Office Hours Windows Hyper V
Sun Secure Global Desktop Software Presentation
Networking
Networking ASA & PIX
Disaster Recovery Planning
Business Systems Analyst Overview
Windows 2008 AD Sychronizer
Beyong Basic networking, Subnetting and Other Mysteries
Project Management for Real World Projects
Intro to puppet & Config Management
Windows 2008 R2 ISNS Features
Intro to MPLS
Windows Deployment Services
Linux Kickstart
Visio
Database Administration for SysAdmins
Intro to Linux
Windows 2008 Foundation Course
PMP Course Review
Management Consulting Practice Meet & Greet
Windows 2008 GPO Class
Enterprise Storage/SAN/NAS
GPO Class Day 2
Book Club
Practice Leader Networking Event
Excel for System/Network Administration
Completing a Domain Rename
Linux Fundamentals
VMWorld conference report
Open Source Systems
Visual Basic Scripting/Intro to Powershell
NAC Deployment
Windows 7 Deployment
Interview Do's & Don'ts
Configuration Management
The Evolution of the Network Engineer Role
Intro to SQL and Database Design Using MySQL
Windows Certifications
Juniper SRX Firewalls
Drobo Products

10. Acknowledgements

Data and research for this paper were conducted by Stacy Ernst and Susan Nguyen. Additional feedback and comments were provided by Eric Su and Stephanie Van Thillo. The training and professional development activities described here were the work of dozens of current and former staff, all of whose participation is acknowledged and appreciated.

Extensible Monitoring with Nagios and Messaging Middleware

Jonathan Reams <jreams@columbia.edu>

CUIT Systems Engineering, Columbia University

Abstract

Monitoring is a core function of systems administration, and is primarily a problem of communication – a good monitoring tool communicates with users about problems, and communicates with hosts and software to take remedial action. The better it communicates, the greater the confidence administrators will have in its view of their environment. Nagios has been a leading open-source monitoring solution for over a decade, but in that time, the way it gets data in and out of its scheduling engine hasn't changed. As applications are written to extend Nagios, each one has to figure out its own way of getting data out of the Nagios core process. This paper explores the use of messaging middleware, in an open-source project called NagMQ, as a way to provide a common interface for Nagios that can be easily utilized by a variety of applications.

1 Introduction

Monitoring is a core function of systems administration, and is primarily a problem of communication – a good monitoring tool communicates with users about problems, and communicates with hosts and software to take remedial action. The better it communicates, the greater the confidence administrators will have in its view of their environment.

Communication is complicated. Marshaling and serializing data is complicated. Ensuring atomicity is complicated. Above all, it is very complicated to do all of this quickly for a large number of connections, but as technology designed to make communication between computers and people easier has advanced, monitoring systems retain the communication methods they were initially designed with a decade ago.

Nagios has been a leading open-source monitoring solution for over a decade, but in that time, the way it gets data in and out of its scheduling engine hasn't changed. As applications are written to extend Nagios, each one has to figure out its own way of getting data out of the Nagios core process. This paper explores the use of messaging middleware, in an open-source project called NagMQ, as a way to provide a common interface for Nagios that can be easily utilized by a variety of applications.

By creating a better interface for moving data into and out of Nagios, we make it easier to solve some of the issues of monitoring very large numbers of hosts and services. In addition to the core of the project, which links Nagios to the outside world, we explore several use cases for NagMQ in a large distributed environment. The project will efficiently distribute the workload of monitoring across dedicated monitoring hosts and to the edge, implement an active-passive failover cluster, and enable easy implementation of alternate user interfaces.

1.1 Background on Nagios

At its core, Nagios is an event loop that runs through a list of checks for a set of services and hosts, executes them, and then executes further commands to notify or take corrective actions when the output of the check changes from the last time it was run. Although some special kinds of checks are run synchronously in the event loop, most are run asynchronously with a special Nagios result collector process that collects the results of the checks and communicates them back to the event loop via the file system. Because each check also has a corresponding collector process, and because Nagios is configured to fork twice per check by default, each check may result in 3 different processes. As checks execute, a small text file is written out by the collector process that contains the textual and numerical output as well as some statistical and timing information for the executed check. The master Nagios process periodically reads all these text files, parses the output of the checks, takes any actions (like notifying the owner of the check or taking corrective action), and schedules the next execution of the check as necessary.

In addition to periodically reading in check results, the master Nagios process also writes out its internal state for use by the user interface. The web interface reads both the configuration files for Nagios and the status data file whenever it needs to display any dynamic information to the user, regardless of how much or what kind of data the user is requesting. These status files can also grow to be very large in large installations of

Nagios – in the production environment described by this paper, the status, cached configuration, and retention data files add up to 54 megabytes that must be read into memory and parsed whenever the user requests some information.

Input into Nagios at runtime is possible through a named FIFO pipe that is opened by Nagios and accessible to the web interface and other addons. Passive check results and commands are formatted into a semicolon delimited line of text and written into the pipe. Separate threads in the Nagios daemon execute any commands coming in from the pipe immediately, and write out passive check results to the file system for later processing by the check result reaper. Because the pipe is just a UNIX FIFO, it has a maximum capacity before clients are blocked (on Linux, the capacity of a FIFO is 64KB, since Linux v.2.6.11).

An optional pluggable event broker (NEB), introduced in Nagios 2, allows developers to register callbacks for most events in the master Nagios process (Galstad, Nagios 2.0b1 Available, 2004). Despite having direct access to the raw data as it moved through Nagios, NEB plugins couldn't override the default execution behavior until Nagios 3.2.2 (Galstad, Nagios Core 3.2.2 Released, 2010). All the Nagios add-on projects mentioned in this paper use a NEB plugin to facilitate communication between Nagios and their application.

The enemy of Nagios is check latency, which is the difference between the actual time a check is executed and the time it was supposed to be executed. Because Nagios is an event loop running in a single thread, latency can rise whenever checks start to fail, and more synchronous host checks need to be run to check dependencies between checks, when checks need to be re-executed to determine whether a problem is transient (to determine the "hard state"), or when notifications need to be sent to a user. As latency goes up, the confidence that Nagios will actually detect problems goes down.

As we shall see, making as many actions as possible in the event loop asynchronous – and thereby reducing the amount of time Nagios spends taking each action – is critical to good performance of the system overall. Design choices – like using the file system, which is backed by disks that are slower than memory, as the transport for check results and for getting data out to front-end interfaces – create challenges to making Nagios scale effectively. To minimize impact on performance, many NEB plugins simply hand off the event to some external daemon to be handled asynchronously.

1.2 Background on Messaging

The purpose of messaging middleware is to provide a framework for the complicated work of communication between software, so that the application can focus on the application, and not moving data. One of the most mature examples of modern messaging middleware is JMS (Java Messaging Services). JMS isn't a single protocol or product; it's an API for Java that supports a variety of messaging patterns. Although there are many products available that support or extend JMS, they are often incompatible with each other. Some examples of JMS-compatible message brokers are Apache's ActiveMQ and Qpid, JBoss Messaging, and RabbitMQ (which also supports AMQP).

When JPMorgan and iMatix developed the Advanced Message Queuing Protocol (AMQP) in 2006, the goal was to create a common open source and standardized protocol for messaging, drawing heavily from JMS. (O'Hara, May 2007) AMQP provided a messaging system that was relatively easy to access via a variety of programming languages and operating systems, and since AMQP is a protocol and standard rather than a product, there are a number of implementations of the broker and client libraries – making it easy to adopt for a variety of applications. RabbitMQ and OpenAMQP are examples of AMQP message brokers.

AMQP, JMS, and other related messaging systems – such as ActiveMQ's support for the STOMP protocol – rely on a central broker that sits between all clients that want to send messages and passes messages between them (iMatix, 2008); the broker is not just a proxy for information, it sits at the center of the application, and a lot of application logic is put into the way messages move through the broker. This design introduces a lot of overhead. Where before there may have been a library that supported communication, now there is a whole infrastructure. The design of Nagios already puts the Nagios daemon at the center of your monitoring with a number of applications and add-ons hanging off of it; adding a new message broker to coordinate communication confuses the picture by putting Nagios core as a leaf instead of a root.

ZeroMQ is a brokerless messaging library written by iMatix – they stopped developing for AMQP and its reference implementation OpenAMQP to work on ZeroMQ (Pieter Hintjens, 2010). Unlike AMQP/ActiveMQ, it does not borrow any semantics from JMS, and acts more like a BSD sockets library – in fact the API makes a lot of effort to be an almost drop-in replacement for standard socket calls. Unlike

JMS libraries, it does not do any data marshaling – message payloads are treated as opaque binary blobs. (Hintjens, Welcome from AMQP, 2012) (Hintjens, ZeroMQ Guide) The one exception is that its subscription socket will do prefix-matching on incoming messages, and discard any messages that do not match any registered subscriptions.

1.3 Related Work

There are a number of projects available for making Nagios scale to large installations and for making it easier to get information into and out of it. We will discuss four different projects that provide similar functionality for Nagios.

1.3.1 DNX

DNX is old enough that the ability to override checks from a NEB plugin in Nagios comes from a patch for DNX. The DNX NEB plugin has a number of internal threads for communicating with clients. When the DNX worker process starts, it connects to the "head node" running Nagios and waits for check jobs. Jobs are executed by the head nodes, and the results are sent back to the DNX NEB for processing. Check results are added to a list internal to DNX that gets merged with the Nagios check result list when the reaper event occurs. (Augustine, 2007) DNX has not had a new release since April 2010, and may no longer be under development.

1.3.2 Mod_gearman

Mod_gearman is another distributed check execution system based on the gearman job distribution framework. In addition to distributing load between different monitoring hosts, the use of gearman allows for more advanced distribution patterns. For example, checks for a remote site can be run at the remote site and results proxied back through gearman, reducing the number of firewall exceptions needed to get the data in and out. Mod_gearman also allows you to duplicate the check results out to a passive Nagios host, providing state replication for high availability. There are also a number of advanced options for selecting which checks get executed where, such as affinity for hostgroups and dedicated workers for hosts, services, and event handlers. (ConSol* Labs) As we shall see in the performance testing, mod_gearman runs check jobs synchronously in its workers – so its performance overall requires that there be a large pool of worker processes and that check duration be very short.

1.3.3 NDOutils

NDOutils is a commonly used add-on to Nagios that is used for getting configuration and state information out of Nagios for insertion into a database or other 3rd party applications. It consists of a NEB module (NDOmod), which serializes Nagios data and transmits it over a socket to files and sockets, and two daemons which process that data into a database; NDO2db inserts configuration sent by NDOmod into a database and LOG2db inserts historical Nagios log information into the database. (Galstad, NDOUtils Documentation, 2007)

1.3.4 Merlin

The Merlin project was started to "create an easy way to set up distributed Nagios installations, allowing Nagios processes to exchange information directly as an alternative to the standard Nagios way using [the Nagios command pipe]." (Ericsson, Merlin) As well as moving data between Nagios processes, Merlin stores its data in a database and serves as the backend of the Ninja frontend project.

Like NagMQ, Merlin is focused on communication with Nagios, with different instances of Nagios taking on different roles – a "NOC", which receives check results and sends configuration data; a "poller", that sends check results to NOC processes; and a "peer", that checks hosts and services redundantly for high availability. (Ericsson, Merlin). It is focused on instances of Nagios communicating with each other for load balancing and fail-over, whereas NagMQ is more general, but in many ways it has the same purpose as NagMQ. The main difference between NagMQ and Merlin, in terms of how they are deployed and used, is that NagMQ depends only on having the ZeroMQ libraries installed – whereas Merlin requires a database and has a daemon running outside of Nagios.

2.1 Architecture

The center of the NagMQ project is an event broker plugin that provides interfaces into Nagios via ZeroMQ message queues. The data from Nagios is serialized into JSON objects/arrays before it is put on the wire – JSON was chosen because it is a relatively well-defined data format with relatively quick parsers for most languages. For the most part, the keys in NagMQ objects correspond to the name of the corresponding value in a Nagios data structure. The three interfaces of NagMQ are:

1. Publisher for events as they happen in the Nagios event loop

2. Command and state update interface

3. State and configuration information query interface

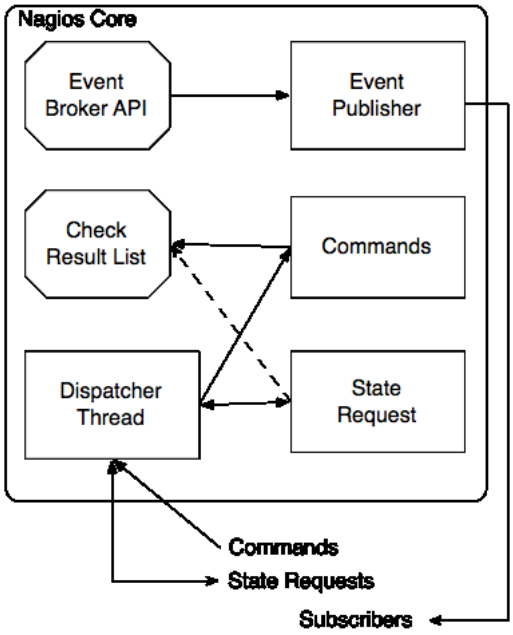


Diagram of the NagMQ NEB internals. The Event Broker API,Check Result List, and Event Publisher all run inside the main Nagios thread

2.1.1 Event Publisher

For the event publisher, messages consist of a prefix that includes the type of message and the relevant host name and service description, if one exists, and the actual message JSON payload. Consumers must subscribe to at least one prefix in order to receive messages. If there are no connected subscribers, the JSON payload is still generated, but is discarded by the message queue.

If a connected client wanted to receive information about all processed service checks for a given host, it would subscribe to the "service_check_processed hostname" prefix – and all other messages would be discarded. In ZeroMQ 3.x, these subscriptions are forwarded to the publisher and only those messages will be sent over the wire, whereas in ZeroMQ 2.x, the filtering is done at the subscriber – this is not an issue except as a consideration of the amount of bandwidth needed to transmit all of Nagios' events. (Sustrik M. , 2011)

Because the actual transmission of messages occurs in a dedicated I/O thread, only generating the JSON actually occurs inside the event-loop, and each call to the broker is essentially asynchronous.

The publisher interface can also be configured to override the default behavior of service checks, event handlers, notifications, and, with a patch, asynchronous host checks, so that they can be executed outside the Nagios daemon by an application subscribing to the message queue – this is used for implementing distributed check execution.

1	`service_check_initiate malanga testexec`
2	`{ "host_name": "malanga", "service_description": "testexec", "check_type": 0, "check_options": 0, "scheduled_check": 1, "reschedule_check": 1, "current_attempt": 1, "max_attempts": 2, "state": 0, "last_state": 0, "last_hard_state": 0, "last_check": 1347985062, "last_state_change": 1343674250, "latency": 0.16800, "timeout": 60, "type": "service_check_initiate", "command_name": "check_by_sr", "command_args": "test!exitcode=0", "command_line": "/opt/local/bin/check_by_sr -H malanga -c test -o \"exitcode=0\"", "has_been_checked": true, "check_interval": 3, "retry_interval": 2, "accept_passive_checks": true, "timestamp": { "tv_sec": 1347985242, "tv_usec": 169135 } }`

Example messages sent by publisher to indicate the beginning of a service check; first column is message part number, second is text assent.

2.1.2 Command Interface

Messages sent to the command interface have no prefix and should be a single JSON object containing the type of command or update to be processed, information about the target of the command (e.g. the host name and service description), and any other parameters.

This interface also has its own internal message queues to insert check results into the reaper queue efficiently. The command interface may have multiple worker threads. As check results are received by command worker threads, they are sent to a queue where they accumulate until the reaper event is started and each message is popped off the queue and appended to the list of results to process. This means that check results sent to NagMQ never touch the file system, reducing a potential I/O bottleneck.

In addition to check results, downtimes, comments, acknowledgements, and commands, the command interface will also accept state information for all hosts/services in order to facilitate state synchronization between Nagios hosts at start-up. The interfaces for receiving check results and other state changes expect the same output format as messages published by the event publisher – so the results of one instance of NagMQ can be fed into another for state synchronization.

This is the only interface that does not send any response back to the user, but unlike a publish/subscribe interface, sending a message to this interface will block until the interface is running and able to accept new messages – although messages may be accepted and queued instead of being processed immediately, so successfully sending a message is not a guarantee that it has been processed.

2.1.3 State Request

In addition to listing all hosts, services, contacts, hostgroups, servicegroups, and downtimes, the state request interface also allows clients to query the full state and configuration of individual objects. Like the command interface, the state interface can have multiple worker threads.

By default, the state request interface will return all the information about any objects that match the specification of the query, and clients are encouraged to explicitly list the keys they want returned for their objects. Certain keys (plugin_output, long_output, and perf_data – the texual output of check plugins) require locking the Nagios event loop to ensure thread safety while copying out the data, because they are changed by updated check results. The loop is only locked for the time necessary to copy these three strings out to the payload, so the chance of driving up latency is very low; it is more likely that state requests will take longer if they are issued when the reaper is running.

2.2 Modularity

NagMQ is meant to be as simple or as complicated as necessary for whatever application it is being used. All the interfaces in NagMQ are optional, and NagMQ allows the use of any of the transports and semantics available from ZeroMQ. Endpoints can connect and bind in any order, and clients will transparently reconnect if their peer endpoint becomes temporarily unavailable or if the connect started before the peer had bound to its address. ZeroMQ is brokerless, and clients can connect to and use NagMQ directly without any kind of broker, but the project is distributed with a simple proxy broker for providing advanced routing and for converting one messaging pattern to another (for example subscribing to check initiation messages and fair queuing them to worker processes for distributed check execution).

The NEB module has a dispatcher thread which is enabled when the state request and/or command interfaces are enabled. If they are not configured to run in their own threads, the dispatcher thread calls their processing routines directly. Otherwise, it dispatches commands and requests from connecting clients to the dedicated worker threads.

2.3 Security

NagMQ does not provide any encryption or authorization for messages, and ZeroMQ does not provide privacy or verification for messages. ZeroMQ 3.x has an option to restrict TCP connections to certain address ranges, but ZeroMQ 2.x, connection-level security must be done externally (e.g. through a firewall, or IPSec, etc.). Future work may include adding encryption to NagMQ by running JSON payloads through a block-cipher with a pre-shared key before being put on the wire.

2.4 JSON

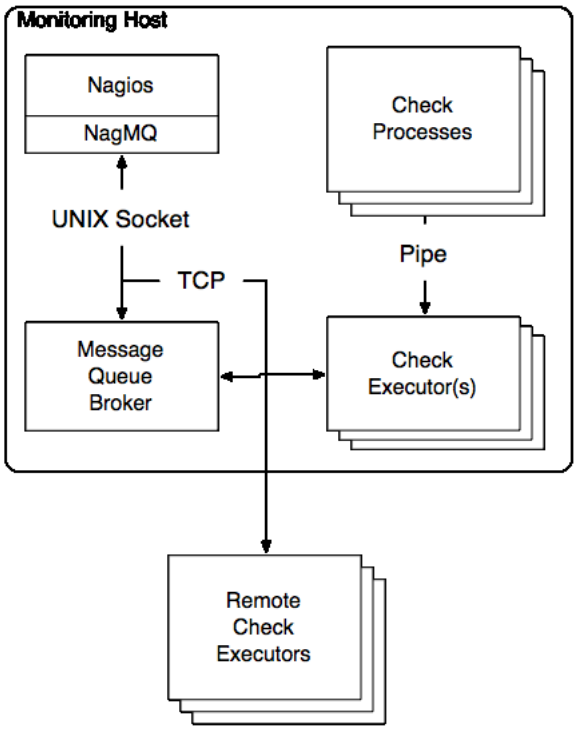
Since NagMQ sends and receives data as JSON payloads, the speed of the parser and emitter has as much of an impact on the performance of the whole module as the messaging system the data is transmitted over. NagMQ has its own write-only JSON emitter, which is used in the event publisher and state request module. Its performance is similar to the performance of the standard C library sprintf – in fact, most calls are just wrappers to sprintf and other standard C calls, with some additional memory handling. Any values needed for constructing a message prefix are cached as they are added, and the resulting buffer is freed by ZeroMQ when it has finished being transmitted to peers.

The command and state request modules also use the Jansson JSON library for parsing requests from clients. Although it is slower than the JSON emitter, because it constructs a hash table for each object it parses, all the modules that use it can be run in multiple separate threads to ensure high performance.

3. Included Applications

Included with the NagMQ event broker are a ZeroMQ proxy and a number of scripts and applications intended to show the possible uses of NagMQ; unlike many other Nagios add-on projects, NagMQ is not intended to be a single add-on that solves every problem on its own, so the included applications are intended to be examples that can be built upon when deploying Nagios, and many of them are written in Python and include no compiled code. We will discuss distributed check execution, high availability with active/passive failover, and some alternate user interfaces that all use NagMQ for communicating back to Nagios.

3.1 Distributed Check Execution



NagMQ includes a distributed check executor that subscribes to messages for the start of a service/host check and executes the command line of the check on whatever host the executor is running on. The executor itself is entirely asynchronous, using LibEv to coordinate I/O with child processes, reap return codes from exited children, and receive new events from ZeroMQ. When connecting to a message broker instead of subscribing directly to the event publisher, jobs will be automatically fair-queued to connected peers by ZeroMQ, and adding capacity requires only starting additional executor processes. The executor can filter which checks it runs either by changing the subscription on the message bus, or by applying a Boolean match against any field in the check initiation payload.

In addition to distributing checks across dedicated monitoring hosts, the executor can also be deployed to the edge as a remote plugin executor. Simply by subscribing to the beginning of service checks for a specific host and applying some clever check naming, the execution agent will run all remote checks as they happen in the Nagios event loop and send the results back without Nagios ever having to start a new process or run a command. At the edge, the remote-execution agent can be completely stateless, with the configuration only telling it how to connect to the message queue, rather than having to list explicitly each command the monitoring system is allowed to run.

3.2 Active-Passive Failover

A basic agent that provides active-passive high availability for Nagios comes with NagMQ and uses a periodic program status message as a heartbeat to detect when the active node has failed. When the agent starts, it launches Nagios with all active checks, notifications, and event handlers disabled. It then requests the state of all the hosts and a service configured on the active node from the state request interface and submits the result to the newly started passive node's command module – it has a special routine for parsing the current state of an array of hosts and services for this purpose. It resolves differences between downtime and comments, performs any fencing required to ensure the active node is not disrupted, and enters a loop of receiving events for check results, downtimes, comments, and acknowledgements and forwards them into the passive node.

If there is a timeout in receiving a check result or other state message to be forwarded to the passive node or a program status heartbeat message, the agent will perform any fencing necessary to become the active node, and send the necessary commands to the node being promoted to start active checks, notifications, and event handlers.

Once a node has become the active node, it receives and discards its own program status message and will perform fencing actions to become passive if it detects Nagios has failed. This gives reasonable assurance that if Nagios on the active node has failed but the HA agent/OS is still running, any shared resources like IP addresses can be cleanly taken over by the new active node.

3.3 User Interfaces

Stock Nagios comes with a web interface that uses a number of CGIs to parse Nagios status data and submit

commands back through the command pipe. NagMQ allows for multiple user interfaces that are able to obtain their data from the state request interface of NagMQ and submit commands through the command interface. In addition to being able to query only the information that the user needs to display, the different transports available to NagMQ allow the user interface to be on another host, or a group of hosts, that are separate from the Nagios monitoring hosts.

NagMQ comes with a command-line interface for Nagios which allows users to view current status – add and remove acknowledgements, downtime, and comments – and enable and disable checks and notifications. It uses the username of the user invoking the command as the contact name to determine whether a user is authorized to submit commands for given targets, and can act on hosts, hostgroups, and service names.

4 Performance

Testing the performance of NagMQ is difficult because it is not a single-purpose application. While developing the applications included with NagMQ, it was often the case that performance problems occurred when the program connecting the message queue couldn't decode the JSON payload fast enough – this is why the distributed check executor is implemented in C.

Since check execution uses several parts of NagMQ (the publisher to receive check events, the message queue broker to divide work up between connected peers, and the command interface for receiving check results), we will compare the check latency and the number of checks run by the check executor per minute to demonstrate the relative performance of NagMQ to stock Nagios and an existing distributed check executor.

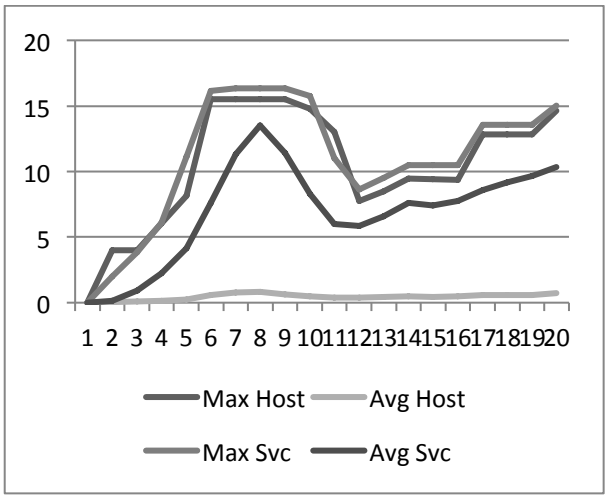
We tested Nagios 3.4.1 in its stock configuration, with NagMQ and its included distributed check executor, and Mod_gearman. Nagios was configured with 20,000 service checks across 2,000 hosts with a dummy check script. The check interval for services was every 3 minutes with a 1-minute retry interval for services – services were configured for 2 check attempts to determine a hard state; and hosts only checked once. Each test ran for 20 minutes, and Nagios started each time with no state from any previous tests.

The check script printed a random string, slept a random number of seconds, and returned a semi-random return code. The return code was based on the random number used for selecting the string that was printed, but was weighted towards returning a successful return code 90% of the time, and a problem 10% of the time. The checks routinely returned an error in order to test the code paths in Nagios that can introduce latency, such as on-demand host checks, retries, and executing many notifications.

All performance graphs have time in minutes on the horizontal axis, and latency in seconds on the vertical axis. Averages for checks executed per minute exclude the first sample for all tests, because Nagios had not been running for a full minute when they were sampled.

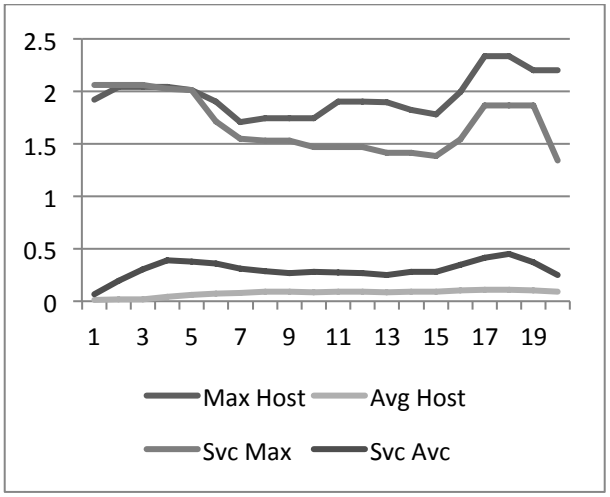
4.1 Stock Nagios

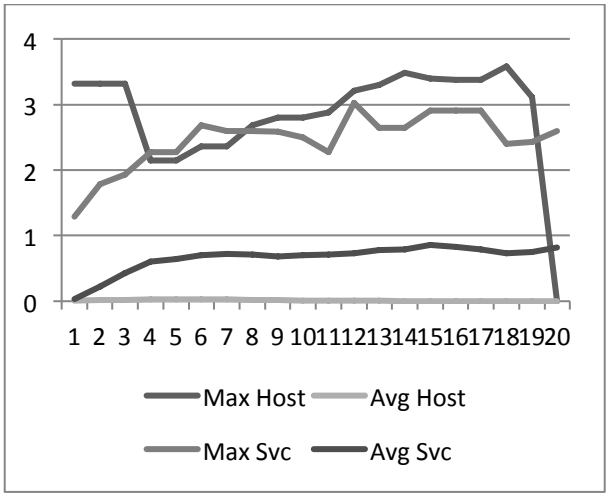
Check latency increased gradually over the course of the test with stock Nagios, with a maximum latency of 14.646 seconds for hosts and 14.014 seconds for services, and a maximum average of 0.73 seconds for hosts and 10.35 for services. Nagios was able to consistently execute an average of 976 host checks and 6219 service checks per minute.



4.2 NagMQ

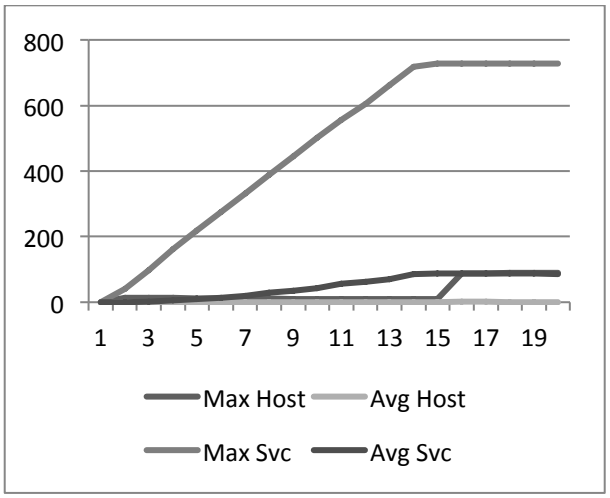
The average check latency for both hosts and services never went above 0.5 seconds during the test, and the maximum check latency was above 2.5 seconds. The executors consistently ran an average of 1,028 host checks and 6,726 service checks per minute.





4.3 Mod_gearman

Mod_gearman had the least expected performance characteristics of the three environments. The first test was run with the check script sleeping a random number of seconds, and the average check latency went up consistently. The number of checks executed per minute was an average of 828 host checks and 747 service checks.



Because performance was so poor, the check script was modified so it no longer slept, but returned immediately. The performance in this test was much better, with check latency for hosts dropping to 0 by the end of the test. The workers consistently executed 1,507 host checks and 7,063 service checks per minute.

4.4 Results

The performance of NagMQ overall varies for different applications, but testing of the distributed executor shows similar performance to popular distributed executors like mod_gearman, with latency below 1 second. Performance of the NagMQ event broker is broken down into two categories: how quickly it can generate JSON message payloads and how quickly it can send and receive payloads.

The JSON emitter that comes with NagMQ was tested by generating 20,000 test payloads with constant data similar to the results of a service check – the equivalent of 16.9 MB of check data – and took 0.27 seconds.

Testing of NagMQ was performed on virtual machine configured with 4 CPUs and 8GB of memory. ZeroMQ is designed to move data around very quickly, and although results will vary with different hardware, the performance of ZeroMQ on our test platform is shown below:

100,000 roundtrips of 4096 bytes via UNIX sockets	
Latency	43.497 [us]
Throughput	138432 [msg/s] 4536.140 [Mb/s]
100,000 roundtrips of 4096 bytes via TCP/IP	
Latency	63.169 [us]
Throughput	87109 [msg/s] 2854.388 [Mb/s]

Problems inside Nagios remain that limit the performance of the system overall. Recent builds of Nagios contains many fixes and improvements that make it faster overall, such as replacing the scheduling queue doubly-linked list with a priority queue (Ericsson, Commit #1971, 2012). In older versions of Nagios, any performance gains from using NagMQ may only be covering up an underlying inefficiency.

5 Future Work

The NagMQ project is mature enough for production use, but there are a lot of enhancements possible to expand its usefulness and improve monitoring with Nagios. The development of NagMQ has been very organic thus far. The first version was intended to include only the event publisher, with applications either waiting for events to determine state or submit commands via the command pipe. The addition of the command module was introduced next to implement distributed check execution. Finally, the state request module was added to implement a command-line interface. There have been additions to all modules as new applications were developed.

The design of NagMQ is limited to a single messaging system and a single payload format, and there have been requests from users for support for other messaging systems, which would also mean different data formats. Although the input portion of NagMQ would have to be rewritten, the emitter could be adapted to support different data formats. Future work could support modular support for different data formats and messaging systems.

NagMQ has no security that is not provided by ZeroMQ, and is limited to ACLs for restricting what IP addresses and CIDR ranges are allowed to connect to a bound address. Future versions of NagMQ and its included applications should include improved security both for authenticating to NagMQ, and for providing encryption for all traffic in and out of the NEB.

Presently NagMQ is only able to receive state information from clients – the configuration of Nagios is still firmly rooted in text files. Adding the necessary interfaces to NagMQ to allow it to receive configuration items and add them to Nagios at run-time would allow Nagios to become completely stateless.

Although it had not been released at the time this paper was written, Nagios 4.x was in testing, and the way Nagios executes checks and communicates with itself has changed significantly in the test builds of Nagios 4. Instead of running checks (Ericsson, Commit #2020, 2012), notifications (Ericsson, Commit #2026, 2012), and processing incoming events in the event loop, it uses dedicated worker processes and asynchronous message passing to get execution out of the event loop. This change focuses Nagios core on being a better scheduling engine, and moves its design closer to the design of NagMQ.

6 Conclusion

The popularity of Nagios leads to a hesitancy in changing the way it interacts with the checks it needs to run, users, and other applications. Although there are a number of different projects that improve specific parts of communication with Nagios, NagMQ attempts to provide a generic interface that is easily accessible from scripts and applications. Although the performance of communicating with NagMQ can vary widely depending on the performance of the JSON parser and emitter used, in the best case it has performance comparable to other popular projects for low-latency applications like check execution.

A big focus of NagMQ was that it should have minimal dependencies and not require many patches to stock Nagios in order to be useful. There were changes required to Nagios to ensure communication between the state request interface and the event loop is thread-safe, which have already been submitted back to the Nagios project. Otherwise, the project includes the Jansson JSON parser and libev, which are used in both the NEB module and the distributed executor, but requires the user to provide the ZeroMQ libraries for C and any scripting languages they wish to use.

NagMQ is licensed under the Apache 2 license and available online at https://github.com/jbreams/nagmq.

7 Works Cited

Augustine, A. (2007, October 17). *DNX Version 0.13 Released!* Retrieved September 13, 2012, from DNX Announce Mailing List: http://sourceforge.net/mailarchive/message.php?msg_id=89683

ConSol* Labs. (n.d.). *Mod_gearman*. Retrieved September 13, 2012, from http://labs.consol.de/nagios/mod-gearman/

Ericsson, A. (2012, June 21). *Commit #1971*. Retrieved from Nagios Subversion Repository:

http://nagios.svn.sourceforge.net/viewvc/nagios?view=revision&revision=1971

Ericsson, A. (2012, July 9). *Commit #2020*. Retrieved from Nagios Subversion Repository: http://nagios.svn.sourceforge.net/viewvc/nagios?view=revision&revision=2020

Ericsson, A. (2012, August 2). *Commit #2026*. Retrieved from Nagios Subversion Repository: http://nagios.svn.sourceforge.net/viewvc/nagios?view=revision&revision=2026

Ericsson, A. (n.d.). *Merlin*. Retrieved September 13, 2012, from Op5 Community Exchange: http://www.op5.org/community/plugin-inventory/op5-projects/merlin

Galstad, E. (2004, December 15). *Nagios 2.0b1 Available*. Retrieved September 13, 2012, from Nagios-announce mailing list: http://sourceforge.net/mailarchive/message.php?msg_id=210063

Galstad, E. (2007, April 18). *NDOUtils Documentation*. Retrieved from Nagios Core Manuals: http://nagios.sourceforge.net/docs/ndoutils/NDOUtils.pdf

Galstad, E. (2010, September 1). *Nagios Core 3.2.2 Released*. Retrieved September 13, 2012, from Nagios Announce Mailing List: http://sourceforge.net/mailarchive/message.php?msg_id=26080524

Hintjens, P. (n.d.). Retrieved from ZeroMQ Guide: http://zguide.zeromq.org/page:all

Hintjens, P. (2005). *Background to AMQP*. Retrieved September 13, 2012, from ZeroMQ Wiki: http://www.zeromq.org/whitepapers:amqp-analysis

Hintjens, P. (2012, September 9). *Welcome from AMQP*. Retrieved from ZeroMQ Wiki: http://www.zeromq.org/docs:welcome-from-amqp

iMatix. (2008, June 10). *Introduction to OpenAMQP*. Retrieved September 13, 2012, from OpenAMQP: http://www.openamq.org/doc:user-1-introduction

O'Hara, J. (May 2007). Toward A Commodity Enterprise Middleware. *ACM Queue*, 48-55.

Pieter Hintjens, M. S. (2010, April 23). *Multithreading Magic*. Retrieved September 19, 2012, from ZeroMQ Wiki: http://www.zeromq.org/whitepapers:multithreading-magic

Sustrik, M. (2008, December 12). *Broker vs. Brokerless*. Retrieved from ZeroMQ Wiki: http://www.zeromq.org/whitepapers:brokerless

Sustrik, M. (2011, September 2). *0MQ/3.0 pubsub*. Retrieved from ZeroMQ Wiki: http://www.zeromq.org/whitepapers:0mq-3-0-pubsub

Efficient Multidimensional Aggregation for Large Scale Monitoring

Lautaro Dolberg, Jérôme François, Thomas Engel
University of Luxembourg
SnT - Interdiciplinary Centre for Security, Reliability and Trust.
Email: firstname.lastname@uni.lu

Abstract

Today, network monitoring becomes necessary on many levels: Internet Service Providers, large companies as well as smaller entities. Since network monitoring supports many applications in various fields (security, service provisioning, etc), it may consider multiple sources of information such as network traffic, user activity, network events and logs, etc. All these ones produce voluminous amount of data which need to be stored, visualized and analyzed for administration purposes. Various techniques to cope with scalability have been proposed as for example sampling or aggregation.

In this paper, we introduce an aggregation technique which is able to handle multiple kinds of dimension, *i.e.* features, like traffic capture or host locations, without giving any preference a priori to a particular feature for ordering the aggregation process among dimensions. Furthermore, feature space granularity is determined on the fly depending on the desired events to monitor. We propose optimizations to keep the computational overhead low.

In particular, the technique is applied to network related data involving multiple dimensions: source and destination IP addresses, services, geographical location of hosts, DNS names, etc. Thus, our approach is validated through multiple scenarios using different dimensions, measuring the impact of the aggregation process and the optimizations as well as by highlighting the ability to figure out important facts or changes in the network.

1 Introduction

Monitoring is a fundamental part of network management. It is essential for checking the network activity and status, e.g. tracking abnormal facts or changes (attacks, configuration errors, failures, etc). Several steps are required for this. First, data has to be collected from various sources and locations. Then, such data must be stored before it can be directly visualized or analized by human expert to provide summarized information, like alarms, to the network operational team.

For example, usual data collected in network might consist of full packet captures on a network or Netflow [9] records, for ISPs (Internet Service Provider), which are known to be helpful in network management context [10]. DNS traffic is also a valuable information in the security context for detecting botnets [6] and malicious domains hosting malware [1]. Geographical or network location of hosts might be helpful for placing servers at the right place in particular within a CDN (Content Distribution Network). Application level analysis can include monitoring of different exchange message types for a given protocol. For instance, a simple counter for error messages in SIP or HTTP may indicate some problems on the network. IDS (Intrusion Detection Systems) or firewall alerts are clearly relevant by nature.

Thus, there is plenty of valuable information which might be also correlated together, network traffic, DNS names or host locations, etc.

Since nowadays volume of such information grows rapidly, scalability represents a challenge. When information volume results to be massive, computing resources infrastructure (storage and analysis) demands for fine grained information, such as deep packet inspection, are too high to be practicable, in some cases inaccurate. For example, thanks to our partners involved in the operational field in Luxembourg, we obtain a Netflow and a DNS dataset. The average number of flows is 60,000/sec and can reach 100,000/sec. For forensics analysis, we faced with the a huge size of the DNS dataset including around 40M unique records over a 10 months period.

In this paper, we improve network monitoring by targeting scalability issues for storage, analysis and visualization of huge volumes of network related data. We analyze multiple sources such as traffic flows or DNS records to infer a global knowledge that might be help-

ful to identify particular events (normal or abnormal). Recent research has shown promising results for aggregation based techniques on macroscopic monitoring [8, 34, 20]. Although we explored aggregation over destination or source IP address separately in [34], this paper proposes a new method implemented as an open source tool which:

- handles multiple types of data
- defines a new tree based structure and algorithms to combine them into a single tree,
- is evaluated in multiple scenarios
- can be easily extended to include new features

Hence, we propose *MAM* (Multidimensional Aggregation Monitoring) is proposed for network related data aggregation including many features (also called dimensions) on very large collections. The goal of this paper is to present *MAM* from a formal point of view as well as from a practical point of view and study its viability for network monitoring purposes.

MAM can consider simultaneously the source and destination IP addresses and the ports of network traffic. A fundamental idea of our approach is to aggregate data, such as traffic load, over multiple dimensions without giving any preference to one. Considering the previous example, it means that the data will not necessarily be aggregated on source IP addresses first and then on ports. Usually, this decision is made by human experts using their own expertise. Some of them will monitor the usage per service first (ports) and then per IP addresses whereas others prefer to have statistics per IP addresses first and then the details for each service. In the first case, statistics about global traffic of an IP address are not directly available and need to be reconstructed by iterating over all ports. Adding more dimensions will add other levels that make the choice of ordering difficult and computational less efficient if some statistics need to be reconstructed afterwards.

Moreover, IP addresses monitoring usually relies on a subnet basis (/X networks) where the human administrator has to specify the prefix size X to use. In our case, the space of a dimension is not split a priori like a fixed size of subnets. The aggregation can thus lead to keep track of information related to IP subnets having different sizes.

In fact, the aggregation process is guided by the desired events to monitor, `i.e.` reaching a quantifiable visibility (5% of the traffic load in bytes or packets for instance) which will alleviate the administrator from both problematic decisions (order of dimensions and split over the space).

Scalability can be achieved by optimizing the usage of storage data structures. *MAM* leverages a tree based structure to store data in a hierarchical representation. It has a bounded size and different strategies are described to satisfy this constraint. Obviously, the strategy can impact the relevance of stored data as well as the aggregation process itself.

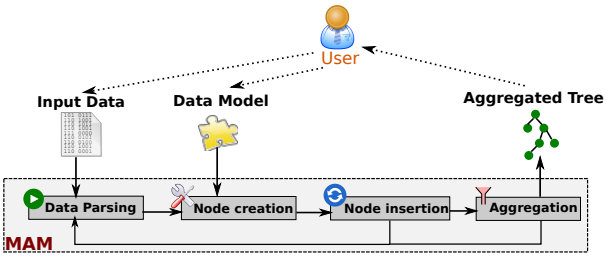


Figure 1: MAM overview

This paper is organized as follows. Section 2 overviews the proposed tool. Section 3 is related work. Section 4 formally defines the data structures for multidimensional aggregation. Section 5 explains spatial aggregation techniques. Section 6 proposes several strategies to cope with scalability. Section 7 is dedicated to the evaluation. Finally, Section 8 concludes the paper and describes our future work.

2 MAM Overview

Figure 1 highlights the main steps of our tool and so the main components. The objective is to aggregate multidimensional data into a single tree structure for each time window (temporal dimension). If the timing information is not provided, *MAM* can still be used to create a single tree for the whole dataset.

For being executed, the user has to provide input data as well as a data model describing the type of data and how to parse and aggregate it. Some data models are already provided with the tool as for example the IP address, services or geographical coordinates. Therefore, the first step is to parse the input data. For each data instance, (usually a line in a file), *MAM* creates a node accordingly. The latter is inserted into the current tree based on the hierarchy of each dimension (next sections provide details). These steps are repeated until the tree has to be compressed, `i.e.` nodes in the tree are aggregated from the leaves to the root for keeping solely relevant and aggregated information. This happens when the size of the tree is too high (online aggregation) for reducing resources consumption or at the end of a time window (simple aggregation). In this case, the tree is returned as a result to the user and *MAM* will continue the process starting from parsing next time window.

3 Related Work

Regarding scalability concerns, flow based information such as Netflow records [9], compresses IP traffic in a compact format [9]. Even mainly supported by most commercial routers, they are still known to require a large storage infrastructure, since Netflow records growth can be undefined.

Thus, sampling techniques have been proposed in the past [11, 29]. Sampling may clearly resolve the scalability issues by discarding large portion of data. However, the main drawback is the difficulty to set an appropriate sampling rate to discard enough information without impacting the relevance of observations made from the remaining data.

Using efficient data structures to optimize storage and access was also explored. Giura et al. [17] introduce NetStore, an efficient storage infrastructure for network flow data using a column-oriented storage that outperforms row-based techniques. BadHoods[27] performs aggregation on a subnet basis to demonstrate that malicious hosts tend to be close into the IP space but they highlight the difficulty to set appropriately the prefix size of subnets to monitor. In addition, Fenwick trees [12] handle single dimension by storing efficiently prefix sums for given values represented as a table.

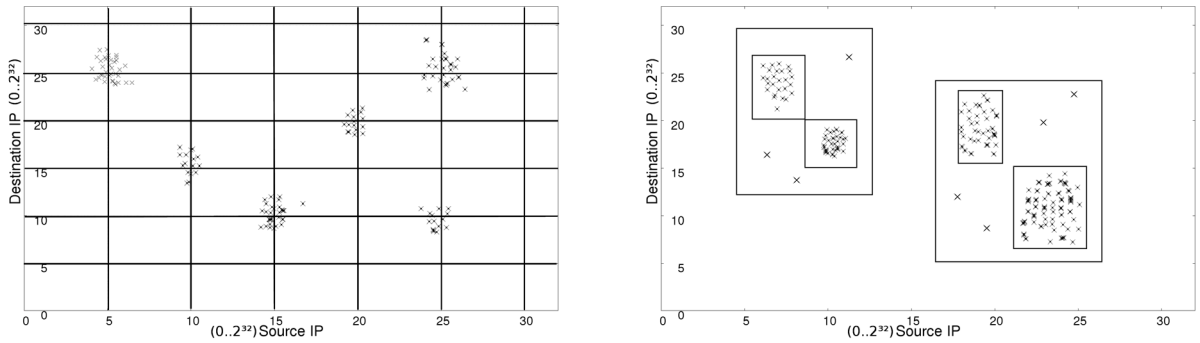
Flow aggregation based on entropy is proposed in [19] where the authors introduce a two dimensional hashtable (source and destination IP addresses). An alternative is to collect data from multiple sources as proposed in [35] to build a community graph for sharing information about host activities based on IP addresses. The scalability is addressed through data filtering as well as using multiple graph construction which can be achieved independently in parallel. In the past, hosts interaction is also a manner to compress data into a graph representation. This is known to be useful for detecting anomalies like botnets [15] in particular when instantiated with the MapReduce paradigm for being executed in a distributed fashion [14]. Another solution for traffic aggregation was presented in Aguri [8] and further on in our previous work Danak [34]. Both of them leverages a tree based structure for storing traffic data on a single dimension. IP addresses are represented by a tree following the common hierarchy of the subnets. Using such structures, it has been proved that anomalies in large scale networks can be tracked as for instance spam or distributed denial of service.

Other sources of information are relevant as noticed in introduction. Assuming DNS data, a context-aware clustering method is proposed in [31]. In fact, two trees are created. One for representing the IP subnets like in [8] and one for representing the DNS domains in a similar manner based on the parent-child relations between subdomains. Log analysis is also concerned and recent advances like [21] promote the efficient programming design as well as data structure for correlating log events in a scalable way.

From a general point of view, storing multi-dimensional data has been investigated in the past. While data structures cited in [5, 13] are plausible modeling tools for K-Dimensional spaces, they are recommended to be used on regular partitioned spaces. Therefore, the previous mentioned data structures requires a pre-established order among dimensions. Some contributions to this subject was made by [28] on Flamingo, and EtherApe [3, 33]. Regarding Data Cube [18], maintaining the whole cube structure is costly from a computational point of view. Furthermore, data requires to be stored within a relational structure and aggregation needs to be fully specified by the user in particular the granularity and the order among dimensions unlike our approach. Some cube approaches fail to process online data since they become no longer valid after new data has been added. In case of our tool, it is possible to append nodes on the fly without additional cost. Flamingo is a visualization tool for monitoring Internet traffic in real time, rendering 3D images. Aggregation is also applied by IP prefix, *i.e.* subnets, and the prefixes are chosen based on routing table entries. EtherApe is a network traffic monitoring tool. Network traffic is displayed as nodes, as for example IP addresses, and links with color codes to represent different protocols.

Some tools are also dedicated to visualization. ENAVis [23] represents multi-dimensional data into a graph where each node contains a single dimension. This helps to see relationships between users and hosts for example. Even if similar to our work, the main fundamental difference is that we chose a tree-based structure where nodes represent multi-dimensional instances of data. Furthermore, our approach integrates an online aggregation technique for automatically split the multi-dimensional space in subspaces. SIFT [36] includes IP subnet aggregation but the user still has to set the granularity of details. This is different from our approach, the same tree subnets can be visualized once they reach a certain threshold of measurable information like the traffic load.

Also our work is mainly complementary to [34, 8], *MAM* aims to aggregate relevant data on several spatial dimensions (IP addresses, ports, etc) like [23, 28]. However, it differs from these latter since the order between dimensions is not defined a priori as well as the space division is automatically determined.



(a) Regular IP Address space partitioning

(b) Dynamic IP Address space partitioning

Figure 2: Different IP Address space partitioning examples

4 Data Aggregation Structures

Aggregation consists to reduce granularity of data by grouping data instances according to a criteria, *i.e.* sharing similar properties.

For instance, aggregation over the IP address space could consider the shared prefix as the property leading to monitor network traffic by subnet. Defining the size of the subnet, equivalent to the prefix size, is quite arbitrary like /24, /28, etc. Changing this parameter leads to different observations regarding the context [27]. This is illustrated in Figure 2 with a naive case example where a regular partition of IP address space, in Figure 2(a), may not lead to identify the clear subgroups of hosts unlike a dynamic partitioning in Figure 2(b). For example, to monitor the hosts listed on Traffic Flow Table 1, monitoring /16 networks will give a general overview of the network activity which is probably too coarse and /24 seems more useful. However, using /24 or more can be too fine-grained at the Internet level. Moreover, since traffic is probably not well-balanced between machines and subnets, some of them should be more carefully monitored, equivalent to consider higher prefix size, while others may be monitored with a coarse-grained view (small prefix size).

To counter this problem, aggregation can be guided by the nature of the events to monitor. For example, the traffic load can be aggregated in order to observe phenomenas reaching a certain proportion of the entire traffic load or of IDS alerts.

For aggregation on a single dimension, a tree structure is suitable [34, 8]. Spatial representation of a bi-dimensional space can be done using a quad tree structure [13] where each internal node has exactly four children. Normally the space is recursively portioned into four quadrants or regions. Then for partitioning a three dimensional space a similar structure, an oct-tree, can be used. A general structure supporting M dimensions is a multidimensional tree (k-d Tree [5]) for k-dimensional space-partitioning. However, in our context, the dimensional space division is not known in advance and done on the fly when the tree is created.

In this paper, we consider features where it is possible to derive an underlying hierarchical relationships covering all potential data instances represented as nodes in a tree. Assuming two data nodes, one is qualified as more specific or there is no relationship between them. Formally the hierarchical relationship between two nodes n_i, n_j represent is given if there is a path from the root of the tree to n_j that passes through n_i. In that case, n_i is more general than n_j.

4.1 Single Dimension

Spatial and temporal aggregation on a single dimension for traffic flows was proposed in Aguri [8] and Danak[34]. Temporal aggregation splits the dataset into fixed size time windows on which spatial aggregation is applied. We extend the notion of spatial aggregation to multiple and generic dimensions (features).

IP address aggregation is performed by extracting the traffic volume (bytes or packets) for the source or destination addresses. Aggregation is based on a tree structure following the common subnet hierarchy where each node represents an IP subnet or a single address. The total volume of traffic transmitted is decomposed in particular volumes expressed as proportions or percentages. Nodes with a proportion of traffic lower than α are aggregated into their parents. An example of this is given in Figure 3 from Traffic Flow Table 1. In this small example, only nodes with more than 10% of the total traffic are kept. As highlighted, root concentrates the global traffic of a /17 network. Each node contains the following information:

1. Dimension name and value (i.e: {app:ROOT}, {src_ip:0.0.0.0/0}, etc)

Traffic Flow Table 1 Traffic flow example of a network nodes within 192.168.0.0/16 (Web and Mail services)

PORT	PROTO	KB	TIME	SOURCE	DEST
80	TCP	1491	2010−02−24 02:20:15	192.168.6.2	92.250.221.82
110	TCP	988	2010−02−24 02:20:19	192.168.8.2	92.250.223.87
443	TCP	902	2010−02−24 02:20:27	192.168.11.2	92.250.220.82
110	TCP	1513	2010−02−24 02:20:29	192.168.112.1	92.250.222.81
80	TCP	1205	2010−02−24 02:20:29	192.168.11.1	92.250.220.82
80	TCP	1491	2010−02−24 02:20:31	192.168.1.2	92.250.220.83
110	TCP	1467	2010−02−24 02:20:39	192.168.12.2	92.250.221.81
80	TCP	927	2010−02−24 02:20:39	192.168.12.2	92.250.220.82
443	TCP	1294	2010−02−24 02:20:39	192.168.11.1	92.250.223.82
110	TCP	940	2010−02−24 02:20:49	192.168.21.2	92.250.221.81
80	TCP	917	2010−02−24 02:20:49	192.168.23.1	92.250.220.82
443	TCP	460	2010−02−24 02:20:59	192.168.26.2	92.250.220.85

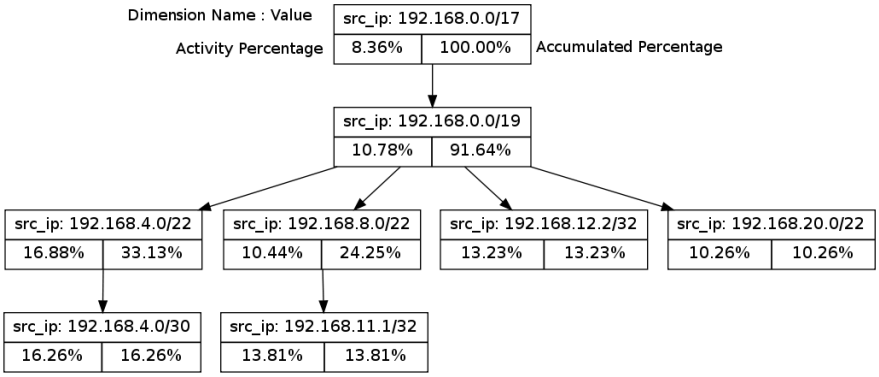


Figure 3: Single dimension tree (source IP addresses) based on Traffic Flow Table 1, activity volume: number of bytes, $\alpha = 10\%$

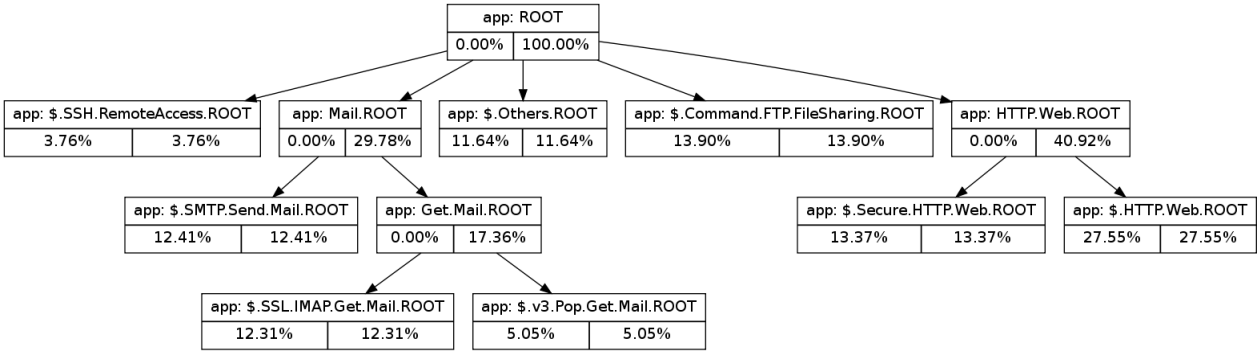


Figure 4: Single dimension tree (application) based on Traffic Flow Table 3, activity volume: number of bytes, $\alpha = 5\%$

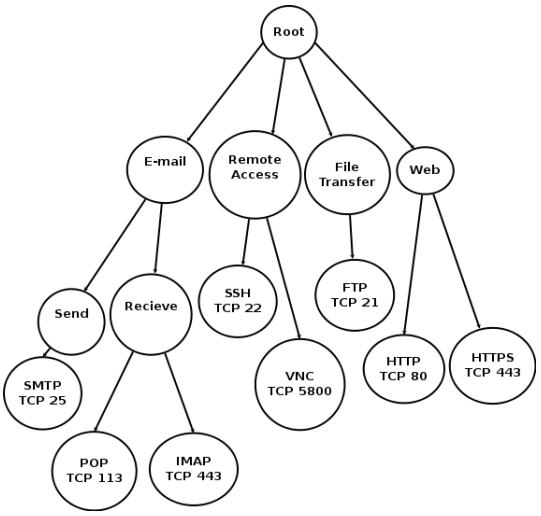


Figure 5: Example Application Taxonomy by TCP Port Numbers

2. Percentage of aggregated activity (activity volume defined as *vol*) for the current node
3. Cumulated percentage of activity of the node and its subtree defined as *acc_vol*

Intuitively, a single dimension tree represent a subset of the entire hierarchy (all the possible values) of a given feature like illustrated in Figure 3.

Formally an IP address single dimension tree of N nodes is [34]:

- A set of N nodes, where T = $\{n_0 \ldots n_N\}$ and $n_i =< prefix_i, prefix\_length_i, vol_i >$. IP subnets are decomposed using CIDR format [16]. $prefix_i$ and $prefix\_length_i$ are the prefix value and size of node n_i while vol_i is the entire traffic load associated to the IP addresses included in the subnet n_i.
- A parent-child relationship where a $child : T \rightarrow \mathscr{P}(T)$ returns a set of child nodes for a given node.

Single dimension aggregation can also be done for many other attributes such as protocol messages, port numbers, spatial coordinates. Aggregation for TCP port numbers, using data from Traffic Flow Table 3, is highlighted in Figure 4. In this case, every node represents an application family or a specific one defined by the taxonomic classification in Figure 5. In our paper, a dimension is a feature which the values may be represented in a hierarchical tree with final values at the leaf nodes and group values as internal nodes.

This definition can be extended for a generic dimension tree as follow:

- A set of N nodes, where T = $\{n_0 \ldots n_N\}$ and $n_i =< f_i, vol_i >$. Where f_i is an associative array modeling a dimension from a given traffic flow. For example in case of application port $f_i = \{app : value_i\}$ where app is a label and $value_i$ is the a string modeling a path in the taxonomic tree described in Figure 5. As mentioned before, it can a full path to a leaf or an intermediate branch describing a sub family of applications. For IP addresses, f_i is $\{prefix : prefix_i\ ,\ prefix\_length : prefix\_length_i\}$.
- A parent-child relationship where a $child : T \rightarrow \mathscr{P}(T)$ returns a set of child nodes for a given node.

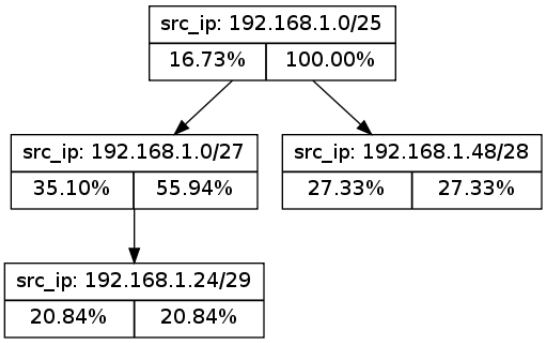


Figure 6: Single dimension tree (Source IP addresses) from Traffic Flow Table2

Single dimension aggregation has proven to be an effective technique for practical network analysis methods and anomalous network traffic detection [34, 8]. However, information from network traffic includes more than one dimension. Furthermore, the anomalies can be present in a combination of dimensions.

A port scanning has to consider the application/service feature while IP scanning has to monitor IP addresses. Assuming a botnet doing port scanning from and to multiple IP addresses, all these features (source and destination IP addresses, destination ports) need to be monitored meanwhile to observe the attack activity globally. So predicting the single dimension or the combination of dimensions to monitor is hard.

Considering Traffic Flow Table 2 illustrating a naive case of several hosts performing a DDoS against a web server. IP address based aggregation (illustrated in Figure 6) cannot clearly detect it but aggregation using TCP port will do. Another scenario is Traffic Flow Table 3. In this case, a reduced group of hosts are targeting several applications which cannot be caught by aggregation on TCP ports. However, the multidimensional tree depicted in Figure 7 is able to detect this behavior. This highlights that 20% of the traffic is due to web session initiations (HTTP) in 192.168.0.0/20 subnet.

4.2 Multiple Dimensions

To extend the previous approach, we present an aggregation technique for monitoring network using multiple dimensions at the same time (eg: IP Address, Full Qualified Domain Name (FQDN), TCP ports, GPS Coordinates, etc). Multidimensional aggregation is performed by using a user-defined threshold α and an interval of η seconds. The latter defines the size of a time window. For each of them, the data is assembled into a tree composed of nodes, including multiple dimensions, where only those such that $acc\_vol > \alpha$ are kept.

In Figure 7, source and destination IP addresses and application from Traffic Flow Table 3 are aggregated.

Assuming a data instance that can be decomposed in many dimensions, a formal definition of a multidimensional tree of M dimensions and N nodes is as follows:

- A set of M associative arrays that model the M dimensions
- A set of N nodes, where T = $\{n_0 \dots n_N\}$ and $n_i = < \{f_{i_1} \dots f_{i_m}\}, vol_i >$, $f_{i_j} \in \{f_{i_1} \dots f_{i_m}\}$ is an associative array modeling the $j-th$ dimension according to the previous definition

  Thus, we can define $f_{i_1} = \{prefix : prefix_i \ , \ prefix\_length : prefix\_length_i\}$ for IP addresses. For the application/service level, $f_{i_3} = \{app : value_i\}$ where app is a fixed label and $value_i$ is the a string modeling a path in the taxonomic tree described in Figure 5.
- A parent-child relationship where a $child : T \rightarrow \mathscr{P}(T)$ returns a set of child nodes for a given node.

5 Tree Based representations

The core idea of a tree-based aggregation mechanism is to aggregate multi dimensional data for creating a summary of the network activity for each time window. Aggregation can be used for post-analysis or for real time monitoring. In the first scenario, memory and computational cost can be rise considerably for reaching a high level of precision. However in the case of real time monitoring execution time is a real constraint. Thus, two optimization strategies are introduced in Section 6 for real time computing that may lead to a particle approach for monitoring.

In this Section, we refer to the simple aggregation which is executed at the end of each time windows.

5.1 Simple aggregation overview

As long as a single data instance is inputed, a leaf node is built after extracting relevant information (dimensions and values). to be inserted at the right place (or updating the node in the tree if it already exists) creating intermediate nodes might be necessary, like nodes representing intermediate IP subnets. If the node already exists, vol_i is updated accordingly. For producing an outline afterwards, the tree is traversed post-order to aggregate nodes and to compute cumulative percentages (acc_vol). If the activity volume, vol_i, of a node n_i is less than the aggregation threshold, α, it is aggregated to its parent node, n_j. Thus, the node n_i is removed, vol_i is added to vol_j and all child nodes of n_i are attached to n_j. This allows to delete intermediate nodes which do not represent large activity volumes unlike their child nodes. Otherwise ($vol_i < \alpha$), the node is kept as it is.

Moreover, during the post-order traversal, activity volumes, vol_i are computed as percentages. In fact, they are stored as absolute values during the tree construction since the total activity volume is not known. At the end, thanks a global counter, $VOL = \sum_i vol_i$, each vol_i is updated accordingly, `i.e.` vol_i/VOL.

5.2 Directions

Due to the hierarchical relationships, we suppose there is a strict order relation between values of the same dimension. For constructing the multidimensional prefix tree structure, developing the concept of directions is necessary. Intuitively the directions correspond to find the correct path in the tree to attach a node or to navigate within the tree in order to access a given node.

Some dimensions are more likely to find or define a natural direction. For example, IP addresses have two directions, 0 or 1, modeling the next bit value. Regarding the subnet X.Y.Z.0/24, all IP addresses which the 25th bit is 1 (like X.Y.Z.128/25) will be placed on the left branch while every others which this bit is 0 will be placed on the right branch.

Other dimensions like UDP/TCP ports can be compared as integers but it is not so relevant as services of the same category like mail are not necessarily neighbors in the port range. In this case, a hierarchical classification, like in Figure 5, is required. In our tool the direction function may be customized by the user. Direction function is used to label the parent-child relationship on the multidimensional tree. Therefore, we also consider the longest common prefix which represents the most specific common ancestor between two nodes

Assuming $n_i = < \{f_{i_1} \dots f_{i_m}\}, vol_i >$ and $n_j = < \{f_{j_1} \dots f_{j_m}\}, vol_j >$, the longest common prefix is defined as:

$$lcp(n_i, n_j) =< \{f_{lcp_1} \dots f_{lcp_m}\} \quad (1)$$

where f_{lcp_i} is the most specific common part for the ith dimension, which corresponds to the longest sequence of

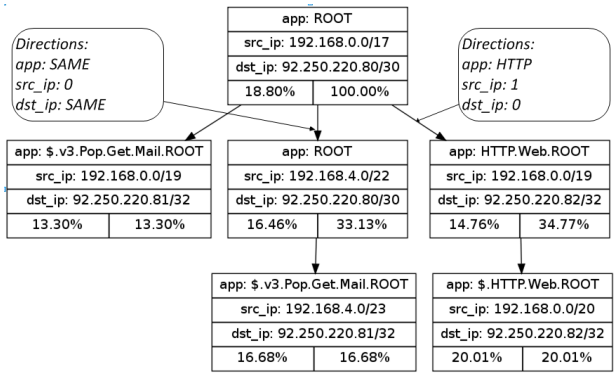


Figure 7: Multiple dimension aggregation based on Traffic Flow Table 3, $\alpha = 10\%$

Traffic Flow Table 2 Traffic Flow Table showing an example for a possible DDoS against a web server.

PORT	PROTO	KB	TIME		SOURCE	DEST
80	TCP	895	2010−02−24	02:20:59	192.168.1.17	92.250.220.82
80	TCP	47	2010−02−24	02:20:59	192.168.1.25	92.250.220.82
80	TCP	570	2010−02−24	02:20:59	192.168.1.45	92.250.220.82
80	TCP	952	2010−02−24	02:20:59	192.168.1.44	92.250.220.82
80	TCP	408	2010−02−24	02:20:59	192.168.1.61	92.250.220.82
80	TCP	609	2010−02−24	02:20:59	192.168.1.9	92.250.220.82
80	TCP	690	2010−02−24	02:20:59	192.168.1.15	92.250.220.82
80	TCP	88	2010−02−24	02:20:59	192.168.1.29	92.250.220.82
80	TCP	997	2010−02−24	02:20:59	192.168.1.27	92.250.220.82
80	TCP	650	2010−02−24	02:20:59	192.168.1.9	92.250.220.82
80	TCP	298	2010−02−24	02:20:59	192.168.1.46	92.250.220.82
80	TCP	502	2010−02−24	02:20:59	192.168.1.52	92.250.220.82

Traffic Flow Table 3 Traffic Flow Table example for a destination address being targeted by reduced group of host .

PORT	PROTO	KB	TIME		SOURCE	DEST
25	TCP	4660	2010−02−24	02:20:59	192.168.1.2	92.250.220.82
443	TCP	2417	2010−02−24	02:20:59	192.168.1.2	92.250.220.82
443	TCP	1945	2010−02−24	02:20:59	192.168.1.2	92.250.220.82
21	TCP	4206	2010−02−24	02:20:59	192.168.1.1	92.250.220.82
80	TCP	4336	2010−02−24	02:20:59	192.168.1.3	92.250.220.82
110	TCP	2110	2010−02−24	02:20:59	192.168.1.2	92.250.220.82
23	TCP	4257	2010−02−24	02:20:59	192.168.1.1	92.250.220.82
25	TCP	2005	2010−02−24	02:20:59	192.168.1.3	92.250.220.82
993	TCP	2434	2010−02−24	02:20:59	192.168.1.1	92.250.220.82
443	TCP	3270	2010−02−24	02:20:59	192.168.1.2	92.250.220.82
993	TCP	4775	2010−02−24	02:20:59	192.168.1.2	92.250.220.82
22	TCP	690	2010−02−24	02:20:59	192.168.1.3	92.250.220.82

directions. Therefore, for IP address, this is a sequence of bits which is similar to the standard definition.

Considering T_M a M dimensional tree, defined in Section 4.2, a multi-dimensional direction is defined as a tuple of M directions (one for each dimension):

$$\forall n_i \in T_M, n_j \in T_M, n_k \in T_M, n_j \in child(n_i), n_k \in child(n_i)$$
$$n_k \neq n_j \iff direction(n_i, n_k) \neq direction(n_j, n_k) \quad (2)$$

This corresponds to have only one child node per unique tuple of directions. Conceptually, a direction can be any tuples that allows to distinguish child nodes. Practically, it can usually match concrete value like the bit values for IP address or the application subclass for TCP ports. Directions are illustrated in figure 7 using a 3-tuple (application, source and destination IP addresses). The direction *SAME* was introduced to allow a child node to be different from its parent on a subset of dimension values while some remain the same. Preliminary tests show that this limits the number of internal nodes. However, at least one direction of the tuple must not be *SAME*.

5.3 Tree Construction

Intuitively, the tree construction is done by creating a root and then adding nodes or updating existing ones. Based on the directions, a pre-order traversal is done looking for a match to insert the node n_i (line 4 in Algorithm 1). Assuming that the traversal stops on the node n_j, there are different cases:

- if there is a perfect match (dimension values are the same), the activity volume is updated $vol_j \leftarrow vol_j + vol_i$. This is done in line 5 of algorithm 1.
- if n_i is a child n_j, a new child node is created by computing the directions tuple from n_j to n_i (there is not yet a node in this position otherwise the traversal should have continue). This is done in line of algorithm 1.
- otherwise, the traversal has followed the direction but ends in a too specific node (it happens because no all possible internal nodes are cerated for scalability issue, for instance X.Y.Z.0/24 may be a child of X.Y.0.0/16 on the IP address dimension). In this case, a new branching point (internal node) is created from $lcp(n_i, n_j)$. So n_i and n_j are the child nodes and directions are then computed, *i.e.* $direction(n_i, lcp(n_i, n_j))$ and $direction(n_j, lcp(n_i, n_j))$. This is done line 15 of algorithm 1.

Therefore, by construction, every node represents a subspace of its parent according to all dimensions.

Once the tree construction is finished (end of the time window), aggregation takes place. Aggregation is done by traversing the tree in post order to find nodes having a activity volume $vol_i \leq \alpha$. These nodes are aggregated into their parents. By doing this, directions are discarded since they are only needed for the construction and because they may not satisfy equation (2) due to the deletion of internal nodes. Therefore, the k-dimensional space is not divided a priori and the space is not split at regular intervals. This allows, again, to have an irregular granularity over the dimensions for efficiently monitoring of the targeted events.

Algorithm 1 Update Tree $insert\_node(tree, n_i)$

```
1:  if tree is empty then
2:    tree.set_root(n_i)
3:  else
4:    n_i ← tree.search_matching_node(n_i)
5:    if n_j matches all directions then
6:      update n_j volume {Perfect Match}
7:    else
8:      if n_j is n_i child then
9:        {Partial Match}
10:       for dim ∈ n_i do
11:         add n_i to n_j childs
12:       end for
13:       update tree set branching_point parent of n_j
          and n_i
14:     else
15:       branch ← empty node {New Branch case}
16:       for dim ∈ n_j do
17:         branch[dim] ←
            Direction_dim(n_j[dim], n_i[dim])
18:       end for
19:       update tree set branching_point parent of n_j
          and n_i
20:     end if
21:   end if
22: end if
```

6 Online Tree Aggregation Strategies

Since memory consumption grows along with the size of input data, and so the size of the tree, we developed strategies for maintaining the tree structure under a predefined size. Once the number of nodes is higher than MAX_NODES, one of the following strategy is triggered:

- Root aggregation: this strategy performs the simple aggregation (as described in the previous section) from the root

- Least Recently Used (LRU): in this case the least recently used nodes are candidates

These methods are qualified as online because they perform pre-aggregation before the end of the time window for saving memory resources.

6.1 Root Aggregation

Root aggregation algorithm is shown in Algorithm 2. Every time the tree grows over the user defined threshold, the aggregation mechanism explained in Section 5.3 is triggered. It is the most simple solution but not an efficient mechanism since all nodes below the threshold α are aggregated while the objective is to reduce the number of nodes to MAX_NODES. This can be seen in line 5 of Algorithm 2. The cost of the aggregation mechanism triggered in line 6 is $O(n \times log(n))$ [8]. Its worst case complexity is $O(n^2 \times log(n))$ where n is the number of nodes. This is because worst case scenario represents triggering aggregation after every inserted node (For Loop in line 2). Details related to mechanisms to recover from direction incoherency entailed by internal nodes deletion are included in the source files provided (source file containing Tree class definition).

Algorithm 2 Build Tree $T(dimensions, data)$

```
1: tree ← empty_tree
2: for d ∈ data do
3:     node ← build_node(d)
4:     tree.insert_tree(node)
5:     if tree.size > MAX_NODES then
6:         tree.aggregate()
7:     end if
8: end for
```

6.2 LRU Aggregation

Algorithm 3 consists of triggering aggregation on the least recently used node only. The main idea behind this mechanism is to label every node with a timestamped tag that indicates the last time it was used. This is done in Algorithm 4 and used in line 4 in Algorithm 3 for updating the timestamp of nodes which have been gone through when a node is inserted or updated (i.e. its parent and ancestor nodes).

Aggregation is performed only for the least recently used node. To achieve that, a min-max heap is employed [2] structure using as key the timestamped tag present in each node. Algorithm 4 implements this mechanism extending Algorithm 1 functionality for labeling and maintaining the heap structure used for retrieve the LRU node. This corresponds to line 14 of Algorithm 4. This operation is based on updating a min max heap which has a complexity of $O(log_2(n))$ [2]. Assuming n, the number of nodes, the average size of a tree branch is $log(n)$. If every node in the branch has to be updated, an entry on the heap must be modified. Hence the subcost of that operation is $O(log^2(n))$ and in the worst case $O(n \times log(n))$ (a single branch of n nodes). To calculate this complexity, else branch in line 5 of Algorithm 4 is going to be executed. During the last for-cycle (line 14), the list *update_nodes* will contain $log(n)$ elements that corresponding to the explored path to place *node* in *tree*.

After updating the last time used timestamp, the max heap is updated with a complexity of $O(log_2(n))$. So the total complexity is $O(log^2(n) + log(n)) = O(log^2(n))$.

Algorithm 3 Build Tree LRU $T(dimensions, data)$

```
1: tree ← empty_tree
2: for d ∈ data do
3:     node ← build_node(d)
4:     tree.update_lru_tree(node)
5:     if tree.size > MAX_NODES then
6:         candidate ← tree.lru_heap.top()
           Get the top element, candidate to be aggregated
7:         candidate.aggregate()
8:     end if
9: end for
```

Algorithm 4 Update Tree $update\_lru\_tree(tree, node)$

```
1: if tree is empty then
2:     node.ltu ← now
3:     update_nodes ← [node]
4:     tree.set_root(node)
5: else
6:     path ← tree.insert_node(node)
7:     for n in reverse(path) do
8:         n.ltu ← now
9:         now ← now + 1
10:        update_nodes.append(n)
11:    end for
12: end if
13: for n in update_nodes do
14:    tree.ltu_heap.update(n)
15: end for
```

As highlighted in the *else* statement (line 5), the timestamp is updated such that the leaf node (inserted or updated) has a timestamp older that its ancestors (reverse path is constructed during the insertion itself without any additional cost). This ensures that a leaf node is always retrieved and aggregated in line 7 in Algorithm 3. Otherwise, such an element could be an internal node and aggregation may lead to remove entire subtrees.

6.3 Issues

Since online aggregation is done by sequentially reading data, a change of data ordering will produce different results. Inserting the same data instance at the end of the window will not have the same effect than at the beginning since the tree may have already been aggregated many times before. The impact of the data ordering will be evaluated in section 7.3.

7 Evaluation

In this Section we evaluate the following main aspects of the proposed tool:

- Scenarios showing aggregation benefits with multiple types of data:
  - Netflow
  - SIP Messages
  - Geographical coordinates associated to Netflow captures
  - DNS names associated to Netflow captures
- Performance of the proposed strategies of online aggregation
- Order of data impact on aggregation accuracy

For sake of clarity, only partial trees are shown in Figures. Except when mentioned, LRU strategy is applied.

7.1 Data sets

7.1.1 Netflow

NetFlow [9] was developed by Cisco Systems and is supported by many device vendors. Thus, Netflow or other flow-based approaches are now considered as a standard for IP monitoring. A flow record groups multiple packets sharing similar properties and in particular source and destination addresses, protocols and ports. Available information includes useful information like a timestamp, number of packets or bytes exchanged. The interested reader can refer to [9] for further information.

Real Netflow captures were provided by one major ISP in Luxembourg. As assumed to be free of attacks, we also inject a realist attack in the same manner as in [34] for assessing the ability of our approach to catch valuable information about anomalies.

The duration of the capture is 26 days from 01/30/2010 to 02/24/2010 with an average number of flows aroung 60,000/sec. A total of 279815 unique IP addresses using 64470 different UDP and TCP ports are represented.

The following information is extracted:

- Timestamp
- IPv4 Source Address
- IPv4 Destination Address
- TCP or UDP source port
- TCP or UDP destination port
- Traffic Volume in bytes is considered as the activity volume (*vol*)

7.1.2 DNS Data sets

This dataset consists is enhanced compared to the previous on by performing a reverse DNS look up [24, 4] for source and destination IPv4 addresses. Specifically, on reverse DNS look up the pointer to canonical name is retrieved (PTR) [26]. The dataset generated after this method is still representative since every Internet-reachable host should have a name according to [24, 4]. The hierarchical relationship on the DNS dimension is straightforward by using the traditional order between a domain and its subdomain.

7.1.3 Geographical coordinates Data sets

This dataset contains geographical coordinates associated to source and destination IPv4 address of every flow. This is done by using GeoIP database available in [25]. Therefore, the dataset includes the same information as the Netflow one and includes also the latitude and longitude for source and destination IPv4 addresses. Defining dimensions and directions for latitude or longitude is a bit harder as it is not a discrete space. When nodes are inserted in the tree in a way that a new one has to be created, the latter is a rectangle area such that it contains all child nodes using two directions (left and right and top and bottom). This corresponds to compute minimal and maximal values for both the longitude and latitude. Another approach could have considered a taxonomy per continent, country, region, city for example.

7.1.4 SIP Messages

SIP (Session Initiation Protocol) [32] is widely used in VoIP networks. SIP messages are divided into requests and responses. Keywords identify the type of a request such as `INVITE, OPTIONS, REGISTER, NOTIFY` while 3 digits numbers are used for responses (STATUS). The first digit, between 1 and 6, indicates the class (1xx: provisional, 2xx: success, 3xx: redirection, 4xx: client error, 5xx: server error, 6xx: global failure). Therefore, SIP Messages can be classified hierarchically according to the SIP Response Codes and SIP Request

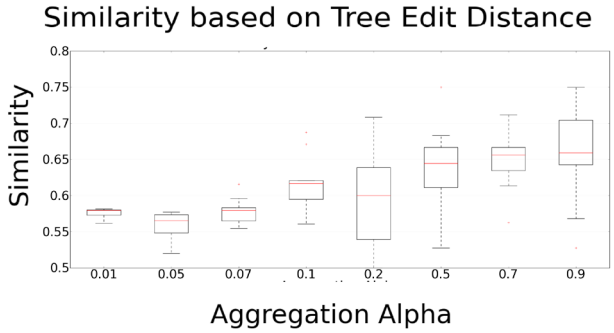


Figure 8: Similarity box plots for trees generated using Netflow samples compared to trees generated after a the same Netflow capture reversed for a 5 minutes window

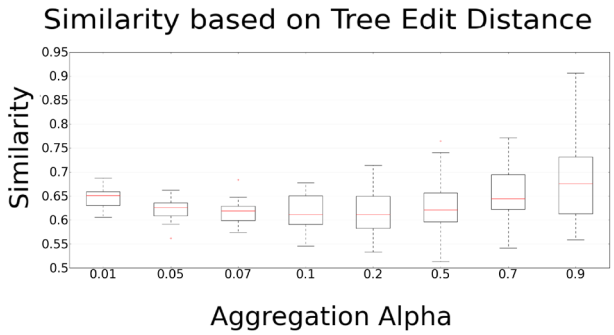


Figure 9: Similarity box plots for trees generated using Netflow samples compared to trees generated after the same randomized flows for a 5 minutes window

Methods described in [32] similar to a taxonomy like in Figure 5 where the first level of division from the root node will be the request and responses types.

This dataset was collected on a local testbed and has the following characteristics:

- 33952 SIP Messages, 17104 Requests, 16848 Responses
- 77 IP Addresses
- 38 Source IP Addresses, 266 source users
- 35 Destinations IP Addresses, 317 destination users.

7.2 Metrics

Trees are among the most common and well-studied combinatorial structures in computer science. The aggregated tree construction is sensitive to the data order. The goal is to measure the degree of similarity for trees built from the same data but in a different order. The Levenshtein distance [22, 7] is usually refereed as edit distance between two strings. It is defined as the minimum amount of operations to transform one string into the other (deletion, insertion and relabeling or substitution). As an example the strings ”sos” and ”sbs” have a distance of 1 by performing one substitution of ”o” into ”b”. This notion of distance was initially defined for strings where every operation has a single cost of 1. As a node is a more complex structure, thus deletions and insertions are always associated to a cost of 1. However substitutions in multidimensional nodes are decomposed into each dimension. According to the formal definition of a M multidimensional tree, a node was defined as $n_i = < \{f_{i_1} \dots f_{i_m}\}, vol_i >$. A distance function per each dimension can calculate the cost of an operation as:

$$\frac{\sum_{k=1}^{M} Distance_k(f_{1_k}, f_{2_k})}{m}$$

Thus, the distance function has to be defined for each dimension. In our case, we define it based on the longest common prefix between two IP addresses or on the longest common path in the taxonomy tree for ports.

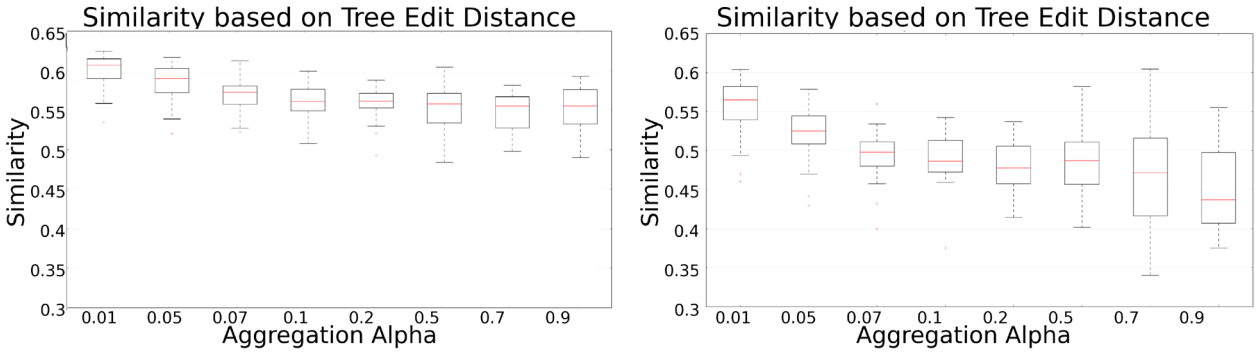
Tree distance can be computed by calculating a distance matrix as the result of the pair wise comparison between the nodes of the trees to compare. Then we calculate the average over the minimal distance of every row and column.

7.3 Order of data impact

In order to evaluate the impact of data order we used the similarity metrics described in Section 7.2. The focus is set on analyzing the distortion of the tree shape after altering the data order.

The evaluation was performed on 3 dimensions: source IPv4 address, destination IPv4 address and TCP port. For strengthen the evaluation, experiments are executed 1000 times with different samples and percentiles (25th, 50th, 75th) as well as minimal and maximal values are then calculated and represented in a box plot graph like Figure 8.

- Normal Sample vs Reversed Sample: This test consists in comparing trees generated from a 5 minutes sample ($\eta = 5min$) of the Netflow dataset and the same data, *i.e.* Netflow records, in the reverse order. In Figure 8, the similarity grows with aggregation ratio α except for small values. This behavior is logic as increasing the aggregating will keep track of global phenomenas which are usually present in the whole sample, `i.e.` not at a specific time unlike local events, and so less sensitive to the data order.



(a) 24 hours offset

(b) 36 hours offset

Figure 10: Similarity - trees generated using Netflow samples compared to trees generated after a Netflow capture from a offset using a 5 minutes window

- Normal Sample vs Random Sample: this test is similar but data are not reversed but randomized. The same observations can be made in Figure 9 even if the impact of α is less visible.

7.4 Scenarios

It is important to note that the similarity is a number between 0 and 1 with 1 when the trees are identical. We can observe from our experiments that the data order has an impact on the aggregation process itself since the value never reaches 1. However, it is important to verify that the similarity value is lower when data differs. Therefore, we can assumed that in most of cases, a similarity higher than 0.6 may reflect similar data but which may have been process in a different order.

7.4.1 Netflow

In order to evaluate the aggregation accuracy, several tests over the Netflow dataset were conducted. The evaluation consists in comparing similar and different traffic flows by observing the similarity variation. From the dataset, 50 time windows ($\eta = 5min$) are randomly picked up from working days between 01/30/2010 and 02/24/2010. Once the tree is built, it is compared with the tree associated to data with a 24 hours or 36 hours offset. Activity at the same time between two working days (24h offset) is usually considered as similar and so should exhibit a higher similarity than activity at a different times between two days (36h offset). Figure 10 shows that the similarity with a 36h offset decreases higher than with the 24h offset, leading to an easier differentiation. Even if the average similarity tends to be close between figures 10(a) and 10(b), the box plots clearly highlight a lower stability with a 36h offset which leads to the same conclusion. With respect to experiments on the data order (section 7.3), values are mainly lower than 0.6 for the 24h offset. Therefore, impact of the data order is lower than the impact of activity variation at the same time between two consecutive days. From this point of view, our approach is thus viable.

To evaluate the aggregation helpfulness in an malicious environment, the following experiment has been conducted:

- injection of DDoS attack directed against web servers running on TCP port 80 with a repeated sequence (3 packets and 128 bytes) sent by burst of 10 packets every 60-120 milliseconds. The attack was injected in two periods of 2 minutes each.
- 60 trees were generated, $T_0 \dots T_{59}$, to summarize network activity ($\eta = 25s, \alpha = 0.05$) by monitoring both the source and destination IP addresses meanwhile
- we compute the edit distance as defined in Section 7.2 between two trees to figure out the attacks: $z[n] = EditDistance(T_n, T_{n+1})$

In figure 11, the two attack occurrences are clearly distinguishable by observing peaks due the high variation of the edit distance when the attacks start and end. Although more advance techniques, as for example machine learning based methods, could be leveraged and evaluated on other kinds of attacks, such a complete evaluation is not in the scope of the paper.

7.4.2 SIP Messages

In order to evaluate Aggregation accuracy for SIP messages, we tested the aggregation mechanism to detect high volume of error messages which is helpful for figuring out anomalies. As the testbed was locally made, we were able to generate anomalies (bypassing the registration). From our traces, we extracted one tree before

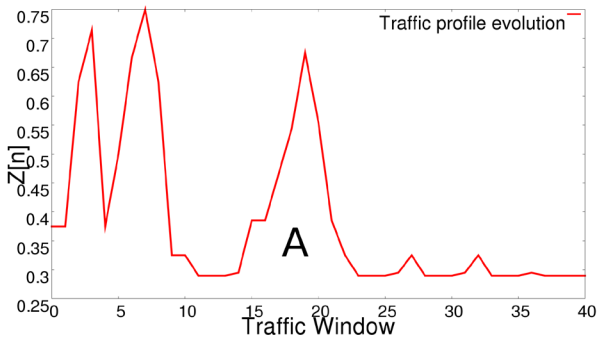


Figure 11: Distance Function Evaluation for aggregated trees generated from Flow capture of a TCP Flood attack

(figure 12) and after (figure 13) the problem occurs. In particular, a node containing an error (*ProxyAuthenticationRequired.ClientError*) appears. A simple counter per message type would also be able to detect it. However, the tree informs that a relatively big portion of traffic (7.32%) is concerned by this error on a single subnet, 192.168.1.0/29. It is helpful for identifying meanwhile the sources from where the errors come and so for recovering afterwards. Although this is a example to highlight the benefits of our approach, more advanced technique for analysis are possible. Retrieving automatically new nodes or only nodes representing errors could help an administrator.

From a general point of view, the trees produced by *MAM* can be manually or automatically analyzed. For manual analysis, this eases the understanding as the information is compacted and is sorted from top to bottom by the impact (accumulated volume of activity). Therefore, if an observation looks abnormal at a high level in the tree, finding the root cause requires to follow the parent-child relation. When reading the tree, the user can easily follow the interesting branches like those related to erroneous responses. Future visualizations techniques could propose a dynamic navigation within the tree by providing information on demand. Experiments where conducted using ($\eta = 120s - 360s, \alpha = 0.05$)

7.4.3 DNS names associated to netflow captures

For DNS names, the dimension *Domain*, representing a FQDN, takes in account top level domains and sub domains separated by dots. Each dot is another split between a parent and its child in the hierarchical tree. Figure 14 illustrates the approach using the dataset described in Section 7.1.2 by limiting our study to source IP addresses associated to the french top level domain (*.fr* domains). We also consider the domains of destination IP addresses. For example we can notice that most of traffic originating from *.fr* to *.com* domains targets *deezer*.

Assuming a fast-flux botnet [30], the DNS names and IP addresses should be monitored while considering the inverse of the DNS Time-To-Live should be considered as activity volumes. Using a short TTL and a large set of IP addresses (usually compromised machines), the attacker changes frequently the IP address associated to the domain hosting malware. By using the inverse of TTL, such behavior will appear using *MAM* by highlighting the problematic domain and IP addresses or subnets. Experiments were conducted summarizing network activity using parameters ($\eta = 0s - 300s, \alpha = 0.05 - 0.1$)

7.4.4 Geographical coordinates associated to Netflow captures

Evaluating results of geographical coordinates aggregation requires a more sophisticated display structure. Representing areas (ranges of geographical coordinates) in a map shows more information rather than displaying a tree with coordinates. Hence, the Google Maps API[1] was used to show a real map. Only flows having the origin and destination in a fixed area (France or Europe) are extracted. The color opacity reflects the aggregated traffic load by area. Figure 15(a) highlights that Luxembourg is the center of the most inner square which is natural as the dataset we used is provided by a Luxembourg operator, and so most of flows are from and to hosts in this country. As an other illustrative example, a traffic diagram of France is presented in figure 15(b) which logically highlights the concentration of network traffic around Paris. These experiments show, that without specifically guiding the aggregation by pre-defining geographical zones or using cities, our method is able to figure out automatically important parts, well known areas (around Paris) as well as others (for example in South of France in figure 15(b)).

As another illustrative case, a content provider may track his user location for optimizing the placement of his servers or to provide locally profiled content.

7.5 Performance

Several benchmarks have been done in a 8 core Intel(R) Xeon(R) CPU E5345@ 2.33GHz. They are based on the running times regarding the number of nodes in a tree. The Netflow dataset was used by considering source and destination addresses as well as destination ports. As introduced in Section 6, the worst case computational complexity of the LRU strategy is $n \times log(n)$. In figure 16(b), the empirical results are quite lower. This behavior can be explained because the worst case scenario is a pathological case that is not usually common found in

[1] https://developers.google.com/maps/

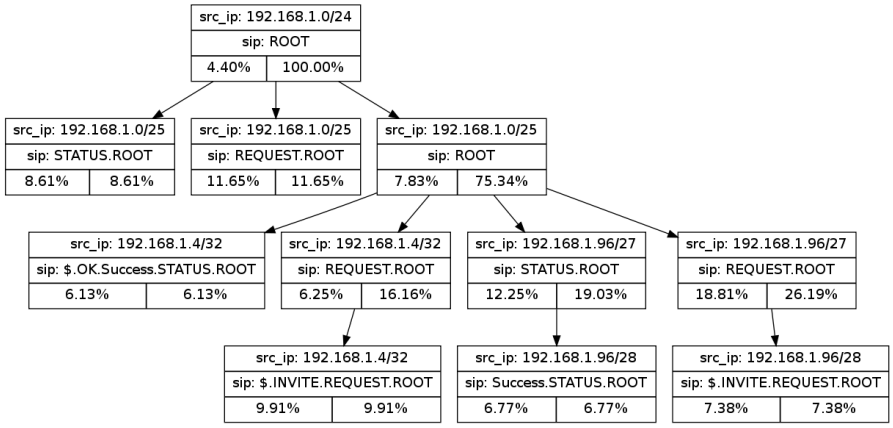


Figure 12: Aggregation for SIP conversations on networks 192.168.1.0/24, dimensions Message Type and source IP Address - no error (Partial view)

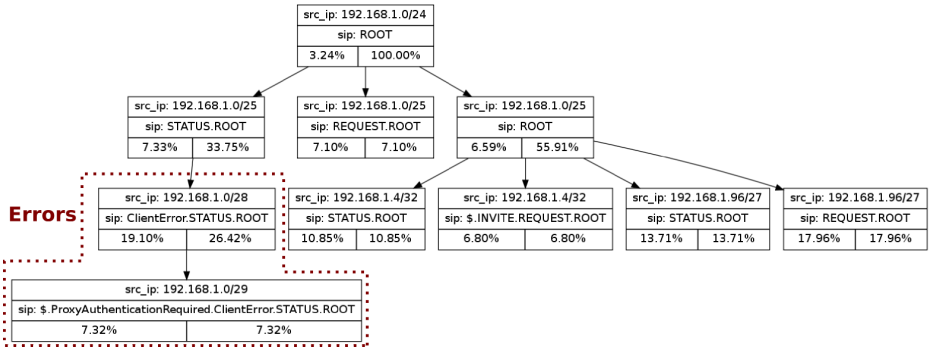


Figure 13: aggregation for SIP conversations on networks 192.168.1.0/24, dimensions Message Type and source IP Address - with errors (Partial view)

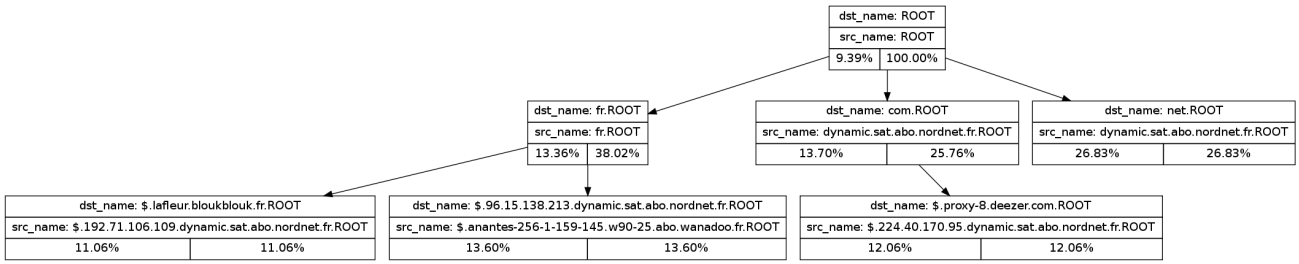
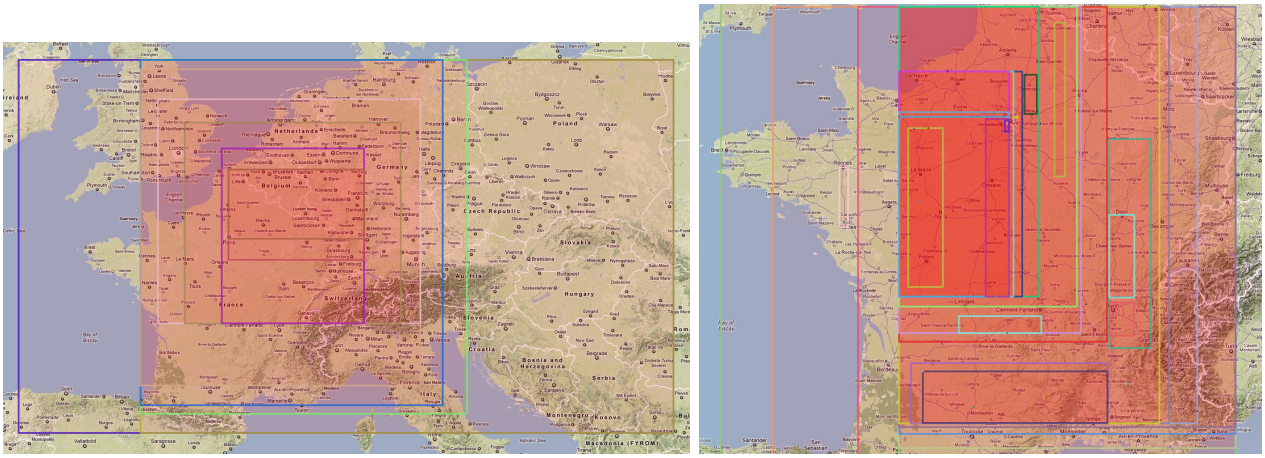


Figure 14: DNS names aggregation using as dimensions PTR records for source and destination (Partial view)



(a) Internal Traffic aggregation for Europe

(b) Internal Traffic aggregation for France

Figure 15: Aggregation example for Geographical coordinates associated to netflow captures aggregation for traffic within Europe.

the captures. Furthermore, this is coherent with the average computational complexity ($log^2(n)$) introduced in Section 6.

Figure 16(a) highlights similar results for the simple aggregation at the end of time windows. Therefore, algorithms are equivalent from a computational power side. Applying the LRU strategy does not entail any notable overhead. However, from a memory point of view, the LRU strategy saves a lot of memory space since the tree size is bounded by *MAX_NODES*.

The average size of the trees before and after the aggregation is given in table 1 when no LRU strategy is used during our previous experiments. First, using LRU strategy strongly limits the memory usage since the tree size before aggregation is quite higher 1000 (default value for MAX_NODES when LRU is applied) except for SIP since the dataset was generated on a local testbed. Therefore, in real scenarios, memory usage is highly reduced during the tree construction, which improves the scalability. α set to 5% provides good results in previous experiments while the number of nodes after the aggregation at the end of a time window is drastically reduced. The tree size never exceeds 90 nodes. Therefore, scalability is again increased from a storage point of view as well as for doing analysis (manual or automated) on the final built trees.

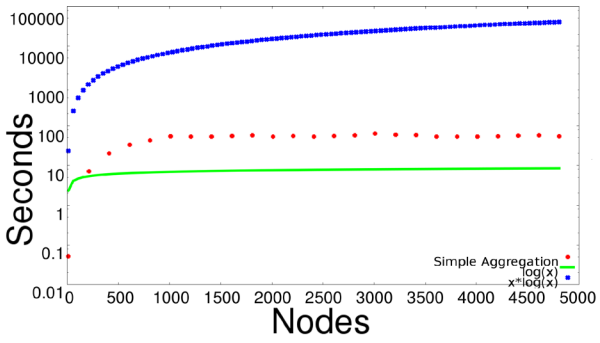
Support real-time flow aggregation is validated by associating Table 1 and figure 16(b), 3288 netflow records (average number of flows in a 5 minutes window) are always aggregated in less than 100 seconds, this allows to aggregate flows in real-time. Memory consumption can be computed as the sum of the dimensions data type size and the tree final size. For GPS coordinates every node holds 32 bits for 2 integers. Regarding the data structure 8 additional bytes and a list of pointers (4 bytes in 32bits) to the children are required. With an average tree size of 14664 nodes tree this results in $14664 \times (4+8+8) = 286.4MB$.

	Average tree size before aggregation	Average tree size after aggregation
Netflow	3288	90
DNS	8600	53
SIP	1077	18
Geographical	14664	45

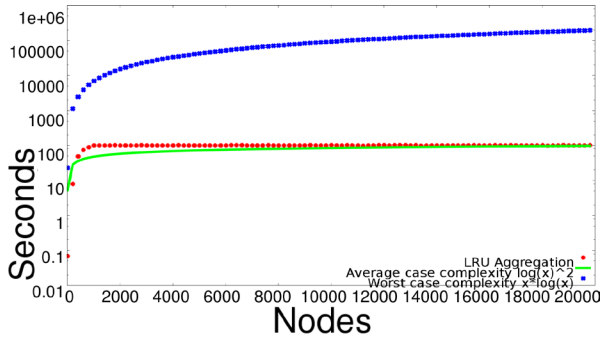
Table 1: Tree size reduction using aggregation (average on all time windows) with $\alpha = 0.05$

8 Conclusion

In this paper, a multidimensional aggregation is introduced which is able to handle the multiple dimensions without having any predefined order. The aggregation increases the scalability by compressing data and by creating summarized outputs which are smaller and easier to analyse. Moreover, the aggregation is guided by the nature of the events to monitor, *i.e.* a configurable activity volume. This leads to split the space into partition of different sizes. Hence, distributed coordinated behaviors can be observed unlike traditional approaches relying on a regular space division which does not necessarily reflect the current distribution on the network activity. Based on formal definitions, this paper highlights the use of common dimensions related to network administra-



(a) Time performance for simple aggregation



(b) Time performance for LRU aggregation

Figure 16: Time performance for different strategies

tion like IP addresses, applications, DNS, but could be extended to any other kinds of dimensions such as polar coordinates by extending the feature class and implementing the requested methods in the API. We provide an open source tool[2] to export png file or Google Maps and provide ready to use implementations of IP addresses, applications, DNS and GPS features.

Experiments focus on sample scenarios where the aggregation reveals important facts or changes in the network. Theoretical as well as practical complexities were studied, in particular to prove the benefits of using a LRU strategy for improving the scalability of our approach. Furthermore, the data order problem due to such a process was evaluated and the results highlight a negligible impact. Similarly to our previous work on single dimension aggregation [34], aggregated trees are also good candidates for data mining algorithms and not exclusively dedicated to being processed by a human expert. Therefore, our next work will focus on this topic and on the definition of other dimensions like IPv6 addresses.

Acknowledgement: This work is partly funded by OUTSMART, a European FP7 project under the Future Internet Private Public Partnership programme. It is also supported by MOVE, a CORE project funded by FNR in Luxembourg.

[2] `http://lorre.uni.lu/~jerome/files/mam.tar.gz`

References

[1] ANTONAKAKIS, M., PERDISCI, R., LEE, W., VASILOGLOU, II, N., AND DAGON, D. Detecting malware domains at the upper dns hierarchy. In *Proceedings of the 20th USENIX conference on Security* (Berkeley, CA, USA, 2011), SEC'11, USENIX Association, pp. 27–27.

[2] ATKINSON, M., SACK, J., SANTORO, N., AND STROTHOTTE, T. Min-max heaps and generalized priority queues. *Communications of the ACM 29*, 10 (1986), 996–1000.

[3] BALL, R., FINK, G., AND NORTH, C. Home-centric visualization of network traffic for security administration. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security* (2004), ACM, pp. 55–64.

[4] BARR, D. RFC 1912: Common DNS operational and configuration errors.

[5] BENTLEY, J. Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9 (1975), 509–517.

[6] BILGE, L., KIRDA, E., KRUEGEL, C., AND BALDUZZI, M. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2011)* (2011), The Internet Society.

[7] BILLE, P. A survey on tree edit distance and related problems. *Theoretical computer science 337*, 1 (2005), 217–239.

[8] CHO, K., KAIZAKI, R., AND KATO, A. Aguri: An aggregation-based traffic profiler. In *Quality of Future Internet Services* (2001), Springer, pp. 222–242.

[9] CLAISE, B., SADASIVAN, G., VALLURI, V., AND DJERNAES, M. Rfc 3954: Cisco systems netflow services export version 9. Tech. rep., 2004.

[10] DRAGO, I., SADRE, R., AND PRAS, A. Report of the third workshop on the usage of netflow/ipfix in network management. *J. Netw. Syst. Manage. 19*, 4 (2011), 529–535.

[11] ESTAN, C., KEYS, K., MOORE, D., AND VARGHESE, G. Building a better netflow. *ACM SIGCOMM Computer Communication Review 34*, 4 (2004), 245–256.

[12] FENWICK, P. A new data structure for cumulative frequency tables. *Software: Practice and Experience 24*, 3 (1994), 327–336.

[13] FINKEL, R., AND BENTLEY, J. Quad trees a data structure for retrieval on composite keys. *Acta informatica 4*, 1 (1974), 1–9.

[14] FRANÇOIS, J., WANG, S., BRONZI, W., STATE, R., AND ENGEL, T. BotCloud: Detecting Botnets Using MapReduce. In *Workshop on Information Forensics and Security (WIFS)* (Foz do Iguaçu, Brazil, December 2011), IEEE, Ed.

[15] FRANÇOIS, J., WANG, S., STATE, R., AND ENGEL, T. BotTrack: Tracking Botnets using NetFlow and PageRank. In *IFIP/TC6 NETWORKING 2011* (Valencia, Spain, May 2011), Springer, Ed.

[16] FULLER, V., LI, T., YU, J., AND VARADHAN, K. Rfc1519: Classless inter-domain routing (cidr): an address assignment and aggregation strategy.

[17] GIURA, P., AND MEMON, N. Netstore: An efficient storage infrastructure for network forensics and monitoring. In *Recent Advances in Intrusion Detection* (2010), Springer, pp. 277–296.

[18] GRAY, J., CHAUDHURI, S., BOSWORTH, A., LAYMAN, A., REICHART, D., VENKATRAO, M., PELLOW, F., AND PIRAHESH, H. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov. 1*, 1 (1997), 29–53.

[19] HU, Y., CHIU, D.-M., AND LUI, J. C. S. Entropy based adaptive flow aggregation. *IEEE/ACM Trans. Netw. 17*, 3 (2009), 698–711.

[20] KAIZAKI, R., CHO, K., AND NAKAMURA, O. Detection of denial of service attacks using aguri. In *International Conference Telecommunications, Beijing China* (2002).

[21] KRIZAK, P. Log analysis and event correlation using variable temporal event correlator (vtec). In *Proceedings of the 24th international conference on Large installation system administration* (2010), LISA'10, USENIX Association.

[22] LEVENSHTEIN, V. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (1966), vol. 10, pp. 707–710.

[23] LIAO, Q., BLAICH, A., STRIEGEL, A., AND THAIN, D. Enavis: enterprise network activities visualization. In *Proceedings of the 22nd conference on Large installation system administration conference* (2008), LISA'08, USENIX Association, pp. 59–74.

[24] LOTTOR, M. RFC 1033: Domain administrators operations guide.

[25] MAXMIND, L. www.maxmind.com, 2007.

[26] MOCKAPETRIS, P. Rfc 1035: Domain names - implementation and specification.

[27] MOREIRA MOURA, G., SADRE, R., SPEROTTO, A., AND PRAS, A. Internet bad neighborhoods aggregation.

[28] OBERHEIDE, J., GOFF, M., AND KARIR, M. Flamingo: Visualizing internet traffic. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP* (2006), IEEE, pp. 150–161.

[29] PAREDES-OLIVA, I., BARLET-ROS, P., AND SOLÉ-PARETA, J. Portscan detection with sampled netflow. *Traffic Monitoring and Analysis* (2009), 26–33.

[30] PERDISCI, R., CORONA, I., DAGON, D., AND LEE, W. Detecting malicious flux service networks through passive analysis of recursive dns traces. In *Annual Computer Security Applications Conference - ACSAC* (2009), IEEE Computer Society.

[31] PLONKA, D., AND BARFORD, P. Context-aware clustering of dns query traffic. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), IMC '08, ACM.

[32] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. Rfc 3261 sip: Session initiation protocol.

[33] TOLEDO, J., ADRIGHEM, V., GHETTA, R., MANN, E., AND PETERS, F. Etherape: a graphical network monitor, accessed on 05/14/2012.

[34] WAGNER, C., FRANCOIS, J., ENGEL, T., ET AL. Danak: Finding the odd! In *Network and System Security (NSS), 2011 5th International Conference on* (2011), IEEE, pp. 161–168.

[35] WEIGERT, S., HILTUNEN, M., AND FETZER, C. Community-based analysis of netflow for early detection of security incidents. In *Proceedings of the 25th Large Installation System Administration Conference (LISA'11)* (2011), USENIX Association.

[36] YURCIK, W. Visualizing netflows for security at line speed: the sift tool suite. In *Proceedings of the 19th conference on Large Installation System Administration Conference - Volume 19* (2005), LISA '05, USENIX Association.

On the Accurate Identification of Network Service Dependencies in Distributed Systems

Barry Peddycord III, Peng Ning
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
Emails: {bwpeddyc, pning}@ncsu.edu

Sushil Jajodia
Center for secure Information Systems
George Mason University
Fairfax, VA 22030
Email: jajodia@gmu.edu

Abstract

The automated identification of network service dependencies remains a challenging problem in the administration of large distributed systems. Advances in developing solutions for this problem have immediate and tangible benefits to operators in the field. When the dependencies of the services in a network are better-understood, planning for and responding to system failures becomes more efficient, minimizing downtime and managing resources more effectively.

This paper introduces three novel techniques to assist in the automatic identification of network service dependencies through passively monitoring and analyzing network traffic, including a logarithm-based ranking scheme aimed at more accurate detection of network service dependencies with lower false positives, an inference technique for identifying the dependencies involving infrequently used network services, and an approach for automated discovery of clusters of network services configured for load balancing or backup purposes. This paper also presents the experimental evaluation of these techniques using real-world traffic collected from a production network. The experimental results demonstrate that these techniques advance the state of the art in automated detection and inference of network service dependencies.

Tags: network services, dependency, service clusters, research

1 Introduction

Enterprise networks are large and complicated installations that feature intricate and often subtle relationships between the individual nodes in the system. They are usually composed of multiple *network services* that provide the core functionalities to the rest of the network, such as authentication and database management services. Many services deployed in a network depend on additional services hosted in the same or other networks. Should a service go down, for example, due to a system failure, those that depend on it will also be affected. Understanding the dependencies of mission-critical services in distributed systems enables system administrators to better prepare for and respond to network failures and attacks [9].

For example, a typical online banking application usually adopts a user front-end hosted on a web server, which depends on a database service that stores user account information as well as an authentication service that verifies the users who would like to use the online banking application. If the database or the authentication service goes down, the end users will not able to use the online banking application. Knowing the dependencies between these services will enable a system administrator to quickly diagnose the problem and recover the online banking application when system failures occur.

Production networks, however, are seldom this simple, and identifying the dependencies between network services in practice is a non-trivial task. Besides being large and complex, a production network is also dynamic, as nodes are added, removed, and repurposed within the network. In a large enterprise network, it is difficult to derive network service dependencies by only relying on the understanding of the network by experienced administrators. To address this problem, several techniques and tools have been developed to help system administrators identify and monitor network service dependencies efficiently and automatically.

1.1 Related Work

A naive approach to discovering network service dependencies is to manually investigate the configuration files of network services [12]. However, this is time-consuming, error-prone, and unable to keep up with changes in the network configuration. Researchers have proposed several automated approaches to determine such dependencies by analyzing the network traffic be-

tween network services. These approaches can be divided into *host-based* and *network-based* approaches.

Host-based approaches work by installing an agent on the nodes in the network to monitor the behavior of the processes running on it. Macroscope, for example, uses an agent that monitors the network interface to identify the processes that produce traffic as well as the dependencies between network services [12]. Similarly, Magpie tracks the execution paths of applications within a computer and follows them across the network to discover dependencies [4]. Pinpoint monitors the traffic path within the distributed system [5], while X-trace uses this traffic to develop trees of events required to execute a task. Constellation leverages distributed installation of agents and distributed machine learning to identify dependencies [3].

While these approaches are often accurate and effective, system administrators generally prefer to avoid installing extra software in their production environments due to concerns regarding security, performance, or administrative overhead, which poses a barrier to their adoption. Recent efforts have instead explored *network-based* approaches that treat each host as a black box and passively analyze the network traffic between them.

Sherlock, for example, identifies dependencies based on the frequency of co-occurrences of network traffic within a small time window [2]. eXpose further utilizes Sherlock's intuition, using statistical measures to filter out traffic that is less likely to lead to dependencies [8]. Orion identifies dependencies using the delay distribution of network traffic [6]. Another approach applies fuzzy-logic algorithms to classify dependencies by building an inference engine from the traffic [7]. NSDMiner uses nested network traffic flows to identify dependencies with similar detection rates but much lower false positives than the earlier network-based approaches [10].

In spite of these earlier efforts, network-based approaches to discovering network service dependencies are still susceptible to false positives and false negatives. Without access to application-level information about the traffic, it is difficult to separate which traffic is caused by a dependency relationship and which traffic is simply coincidental. In addition, because the dependencies are extracted entirely from network traffic, the dependencies of infrequently used services can be missed due to lack of data. It is highly desirable to have more effective approaches that can also identify the dependencies involving infrequently used network services.

1.2 Our Contribution

In this paper, we develop several novel techniques to address the above problems, which are then used to extend NSDMiner [10], the most recent addition to the network based approaches for the discovery of network service dependencies. While we have developed and released an open source implementation of NSDMiner [11] concurrently with this paper, the algorithms discussed are applicable to any implementation.

Firstly, we provide a new ranking mechanic that can more accurately identify true network service dependencies. Note that all dependency candidates inferred from network traffic could be due to either true network service dependencies or coincidental network activities. In this paper, we introduce a new logarithm-based ranking scheme, which considers not only the correlation between related network flows, but also the implication of coincidental network traffic. As a result, this ranking scheme can rank dependency candidates more accurately and consistently. In other words, compared with NSDMiner, it is more likely for the new scheme to rank the dependency candidates due to true network service dependencies higher than false ones.

Secondly, we develop a novel technique to infer network service dependencies for which very little explicit traffic exists. There are infrequently used network services, which do not produce enough traffic to allow their dependencies to be inferred accurately. We observe that many network services, though possibly deployed for different purposes, share similar configurations. For example, the same Apache HTTP server software may be used (on two different servers) to host both a popular website and an infrequently visited academic web server. Despite the separate installations and different purposes, these two services and the network services supporting them are likely to share some similarities. Leveraging this observation, we develop a technique that attempts to identify groups of similar services, extract the common dependencies of members of the group, and then infer the dependencies of the individual services, particularly those involving infrequently used services, from the common dependencies.

Finally, we introduce a new approach to identify redundant *service clusters* in a network. A service cluster consists of a set of identical network services, which is often used for load-balancing or fault-tolerance purposes. As pointed out in [6, 10], service clusters introduce additional challenges to the network-based automatic discovery of service dependencies, since different service instances will split the network traffic intended for getting certain services and thus make the dependency inference more difficult. In this paper, we develop a new way to automatically infer service clusters. This approach is inspired by an observation of load-balancing service clusters, i.e., services in a service cluster are often accessed with similar frequencies. We thus develop an approach to identify service clusters by observing what services are consistently contacted in groups.

We have implemented these techniques as extensions to NSDMiner, a tool that has been demonstrated to be promising for automated discovery of network service dependencies recently [10]. We have performed experimental evaluation with network traffic collected from our department network. The experimental results demonstrate that our new ranking scheme can further reduce the false positive rate compared with NSDMiner with comparable detection rate. In addition, our approach for inferring network service dependencies can further reduce the false negative rate, and our approach for identifying service clusters is able to identify most of the service clusters in our network.

The rest of this paper is organized as follows: Section 2 presents some preliminary discussion of network services and service dependencies, as well as a brief introduction to NSDMiner. Section 3 describes the proposed techniques in detail. Section 4 discusses our experimental results. Section 5 concludes this paper and points out some future research directions. Following the paper, we provide an appendix that highlights implementation concerns and best practices for operators who may wish to use NSDMiner in their own networks.

2 Preliminaries

2.1 Network Services and Network Service Dependencies

A *network service* is a process running on a computer in a network that listens on a port for connections from client processes running on other computers, and provides a service over the network. A network service can be represented by a triple $(ip, port, protocol)$, which specifies the machine the service is hosted on, the type of application, and the network protocol (TCP or UDP). Intuitively, given two network services A and B, A *depends on* B if A accesses resources on B, either indirectly or directly, to complete its own tasks. Such a dependency is denoted $A \rightarrow B$, where A is referred to as the *depending* service and B is referred to as the *depended* service [6]. (Some previous work refers to the depending service and the depended service as the *dependent* service and the *antecedent* service, respectively [9].)

Network service dependencies fall into two categories based on whether A accesses B's resources indirectly or directly, referred to as *remote-remote* and *local-remote* dependencies, respectively [6]. A remote-remote dependency $A \rightarrow B$ is a dependency where a client cannot access A until it has first accessed B. For example, many web services depend on the DNS service. In order for a client to access a web server via its host name, it usually needs to contact a DNS server to translate the web server's host name to its IP address.

A local-remote dependency $A \rightarrow B$ is a dependency relationship where A directly contacts B during the course of its operation. In production networks, this is a much more common dependency relationship [10]. Indeed, production networks typically expose a few public applications or services, which are supported by many internal, unexposed services that provide the core functionality. This paper focuses on local-remote dependencies.

It is also possible for one network service to depend on another service running on the same host, or for dependencies to exist between virtual machines on a single host. In this case, the communication will be done over a channel such as the loopback interface [1]. However, such dependencies do not produce any visible network traffic, and cannot be detected by passively observing the traffic. Our approach is therefore directed at inter-host network service dependencies.

2.2 The NSDMiner Approach

We first provide a brief overview of NSDMinder [10] below, since our techniques are developed as its extensions.

NSDMiner takes a list of *network flows* as its input. A network flow is a continuous, directed stream of packets between two processes running on different hosts. Each network flow is represented as a tuple of seven attributes *(StartTime, EndTime, SourceIP, SourcePort, Protocol, DestIP, DestPort)*. NSDminer mainly considers TCP flows and UDP flows. A TCP flow represents the traffic of a TCP session, and can be identified by locating the 3-way handshake and 4-way handshake or reset that signal the beginning and the end of the session. A UDP flow between two services consists of a continuous stream of UDP packets where the delay between two consecutive packets is no greater than a certain threshold.

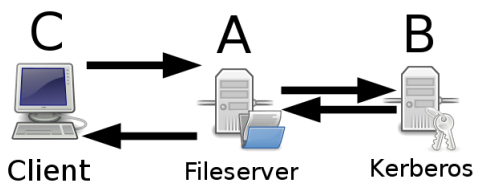


Figure 1: Nested network flows when $A \rightarrow B$.

NSDMiner utilizes a common pattern observed in network applications, illustrated in Figure 1. In general, when a client C accesses a network service A, the network flow between the two will usually be open for the entire duration of the client session. If the network service depends on another service B, it usually contacts B while the client session is running, producing a flow between A and B, nested within the flow from C to A.

To harness this observation, for each observed flow from C to A, NSDMiner counts the number of flows originating from A and occurring during the flow from C to A. NSDMiner models this interaction in a *communication graph*, where each node represents a network service with a weight identifying how many times it occurs as the destination of a flow, and each directed edge represents a *dependency candidate* between two network services with a weight indicating the number of times the nested behavior occurs.

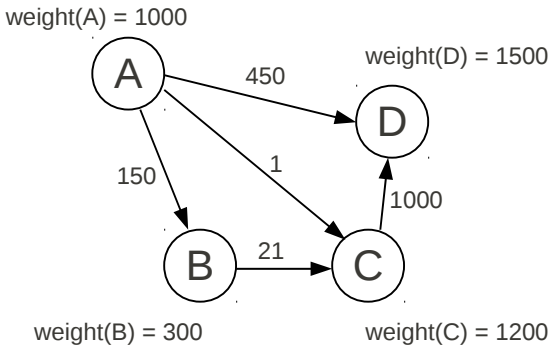


Figure 2: A communication graph produced by NSDMiner.

Figure 2 shows a communication graph produced by NSDMiner where A, B, C, and D are all services running in a production network. Take nodes A and B and the edge $A \rightarrow B$ as an example. This indicates that the network service A has been accessed 1,000 times, and A further used service B 150 times out of these accesses. B was accessed 300 times, 150 of of which come from A and another 150 coming from client machines which are not represented in the graph.

Because multiple network services can be hosted on the same physical machine, it is possible that a flow from A to B may occur during multiple overlapping client sessions, thus creating ambiguity in deriving the weights in the communication graph. NSDMiner uses two modes to mitigate this ambiguity: the *exclusive mode*, where overlapping client sessions are simply discarded, and the *shared mode*, where each of the overlapping sessions is given an equal fraction of a count.

Given a dependency candidate $A \rightarrow B$, the ratio $\frac{weight(A \rightarrow B)}{weight(A)}$ is used as the *confidence* of this candidate. Intuitively, the higher this confidence is, the more likely it is for A to depend on B. NSDMiner directly uses the ratio-based confidence to prune false dependencies due to coincidental traffic; any dependency candidates whose confidence falls below a user-specified threshold will be pruned from the graph. A lower threshold will have greater coverage of true dependencies, but will also introduce potentially more false positives. For the sake of presentation, we call this approach the ratio-based (confidence) ranking scheme.

3 New Techniques for Discovering Network Service Dependencies

In this section, we give the details of the proposed techniques. We first present a logarithm-based ranking scheme to assign confidence values to dependency candidates and rank them more appropriately. This scheme enables us to further reduce false positives beyond the original NSDMiner. We then describe our approach for identifying dependencies of network services for which very little traffic exists, improving the overall detection rate. Finally, we introduce an approach towards automatically discovering clusters of network services, which allows us to aggregate the output of NSDMiner to more effectively identify network service dependencies.

3.1 Logarithm-Based Ranking

Let us first understand the limitations of the ratio-based ranking scheme in the original NSDMiner with the assistance of Figure 3, which shows the confidence of each dependency candidate in Figure 2 as calculated by the ratio-based ranking scheme.

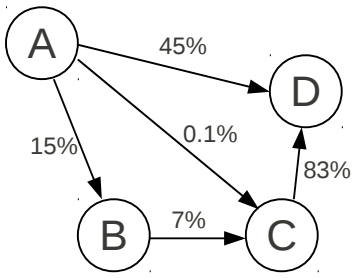


Figure 3: Ratio-based ranking for the graph in Figure 2.

The ratio-based ranking scheme has two major limitations. First of all, it does not take the magnitude of the weights into consideration when assigning confidence values. Consider the dependency candidate $A \rightarrow D$, where A accesses D almost every other time when A itself is accessed. This is very consistent behavior out of the 1,000 accesses to A, which is more likely to be caused by a dependency relationship than coincidental traffic. Note that it only has a confidence value of 45%. Imagine that additional traffic is collected, $weight(A)$ increases to 10,000, and $weight(A \rightarrow D)$ increases to 4,500. Note that the confidence still remains at 45%, though intuition suggests that it would be even more unlikely for such a

high number to be produced by coincidence. Because a raw ratio is used, the magnitude of the two weights have no effect on the final confidence value of the dependency candidate.

Secondly, the ratio-based ranking assumes that the confidence in a dependency relationship scales linearly with the weight of the edge. In other words, it assumes that each observation of a nested connection in support of a dependency candidate contributes equally to its confidence value. However, after a certain threshold, it is no longer reasonable to assume that additional observations are caused by coincidence. At this point, the dependency candidate should have already been assigned a high confidence value. For example, consider Figure 2, where $weight(A) = 1{,}000$, $weight(A \rightarrow C) = 1$, $weight(C) = 1{,}200$, and $weight(C \rightarrow D) = 1{,}000$. Intuitively, 100 new observations in support of $C \rightarrow D$ would have almost no effect on the confidence of $C \rightarrow D$, while the same number of observations for $A \rightarrow C$ would have much more impact on the confidence of $A \rightarrow C$.

These two issues suggest that the ratio-based ranking scheme does not provide the best estimate of the probability of a dependency candidate being true. We propose to adopt a *logarithm-based* ranking mechanic that addresses these issues. Specifically, we propose to define the *confidence of a dependency candidate* $A \rightarrow B$ as $log_{weight(A)} weight(A \rightarrow B)$. Figure 4 shows the logarithm-based confidence values for the same scenario in Figure 2. Note the changes in confidence values in comparison with Figure 3.

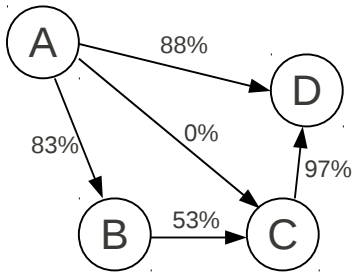


Figure 4: Logarithm-based ranking for the graph in Figure 2.

Both of the above concerns are addressed by the following two properties of the logarithm function that hold for all $1 < a < b < c < d$:

- If $\frac{a}{c} = \frac{b}{d}$, then $log_a c < log_b d$;
- If $\frac{c}{d} - \frac{b}{d} = \frac{b}{d} - \frac{a}{d}$, then $log_d c - log_d b < log_d b - log_d a$.

The first property ensures that when the ratio between the weights of a node and the outgoing edge from the node remain constant, the dependency candidates with greater weights will be assigned higher confidence values than those with lower weights. This reflects the intuition that having more data removes the effect of random coincidence and that the confidence value given to a dependency candidate should take the amount of data collected into account. This has the effect of giving the more obvious dependency candidates higher confidence, preventing them from being filtered out when pruning candidates.

The second property helps reject the notion that the confidence of a dependency candidate scales linearly with the observed traffic, by giving more weights to earlier observations of dependency behavior, and slowly reducing the growth rate as more observations are made. With the ratio-based ranking scheme, both true and false dependency candidates are all clustered at very low confidence values, especially when large amounts of data are collected [10]. This requires the usage of very low filtering thresholds to avoid removing true positives from the results, which unfortunately will also retain a good number of false positives. The logarithm-based ranking scheme puts more disparity between these candidates, making true positives with less traffic more resistant to filtering. As a result, it improves the detection rate and at the same time reduces the false positive rate.

3.2 Inference of Service Dependencies

Our second contribution is an approach for inferring the dependencies of network services that are used infrequently and have insufficient observed traffic associated with them. While many services in a network are used regularly, some services are only accessed occasionally, or at least accessed infrequently during the data collection aimed for discovering network service dependencies. This introduces a great challenge to all automated approaches that attempt to discover network service dependencies by analyzing the activities of these services.

In the following, we develop an approach that uses additional properties in the network beyond network traffic to infer such obscure network service dependencies.

Our approach uses the overlap between the depended services of different network services to infer missing dependencies. When two depending services have several depended services in common, even if the confidence values associated with their respective dependency candidates are low, the overlap itself provides evidence that can be used to make inferences about poorly ranked or potentially missing true dependencies.

While it is not unusual for two network services to share some depended services by coincidence, it is highly unlikely that there will be a substantial overlap in the depended services between them if they are not related. We therefore use the amount of overlap to quan-

tify the similarity between the two services. If a heavily used service and an infrequently used service have a large number of depended services in common, this is likely not the result of a coincidence. We harness this observation to boost the confidence values of the dependency candidates involving infrequently used services.

Examining the similarity between service dependencies using pairs of (depending and depended) services is sensitive to errors. To mitigate the problem, we instead identify groups of network services with large amounts of overlap in terms of their depended services. The members of these groups will have certain common depended services that most of them *agree* on. If most members of the group agree on a particular service as a depended service, we interpret this observation as evidence to infer that the other group members also depend on this service, and add new dependency candidates accordingly.

Given two network services A_1 and A_2, we define the *similarity* between these two services (w.r.t. their depended services) as $Sim(A_1,A_2) = \frac{|\mathscr{D}(A_1)\cap\mathscr{D}(A_2)|}{max(|\mathscr{D}(A_1)|,|\mathscr{D}(A_2)|)}$, where $\mathscr{D}(A_1)$ and $\mathscr{D}(A_2)$ are the sets of services that A_1 and A_2 depends on, respectively. Two services A_1 and A_2 are *similar* if (1) they use the same protocol with the same port and (2) $Sim(A_1,A_2)$ is greater than a threshold.

To model the similarity between services systematically, we build a *similarity graph* based on the communication graph produced by NSDMiner. The nodes in a similarity graph represent the network services, and an undirected edge exists between two nodes if and only if they are similar, as defined above. Figure 5 shows an example of a similarity graph.

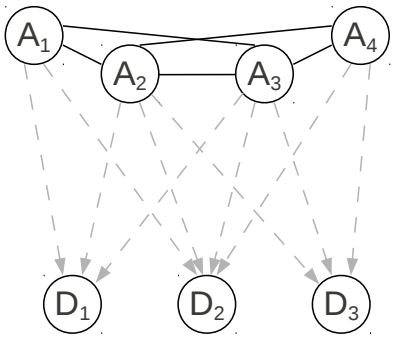


Figure 5: An example similarity graph (solid lines) overlaying on its communication graph, where the similarity and agreement thresholds equal 50%. A dashed line $A_i \rightarrow D_j$ represents that A_i potentially depends on D_j, and a solid line $A_i \rightarrow A_j$ represents that A_i and A_j are similar. The dependency candidates $A_1 \rightarrow D_3$ and $A_4 \rightarrow D_1$ will be inferred because D_1, D_2, D_3 are all agreed on by the group of services in A.

A group of similar network services is represented by a connected subgraph in the similarity graph. After identifying the services that are members of this group, we identify the common depended services that the members of the group agree on. The *agreement* on a depended service D is defined as the ratio between the number of services in the group that depend on D and the size of the group. If the agreement on D is greater than a threshold, we can infer that D is a common depended service of the group, and any members that do not depend on D will have such a dependency candidate inferred.

After identifying the common depended services of a group, we need to estimate the corresponding confidence values. To do so, we modify the original communication graph by adding edges to represent the newly inferred dependencies, and set the weights of the newly inferred dependency candidates to a value such that the ratio between the weight of the candidate and its depending service is proportional to the greatest ratio in the group.

Specifically, let $\mathscr{A}$ be a set of similar services $\{A_1,A_2,...,A_n\}$, and $\mathscr{D}$ be the set of $\mathscr{A}$'s common depended services $\{D_1,D_2,...,D_m\}$. For each pair of services A_i, D_j, if the dependency $A_i \rightarrow D_j$ does not exist, we add an edge from A_i to D_j. As a result, there is an edge from each member in $\mathscr{A}$ to each member in $\mathscr{D}$.

For each depended service D_j in $\mathscr{D}$, let A_{max} be the service in $\mathscr{A}$ such that the ratio $\frac{weight(A_{max}\rightarrow D_j)}{weight(A_{max})}$ is the greatest. For each member service A_i in $\mathscr{A}$, set the $weight(A_i \rightarrow D_j)$ as $\frac{weight(A_i)weight(A_{max}\rightarrow D_j)}{weight(A_{max})}$. The new communication graph can then be used with the logarithm-based ranking to calculate the final confidence value of each candidate.

When the weight of an edge in the extended communication graph is set to this proportional weight, the final confidence value of an inferred dependency will be proportional to how often the depending service is accessed. As a result, even when coincidental traffic causes false dependencies to be inferred, the confidence values of these inferred candidates are low enough to be pruned before it is reported as a dependency.

3.3 Discovery of Service Clusters

Finally we introduce an approach for identifying service clusters in a network. A network service cluster is a set of services $\{B_1,B_2,...,B_n\}$ such that if a dependency $A \rightarrow B_1$ exists, B_1 can be substituted by any other service in the cluster without affecting the completion of A's task. These clusters typically provide load-balancing or backup for core services to improve performance and fault tolerance.

Both Orion [6] and NSDMiner [10] proposed to use the knowledge of service clusters to improve the detection of dependencies that involve service clusters. However, both approaches rely on human users to provide such information [6, 10]. Such manual interventions re-

quire additional human efforts, and are time-consuming and error-prone. In the following, we develop an automated approach to discovering service clusters.

Let us start with the observation behind the proposed approach. When a network service A_1 depends on services in a cluster $\{B_1, B_2, B_3\}$, it is highly likely that $A_1 \rightarrow B_1$, $A_1 \rightarrow B_2$, and $A_1 \rightarrow B_3$ will be observed as dependency candidates. In a load-balancing configuration, the services that depend on the cluster will access the members of the cluster with roughly equal probability. This means that over time, so long as the load-balancing assignments are not "sticky," all services that depend on the cluster will feature all of the members in their lists of depended services. In a backup configuration, the services that depend on the primary server $B_{primary}$ will feature it in their list of depended services in normal situations. When $B_{primary}$ fails, the backup service B_{backup} will take over the service by $B_{primary}$, and these depending services will now have B_{backup} as their depended services (along with $B_{primary}$ before the failure).

In both cases, when services consistently appear together in the lists of depended services of the same depending service, it is likely that they are part of service clusters. By counting the number of cases for which depended services appear together for the same depending service, we can estimate the likelihood that a cluster exists.

Now let us present the approach for discovering service clusters. Our approach takes a communication graph produced by NSDMiner as its input, disregarding the weights of the nodes and edges. The algorithm produces a *clustering graph*, where an edge exists between two nodes B_1 and B_2 if and only if (1) B_1 and B_2 use the same protocol with the same port and (2) there exists a service A such that $A \rightarrow B_1$ and $A \rightarrow B_2$ are dependency candidates. The weight of each edge, called the *support* for the pair of services, is the number of services for which B_1 and B_2 are both depended services. Figure 6 shows an example of a clustering graph generated from a communication graph.

This approach assumes that network services that are parts of clusters will occur together frequently in the lists of depended services, and that pairs of coincidental services will occur in the same list of depended services infrequently. After generating the clustering graph, all edges with weights less than a support threshold will be dropped, and the remaining connected subgraphs will be considered the service clusters of the network. Because clusters are expected to have high supports while the support of coincidental pairs of services will be low, only a moderate absolute support threshold should be necessary for identifying the clusters, such as a value between 5 and 10, depending on the size of the network.

Rather than using clusters to infer new candidates

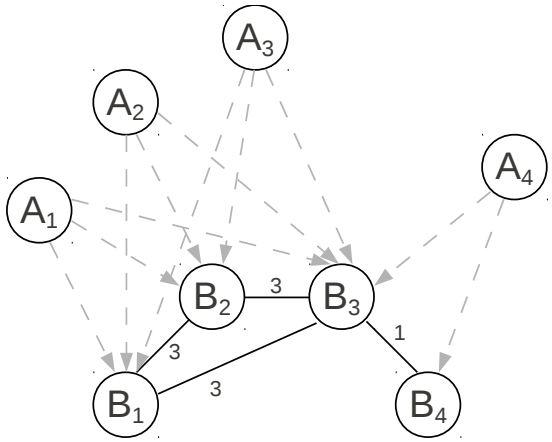


Figure 6: An example of a cluster graph (solid lines) overlaid with its communication graph (dashed lines). Assume the support threshold is 2. $\{B_1, B_2, B_3\}$ is a cluster, while B_4 is not considered a part of this cluster.

or increase the weights of edges, we instead aggregate the results of NSDMiner so that instead of reporting a separate dependency for each member of a cluster ($A_1 \rightarrow B_1, A_1 \rightarrow B_2, A_1 \rightarrow B_3$), we treat all of the members of the cluster as a single group of depended services ($A_1 \rightarrow B_{cluster(1,2,3)}$). In the example in Figure 6, the total number of dependencies after this aggregation is reduced from 12 to 5, making the output smaller and more manageable, reducing the time needed to validate the results. In our evaluation, we show that even when the thresholds for clusters are very strict, the number of dependencies is still greatly reduced.

4 Evaluation

We have implemented the proposed techniques as extensions to NSDMiner. To have a realistic evaluation of these techniques, we use real-world network traffic collected from the production network of Department of Computer Science at North Carolina State University. In the following, we first give a brief description of the evaluation data set, and then present the experimental results.

4.1 Evaluation Data Set

Our experimental evaluation uses the same data set that were used to evaluate the original NSDMiner [10]. In the following, to make this paper self-contained, we repeat the description of this data set.

We evaluate our proposed techniques using the network traffic collected from the production network on the second floor of the building where our department is located, including all the internal traffic across subnets in the building. This network consists of 14 switches

hierarchically organized in two levels. Each computer in this network is connected to a switch via a wall port or a port on the switch directly. The top-level switches are then connected to a master building switch through a 10G multimode fiber, which facilitates communication with the campus network outside of the building.

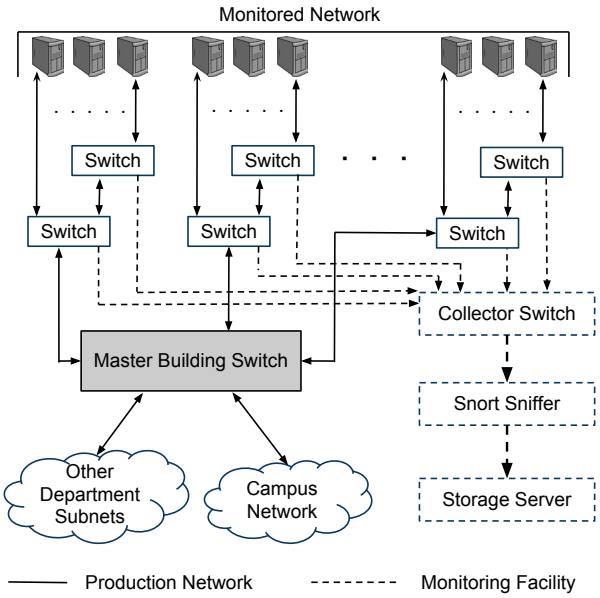


Figure 7: Diagram of the traffic monitoring facility [10].

Figure 7 shows a diagram of our traffic monitoring facility. In order to collect the traffic from our network, each of the switches is configured with a SPAN session so that packets that are routed throughout the building are also forwarded to a machine running a packet-sniffer. The packet-sniffer is Linux machine running *snort* [13] in packet-logging mode, where each packet is stripped down to its IP and TCP/UDP header information and forwarded to a secure data storage machine that is disconnected from the public Internet. We then convert these packet headers into network flow records using *softflowd* [14]. The final evaluation data set includes 46 days of network flows collected between 05/10/2011 and 06/25/2011, which consists of 378 million flow records.

The ground truth for this data set was generated with the assistance of the departmental IT staff. Our traffic covers a total of 26 servers, 3 of which are Windows machines, and 23 are Linux machines. These machines offer a total of 43 services possessing 108 dependencies. The majority of the Linux machines belong to individual faculty, and are used to provide shell and web services for their graduate students, while the rest provide infrastructure such as Kerberos, databases, and DFS replication.

Table 1 summarizes the services and their dependencies. Most of the services offered were dependent on DNS (53) for name resolution. Windows-based services were dependent on Active Directory (AD) services for domain queries (389) and authentication (88). The services that were dependent on those offered on dynamic ports (RPC) were also dependent on endpoint mapper to resolve the port numbers. Most of the Linux-based services were dependent on LDAP (389) for directory access. Two of the interesting services hosted were TFTP (69) and database (3306); they were running stand-alone and was not dependent on any other network service. Windows deployment service (WDS) and DFS replication service were offered on dynamically allocated ports and others were offered on standard well-known ports.

Table 1: Ground truth of service & dependencies

Service	Instances	Dependencies	
webservice (80, 443)	4	2	DNS, DBMS
webservice (80)	1	1	DNS
ssh (realm-4) (22)	5	2	Kerberos, DNS
ssh (realm-5) (22)	17	3	Kerberos, DNS, LDAP
svn (8443)	1	4	DNS, LDAP, port mapper, RPC
proxy DHCP (4011)	1	2	DNS, LDAP
DHCP (68)	1	1	DNS
email (25)	1	2	mail exchange server, DNS
endpoint mapper (135)	2	3	DNS, AD, Kerberos
WDS (RPC)	1	5	DNS, AD (LDAP, port mapper, RPC, Kerberos)
DFS replication (RPC)	2	5	DNS, AD (LDAP, port mapper, RPC, Kerberos)
SMB (445)	2	5	DNS, AD (LDAP, port mapper, RPC, Kerberos)
TFTP (69)	1	0	
database (3306)	2	0	

Note that even though we worked hard to identify all dependencies, there is a possibility that we might have missed some rare non-obvious ones. The reader is advised to keep this in mind while interpreting the experimental results.

4.2 Logarithm-Based Ranking

In order to evaluate the effectiveness of the logarithm-based ranking scheme, we run NSDMiner with our logarithm-based ranking scheme and compare the results to the original ratio-based scheme [10] for both shared and exclusive mode. We compare our results to the previous version of NSDMiner as it has already been demonstrated as the most effective network-based solution [10].

Table 2 shows an initial comparison of the true positive (TP), false positive (FP), and false negative (FN) for both ranking schemes. We choose a filtering threshold for the logarithm-based ranking scheme that set the number of true positives as close to those obtained by the ratio-based ranking at confidence value 0.5% as possible.

The dependency candidates with the highest false positive rates under the ratio-based ranking were those that were accessed the least. For example, in the ratio-based ranking scheme in the shared mode, Microsoft endpoint mapper (of which there are two instances, with three

Table 2: Comparing the false positive rates between the Ratio-based [10] and Log-based ranking schemes by setting the threshold to a value that keeps the number of true positives close to their value under the ratio approach.

	Shared Mode						Exclusive Mode					
	Ratio (0.5%)			Log (45%)			Ratio (0.5%)			Log (44%)		
Service	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
webservice	8	9	1	9	12	0	8	4	1	8	7	0
email	3	1	0	3	3	0	3	0	0	3	2	0
ssh(realm-4)	10	1	0	10	1	0	10	1	0	10	1	0
ssh(realm-5)	41	14	10	43	12	8	40	10	11	43	9	8
svn	4	1	0	4	0	0	4	1	0	4	0	0
proxy DHCP	2	1	0	2	0	0	2	1	0	2	0	0
DHCP	1	6	0	1	1	0	1	6	0	1	1	0
endpoint mapper	6	43	0	6	4	0	3	6	3	3	1	3
WDS (RPC)	4	10	0	3	1	1	4	1	0	2	0	2
DFS replication (RPC)	8	4	0	6	3	2	7	0	1	5	1	3
SMB	9	7	1	9	9	1	9	3	1	9	5	1
TFTP	-	1	-	-	1	-	-	0	-	-	0	-
database	-	1	-	-	2	-	-	1	-	-	2	-
Invalid Services	-	34	-	-	29	-	-	30	-	-	17	-
Total	96	133	12	96	78	12	91	64	17	90	46	18

dependencies each) reported a total of 43 false dependencies using 0.5% confidence [10]. One instance was accessed 465 times, while the other was accessed 133 times. In order for a candidate to be pruned at 0.5% confidence, it would need to have a weight of less than 2.3 and 0.6, respectively. In the latter case, a single nested flow was enough to consider the candidate as a true dependency. Most of the true positives (LDAP, Kerberos, and DNS) had weights above 10.

With the logarithm-based ranking scheme, all dependency candidates with weights less than or equal to 1 are immediately pruned with a confidence value of 0%, as a single observation is never convincing enough to consider it a true dependency. Because $log_{133}10$ is .47, the true positives in this example were given confidence values above 47%. Thus, in Table 2, the true positives are preserved and many false positives are properly pruned.

To see the overall performance of both the ratio-based and the logarithm-based ranking schemes, we plot the Receiver Operating Characteristic (ROC) curve in Figure 8. This figure shows the true positive rate as a function of the false positive rate for both ranking scheme in both the shared and the exclusive mode. We can see a substantial improvement in the detection rate for the same false positive rate when the logarithm-based ranking scheme is used. This is because we can filter more false positives at higher confidence thresholds without incorrectly pruning true dependencies. It is worth noting, however, that even with these improvements, the true positive rate still reaches its maximum value at the same point in both ratio-based and logarithm-based ranking schemes.

Figure 9 illustrates how the behavior of the confidence threshold differs between the ratio-based and the logarithm-based ranking schemes. In both shared and

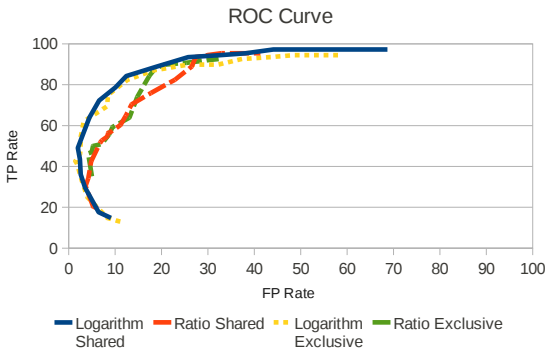


Figure 8: ROC curve for the logarithm-based and the ratio-based ranking schemes in both shared and exclusive modes.

exclusive modes with the ratio-based ranking, the true and false positive rates drop drastically as the confidence threshold increases. The false positive rate approaches its minimum value at a threshold of 10%, but at the cost of missing half of the true dependencies. In the logarithm-based ranking, both true and false positives are pruned at a slower rate. Due to the greater distance between their confidence values, true positives are pruned more slowly than false positives, so at the point when the false positive rate approaches its minimum value, the true positive rate is much higher than that under the ratio-based ranking.

In addition to improving the discovery of true dependencies with lower false positive rate, the logarithm-based ranking scheme more accurately predicts appropriate confidence values for dependency candidates. The confidence value given to a dependency candidate should roughly correspond to the probability that the candidate is true. For example, out of a set of candidates with

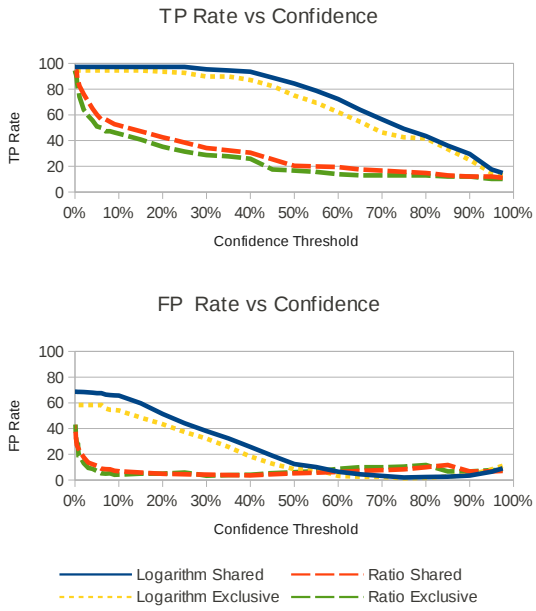


Figure 9: True/False positive rate v.s. confidence threshold.

confidence values of 50%, half of them should be expected to be true positives. Figure 10 shows the distributions of true positives at 5% confidence intervals for both the logarithm-based and ratio-based ranking schemes in both shared and exclusive modes. Note that with the ratio-based ranking scheme, nearly all of the candidates ranked above 10% are true positives. As shown in Figure 9, this is because there are very few candidates ranked above 10% at all. As most of the results are clustered at low confidence values, it only takes a modest threshold to remove them from the results.

In contrast, and especially for the shared mode, the curve for the logarithm-based ranking scheme is much smoother and better illustrates how one would expect the true positive rates to map to the confidence rankings. When the confidence values are more meaningful, it is possible to make an informed decision about how many true and false positives should be expected when a certain cutoff value is chosen. While the curve for the exclusive mode is a bit choppier due to having a few high-confidence false positives, they still exhibit the same general patterns. Many false candidates are ruled out before reaching the pruning stage, and as a result, the true positive concentrations in exclusive mode are higher overall.

4.3 Inference of Service Dependencies

To test the effectiveness of the inference algorithm, we randomly choose from between 1 and 150 dependencies from the ground truth and remove all of the flows associated with those dependencies from the data set. We then

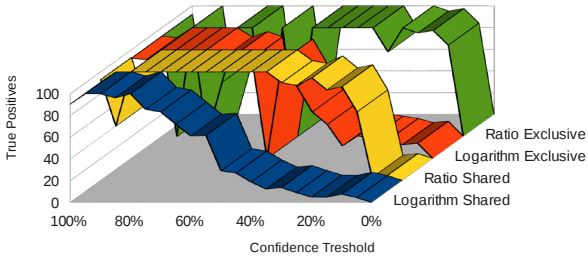


Figure 10: The percentage of true positives that are found at each particular ranking threshold. 95% in the 100 bin means that 95% of candidates reported with scores between 100 and 96 were true, while the other 5% were false. The ideal case is a straight line where $x = y$.

run the inference algorithm to see how many of these dependencies could be recovered from the remaining data. We reason that if the inference algorithm can re-infer dependencies known to be true, then it should successfully be able to infer unknown dependencies in practice.

When running the inference algorithm, we use four combinations of high (75%) and low (25%) thresholds for similarity and agreement to determine the effect that each threshold has on the ability of the approach to recover missing dependencies. Only dependencies with the (logarithm-based) confidence values greater than 30% are used for these experiments. In the following, we only discuss the results of the shared mode in detail; the results of the exclusive mode are similar. For each pair of similarity and agreement thresholds, we ran 10,000 trials of removal and recovery.

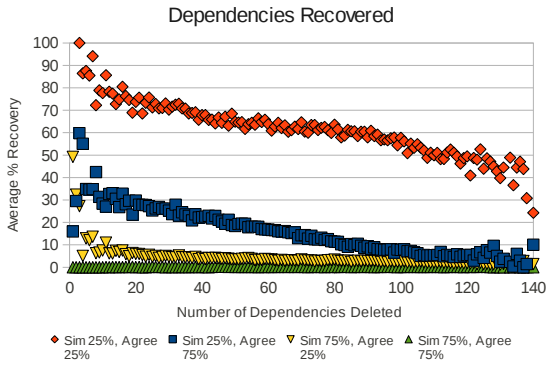


Figure 11: Dependencies recovered where all candidates with the logarithm-based confidence values less than 30% are removed. We show the average percentage of the recovered dependencies as a function of the number of removed dependencies. Each color/shape represents a certain similarity-agreement pair of thresholds.

Figure 11 shows the average percentage of recovered dependencies as a function of the number of dependen-

cies that were removed. As one would expect, as more traffic is removed, it becomes harder to infer dependencies from what is left.

Increasing the agreement threshold has a much more dramatic effect on the recovery rate than the similarity threshold. When both the similarity and agreement thresholds are set to low values such as 25%, it is possible to recover many dependencies, and when both thresholds set to high values such as 75%, no dependencies are recovered at all.

In addition to recovering missing dependencies effectively, the inference approach is able to do so without inferring false positives. Figure 12 shows the average number of false positives that were inferred when running the previous experiments, while Figure 13 demonstrates the overall improvement to the ROC curve. Not only do we reach 100% coverage, we do so with a relatively low false positive rate.

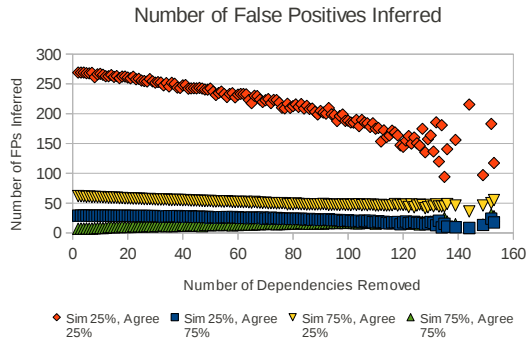


Figure 12: This shows the number of false positives inferred during the same experiments described earlier. Note that while there are often several false positives inferred, many of them are can be pruned in later steps of NSDMiner as they often have low confidence values.

In order to understand the effects of these thresholds, we discuss the specific behavior for our largest group of similar machines, our SSH services. These services mostly correspond to individual faculty machines that give shell and website services to their graduate students. By their nature, some of these are less frequently used than others.

With similarity thresholds of 25%, 27 such services, covering all true cases and 5 false cases, were considered similar. When the agreement threshold was low at 25%, all three true dependencies (and two false positives) were found to be in common and inferred as dependencies to the rest of the group, contributing to the full coverage of the ground truth. At 75% agreement, the only service that could be inferred was DNS.

When the similarity threshold is set to 75%, this group is reduced to seven members. When there are less members, it is easier to reach 25% agreement on depended services, meaning that four additional false positives, were inferred as a result. At 75% agreement, the only common depended services were Kerberos and DNS.

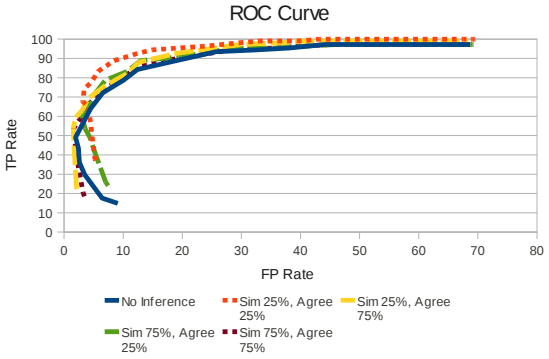


Figure 13: ROC curve for various similarity-agreement thresholds. A higher agreement threshold prevents the eager acceptance of false positives, however it still raises the confidence of true positives, allowing a higher confidence threshold to be set.

While our experiments are performed on a real-world network, it is not necessarily representative of networks in general. Additional work on more diverse configurations will be needed to identify the most appropriate choices for similarity and agreement thresholds. One important factor in this choice is the number of service clusters that are used in the network. The SSH services depended on three of the same clusters, giving them up to 13 true depended services in common, making it possible for them to have dependencies inferred with thresholds up to 50%. If there are fewer clusters in the network, then lower thresholds will be needed to find similar services and common dependencies.

4.4 Discovery of Service Clusters

In order to evaluate the effectiveness of our clustering approach, we run our tool on the output of NSDMiner and compare the clusters that it reports to the clusters in our ground truth. There are two variables that affect the clusters that are reported: the support threshold, which specifies the number of depending services for which members of a cluster must appear together as depended services, and the confidence threshold, which sets the minimum confidence for a depended service for it to be counted as part of a cluster.

Our network contains a total of 10 service clusters, most of which contain 4–6 services. We consider a cluster reported by our clustering tool to be a true positive cluster so long as it contains at least two services where none of the reported services are put into the wrong cluster. Figure 14 shows the number of true clusters as a

function of the number of clusters that are reported in total.

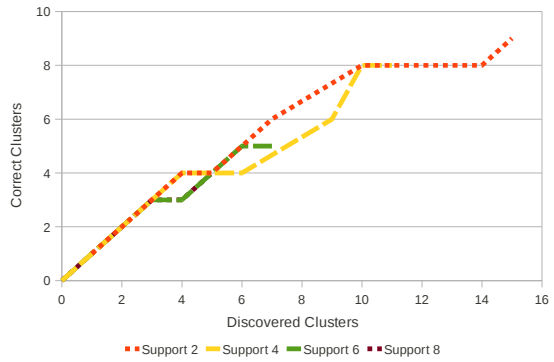


Figure 14: Quality of the clusters as a function of the number discovered. We consider a true cluster to be any size cluster that does not contain false positives. Therefore, a true cluster may be missing some of its members. All members are usually recovered when the confidence threshold is set below 60%.

We can see here that the clusters reported are fairly accurate. When 10 clusters are reported, clustering with low support values of 2 and 4 both report 8 true clusters. When looking carefully at the clusters reported, there was only one cluster that actually accepted false positives into its membership was Microsoft endpoint mapper. Services in the two distinct kerberos clusters were never mixed together.

The most difficult cluster to detect was our University's NTP time servers. No network services in our network explicitly depend on NTP, meaning that traffic to NTP was coincidental and confidence values associated with NTP dependencies were rightfully low. Due to these low thresholds, the NTP cluster could only be discovered at a support threshold of 2 and a confidence value below 30%.

An interesting set of false positives that survived through very high support thresholds were a number of web servers accessed over HTTP and HTTPS. Even at a threshold of 8, the HTTPS servers were still discovered with higher confidence than many of the true positives. SSH is interesting in that if a user uses their SSH account as a proxy for their web browser, then any web servers that are accessed will be considered dependency candidates of that SSH session. If popular websites are used frequently, such as Google (which appeared in a cluster of services running on port 80), they will considered clusters by this algorithm.

Overall, the quality of clusters reported was good. Even at low support thresholds, most of the clusters reported were correct, though this may be a property of our small network. The support threshold needs to be scaled to the size of the network, and the effect on larger networks still needs to be explored.

The value of clustering with respect to identifying dependencies comes from aggregating the services that make up a service cluster into a single depended service. Any degree of clustering, whether it is strict or lenient, will remove a large number of candidates from the output of NSDMiner. In most cases, this is because the true positives have the greatest tendency to influence the clusters that are formed. True positive clusters, by definition, are those with the greatest support values, and the support value of a cluster relates to how many depending services will benefit from the aggregation. So long as the clusters identified are accurate, as we have shown, then reducing the size of the output will be beneficial with no loss of information.

It is difficult to represent this benefit in the form of an ROC curve because aggregation actually increases the false positive rate by removing true positives from the output. Instead, we represent the benefit by showing the size of the output of NSDMiner as a function of the confidence thresholds and support thresholds used in Figure 15. In most cases, we can see an reduction in output size from anywhere between 25% to 50%.

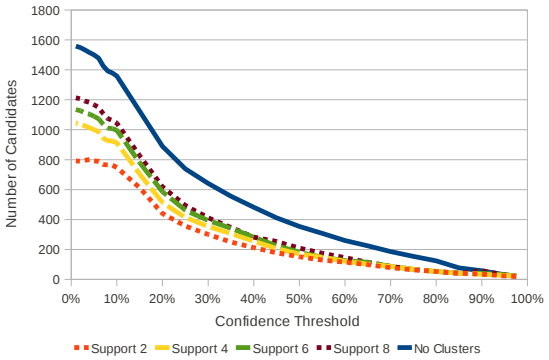


Figure 15: Reduction in the number of candidates for various support thresholds. The confidence threshold corresponds to the logarithm-based ranking introduced in this paper.

5 Conclusions and Future Work

In this paper, we introduced three techniques for improving the discovery of network service dependencies, and implemented them as modifications to the tool NSDMiner [10]. We developed a new ranking formula that makes it easier to judge whether or not a dependency candidate is true or false, an approach for inferring the dependencies infrequently-used services that are similar to others, and an approach for finding service clusters such as backup or load-balancing nodes in a network. Furthermore, we evaluated our approaches on production

traffic in a university network, showing a substantial improvement in both true and false positive rates.

A number of issues are still left unexplored. For example, remote-remote dependencies are still not considered by NSDMiner at all. NSDMiner also relies on the assumption that network sockets are opened and closed on demand, meaning that it will miss dependencies involving middleware applications that keep connections open for extended durations. Finally, NSDMiner assumes that during the time that flows are collected, no changes occur in the network configuration, meaning that it is not able to keep up with a dynamically evolving network. Future work will address these issues and involve the collection additional data from a larger network to further test the effectiveness of these approaches.

Acknowledgements

We would like to offer our thanks to the continued support of our departmental IT staff, Carlos Benavente, Vadim Kuznetsov, Trey Murdoch, and Jeremy Meeler for their assistance in keeping our ground truth up-to-date, as well as Neal McCorkle and William Brockelsby for their help in managing our traffic monitoring infrastructure throughout our experiments. We would like to thank Arun Natarajan, who developed the first version of NSDMiner [10], for all the help and support with the original NSDMiner prototype and validation scripts. Finally, we would like to thank the conference reviewers and shepherds for their feedback in finalizing this paper for publication in LISA '12.

This work is supported by the U.S. Army Research Office (ARO) under a MURI grant W911NF-09-1-0525.

References

[1] BAHL, P., BARHAM, P., BLACK, R., CH, R., GOLDSZMIDT, M., ISAACS, R., K, S., LI, L., MACCORMICK, J., MALTZ, D. A., MORTIER, R., WAWRZONIAK, M., AND ZHANG, M. Discovering dependencies for network management. In *In Proc. V HotNets Workshop* (2006).

[2] BAHL, P., CHANDRA, R., GREENBERG, A., KANDULA, S., MALTZ, D. A., AND ZHANG, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2007), SIGCOMM '07, ACM, pp. 13–24.

[3] BARHAM, P., BLACK, R., GOLDSZMIDT, M., ISAACS, R., MACCORMICK, J., MORTIER, R., AND SIMMA, A. Constellation: automated discovery of service and host dependencies in networked systems. *MSR-TR-2008-67* (2008).

[4] BARHAM, P., DONNELLY, A., ISAACS, R., AND MORTIER, R. Using magpie for request extraction and workload modelling. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6* (Berkeley, CA, USA, 2004), USENIX Association, pp. 18–18.

[5] CHEN, M. Y., ACCARDI, A., KICIMAN, E., LLOYD, J., PATTERSON, D., FOX, A., AND BREWER, E. Path-Based Failure and Evolution Management. In *IN PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI'04* (2004), pp. 309–322.

[6] CHEN, X., ZHANG, M., MAO, Z. M., AND BAHL, P. Automating network application dependency discovery: experiences, limitations, and new solutions. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation* (Berkeley, CA, USA, 2008), OSDI'08, USENIX Association, pp. 117–130.

[7] DECHOUNIOTIS, D., DIMITROPOULOS, X., KIND, A., AND DENAZIS, S. Dependency detection using a fuzzy engine. In *Managing Virtualization of Networks and Services*, A. Clemm, L. Granville, and R. Stadler, Eds., vol. 4785 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007, pp. 110–121. 10.1007/978-3-540-75694-1_10.

[8] KANDULA, S., CHANDRA, R., AND KATABI, D. What's going on?: learning communication rules in edge networks. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication* (New York, NY, USA, 2008), SIGCOMM '08, ACM, pp. 87–98.

[9] KELLER, A., KELLER, E., BLUMENTHAL, U., AND KAR, G. Classification and computation of dependencies for distributed management. In *Proceedings of the Fifth International Conference on Computers and Communications (ISCC* (2000).

[10] NATARAJAN, A., NING, P., LIU, Y., JAJODIA, S., AND HUTCHINSON, S. E. NSDMine: Automated Discovery of Network Service Dependencies. *Proc. IEEE INFOCOM'11 In Submission* (2011).

[11] NSDMINER. http://sourceforge.net/projects/nsdminer.

[12] POPA, L., CHUN, B.-G., STOICA, I., CHANDRASHEKAR, J., AND TAFT, N. Macroscope: end-point approach to networked application dependency discovery. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies* (New York, NY, USA, 2009), CoNEXT '09, ACM, pp. 229–240.

[13] SNORT. http://snort.org.

[14] SOFTFLOWD. http://code.google.com/p/softflowd.

A Appendix

This appendix highlights implementation details that may be of use to network administrators interested in deploying NSDMiner on their own networks. While this appendix is written assuming that the reader will be using our open source implementation of NSDMiner, `nsdmine` [11], the higher-level concepts are applicable to any implementation.

This section is written as follows: we begin with a discussion of NSDMiner's expected input and how to configure a network to collect this input. We then discuss the parameters for NSDMiner's various algorithms with backreferences to the sections in the paper where they were first introduced. Finally, we conclude with the expected output and how to interpret this output. For any details omitted from this section, we refer the reader to NSDMiner's README file.

A.1 Preparation

In order to run NSDMiner, network traffic flows must be collected and stored. This can be accomplished in several ways. Some routers have the ability to save and export Cisco Netflows (using the switch `--cisco`), which are a standard format for representing network traffic flows.

For routers that do not export Cisco Netflows, it is also possible to use port mirroring on routers to copy and forward all traffic to a central location, and use a tool such as *softflowd* [14] to reconstruct flows from the individual packets. This may already be set up on networks running an intrusion detection system such as *snort* [13], in which case it should be possible to configure the IDS to save packets locally to be later processed by NSDMiner.

Our implementation of NSDMiner supports loading pcap (tcpdump) packets by using the `--pcap` switch. We implement NSDMiner as a Python package and represent flows as an object, meaning it is possible to create a loader that can convert network traffic into the format needed by our implementation without having to change the code for the NSDMiner algorithm itself.

The amount of data is needed to use NSDMiner usefully is a function of the size and the activity levels of the network. On our single-building network, it took 2-3 weeks of data to reach the point of diminishing returns [10].

A.2 Choice of Parameters

NSDMiner accepts the following parameters, presented as switches to the `nsdmine` invocation.

- A set of IP addresses I referring to all of the servers for which dependencies should be collected, provided as a comma-seperated filter of IP addresses.
- The confidence threshold α, between 0 and 100. Dependency candidates with a confidence value (as calculated by the algorithm in Section 3.1) less than α will be pruned from the output.
- The similarity threshold δ, between 0 and 100, used for the Inference algorithm (Section 3.2).
- The agreement threshold Δ, between 0 and 100, used for the Inference algorithm (Section 3.2).
- The support threshold σ, an integer greater than 0, used for the Clustering algorithm (Section 3.3).
- A confidence threshold α', an integer greater than 0, and less than α.
- An example of an invocation to NSDMiner setting all of these options would be in the form `nsdmine --filter=`I `--alpha=`α `--infer=`δ,Δ `--clusters=`σ,α'.

NSDMiner works in two phases, the first being dependency graph generation, followed by post-processing. Dependency graph generation analyzes the network flows as explained in Section 2.2. Generation of the dependency graph runs in $O(n^2)$ time, and because the input is generally in hundreds of millions of flows, this can take a few days to generate the entire dependency graph. Specifying exactly which servers should be monitored for dependencies substantially reduces this time.

The second phase is post-processing, where inference (`--infer`), clustering (`--clusters`), and ranking are performed on the dependency graph. Each post-processing component accepts configurable parameters that will vary in effectiveness from network to network. Inference attempts to identify the dependencies of services that are infrequently used by comparing them to similar services in the network. Inference is only useful when there are several services that are configured in the same way, such as in a data center or web-hosting provider's server farm. δ should be set high if NSDMiner is being used on a noisy network. However, if one is analyzing a private LAN, δ can be set lower. Δ should be set high if many of the services in the network are part of clusters, and low if most of the services are only hosted on single machines.

The clustering algorithm attempts to locate redundant service clusters, such as those used for back-up or load-balancing purposes. Clustering can only find service clusters that are used by more than σ services. The setting of σ is much like a peak-detection problem – it should be high enough to ensure that all coincidental traffic is ignored, but low enough to catch all of the actual clusters. α' should be set to about half of a α.

Finally, the dependencies are ranked and given confidence scores normalized from 0 to 100%. Any dependencies with a score less than α will not be present in the final output. Each of the post-processing approaches work in parallel on the original dependency graph generated in phase one, so their ordering does not matter.

A.3 Output

The final output is the list of services running on the machines in I and each of their dependencies, each marked with a confidence ranking from α to 100%. The ranking roughly estimates the probability that the specified dependency is a true positive. If clustering was used, then the clusters that were found will also be reported.

After collecting this input, the operator should then verify each of the reported dependencies manually to determine how accurately NSDMiner runs on his or her network. By verifying the output, it is possible to fine-tune the parameters to make using NSDMiner more reliable in future runs.

What Your CDN Won't Tell You: Optimizing a News Website for Speed and Stability

Julian Dunn
SecondMarket Holdings, Inc.
jdunn@aquezada.com

Blake Crosby
Canadian Broadcasting Corporation
me@blakecrosby.com

Abstract

In this paper, the authors discuss their experiences implementing, operating, and optimizing a content delivery network for Canada's largest news website, that of the Canadian Broadcasting Corporation (CBC). The site receives over one million unique visitors per day. Although CBC uses the Akamai Aqua Platform (formerly EdgeSuite) as its content delivery network of choice, the lessons described here are generally applicable to any infrastructure fronted by a CDN.

1. Introduction

Content Delivery Networks[1] (CDNs) have become increasingly popular over the last decade. With the explosion in users on the Internet, it has become impractical for companies with any significant amount of traffic to invest solely in their own infrastructure to deliver information to their end users.

Nowhere is this truer than in the online media business, where day-to-day traffic volumes can deviate wildly based on the news of the day. For example, the 9/11 attacks on the World Trade Center towers caused site outages for many major media organizations when servers failed under the extreme traffic load[3]. This event — and many others since then, like earthquakes, tsunamis, and the 2008 United States presidential election — repeatedly taught media companies a lesson: they could no longer afford to only invest in capital equipment like hundreds of racks of servers, simply to handle the one or two major news events that might cause traffic spikes of ten or one hundred fold. Some external buffer, like a CDN, is necessary.

Many Internet system administrators are already familiar with the technical design of a CDN. Very briefly, a CDN operates on the premise that long-haul Internet backbones are slow or unreliable and have a long propagation time, whereas access to one's Internet Service Provider (ISP)'s data center is fast, since it is often only one or two hops away. By building a private WAN with proprietary, optimized routing algorithms over the public Internet between ISPs and placing acceleration/caching services in ISP data centers, CDNs can accelerate the speed of content delivery to the end-user. Also, by operating on the assumption that many users will access the same piece of content, CDNs can serve as a distributed cache to further increase the speed of content delivery.

In practice, CDNs are most useful for mostly static content; dynamic, real-time applications, such as Facebook or Twitter, do not benefit as much from CDNs, although those organizations still use CDNs for delivery of static assets such as graphics. Furthermore, increases in speed and response times of even the public Internet backbones over the last decade have diminished the speed advantages of CDNs. They are however, an excellent fit for media organizations like CBC, where relatively-infrequently changing content is coupled with enormous variability in traffic volumes.

2. What does "Tuning a CDN" mean?

Tuning a CDN is a sophisticated process requiring an in-depth understanding of one's overall business and technical requirements. It is more than just establishing and configuring technical parameters, such as Time-To-Live (TTL) policies on content. Although TTLs are an important component of effective CDN operation, they are not the only parameter worth examining — a mistake that CBC initially made.

It's important to develop a detailed understanding of one's content delivery infrastructure prior to embarking on a tuning exercise. Some of the elements worth examining are:

- The architecture of the web-serving infrastructure from which the CDN retrieves content, also known as the origin
- Amount of personalization and content targeting on the site
- Frequency of content change
- User access patterns
- Business requirements governing what metrics should be optimized (cost, content freshness, reliability, complexity)

These elements all have a material impact on how well one's site integrates with a CDN, and ultimately what rules should be employed when.

CDNs vary wildly in the ability for system administrators to tune the operation of their algorithms. In general, performance tuning a CDN comes down to several key parameters:

- The frequency at which the CDN should retrieve content from your origin, and on what basis (Time-To-Live, freshness checks);
- The tradeoff between maximizing the proportion of objects served from the CDN rather than the origin (known as the origin offload percentage) versus content freshness;
- The relative cost of bandwidth at the edge (CDN to end-user) versus the origin. Also important to consider: the so-called midgress bandwidth costs, which is the price that the CDN charges for internal CDN communication between cache hierarchies;
- Whether it is acceptable to serve expired (dirty) objects from the CDN;
- What HTTP optimizations should be implemented between the origin and the CDN to maximize performance.

Akamai's Aqua Dynamic Site Accelerator[4] product is aimed at generic website workloads, so many parameters are tunable via basic configuration. For experienced system administrators, "extended metadata" can be added to a basic configuration, to be able to tune parameters such as content time-to-live (TTL) values, logging and reporting, and content compression. Extended metadata also allows the administrator to implement extra features such as HTTP pipelining, origin forwarding, large object optimization, failover rules, and Edge-Side-Include[5] processing. There are also additional features unavailable to administrators through the control panel, but which can be implemented by Akamai solutions engineers, such as complex redirects using regular expressions, or adjusting the cache key for an HTTP object based on a cookie or User-Agent.

With all these tuning parameters available to the average system administrator, it can be difficult to know what to alter. In the authors' experience, CDN vendors do not provide a lot of guidance, primarily because the answers to which parameters to modify and what values to set vary greatly based on the customer's workload, requirements, and business context. Design and optimization of the entire system, including the architecture of the origin infrastructure, is generally beyond the scope of even a professional services engagement. The most important takeaway before beginning an optimization activity is to understand the existing content access patterns and delivery requirements.

3. Content Access Patterns in Media

Online news sites like CBC's have several characteristics distinct from many other websites.

3.1. Unpredictable Traffic Patterns

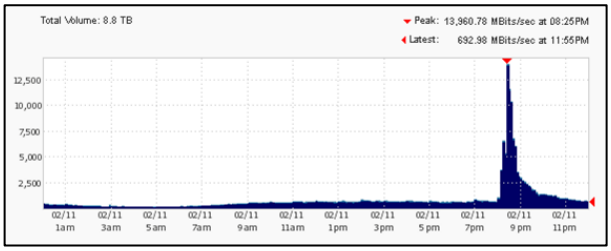


Fig. 1: Example of a traffic patterns seen during a breaking news event, where site traffic (seen here in MBits/sec) increases an order of magnitude in a matter of minutes.

Flash news events, such as natural disasters or political uprisings, can change daily traffic patterns by several orders of magnitude in a matter of minutes, as seen in the above figure, which illustrates traffic seen during the evening of February 11, 2012, when pop singer Whitney Houston died.

3.2. Content Frequently Updated Within First 24 Hours, Then Almost Not At All

Breaking news stories are revised frequently within the first 24 hours, sometimes dozens of times within the first few hours, and then very infrequently after 48 hours. If a story has new developments, reporters will often write a new story linking to the original story, rather than updating the old one. Traffic to the story has a corresponding exponential fall-off, but the long tail is infinite and may peak if there are new developments, even years later, in a story. For example, if a crime is committed on Monday and someone is arrested on Friday, the original crime story may see a burst of traffic as a result of users clicking on it to read more information about the arrest.

3.3. User Accesses Outpace Content Updates

Each news story may see at most several dozen revisions. Between updates, hundreds of thousands of users could see that revision of the story.

4. Content Delivery Requirements in Media

Optimization of content delivery in an online media environment tries to balance several sometimes-conflicting requirements:

Content freshness: The need to deliver the most up-to-date breaking news stories and to disseminate, as quickly as possible, updates to those stories;

Stability and reliability: The need to maintain as near a 100% system uptime as possible even under extreme traffic load;

Minimize system complexity: The business rules used to configure the content delivery infrastructure should be broad, yet simple enough that they can cover most operating scenarios, even in the face of extreme site traffic. Notwithstanding the foregoing, if rules need to be changed, their effects should be instantaneous, since they are generally in response to an incident. Finally, the rules should be understandable by even junior operations staff;

Cost minimization: The need to keep operating costs within budget parameters even in the face of extreme site traffic caused by high-impact breaking news, e.g. Arab Spring or the 2011 Japanese tsunami & nuclear power plant crisis. CBC is also a publicly funded organization and thus vigilant management of both operating and capital costs is vital, since the corporation is subject to extra levels of scrutiny not usually attached to private corporations.[6]

5. Existing CDN Deployment and Problems

Between 1995 and 2006, CBC's website was driven by a family of J2EE applications that rendered news content from a relational database. The design was understandable, given the stated business requirement for content freshness as a top priority. However, a dynamic application was not a good fit for CBC, given the content access patterns described in section 3: the application unnecessarily re-rendered stories for each and every user request, and resulted in many outages under high load.

The original CDN implementation was perceived as an insurance policy against origin failures. However, in so doing, the set of business rules required to implement impose static object caching on a dynamic system quickly grew out of control. The rules were not only difficult to maintain, but often did not meet business requirements: objects were frequently updated before their TTL expired, causing many urgent phone calls to operations staff to purge the CDN cache so users could see the new content. These purge requests would take up to fifteen minutes to be propagated across the CDN's extensive server network of 60,000 servers)[7].

The first step in properly re-implementing the content delivery architecture was to ensure a stable origin upon which to build. Because of the content access patterns described above and lack of content targeting, user sessions or personalization on the site, it was adequate to convert the origin to a largely static delivery system. As such, CBC's news stories today are effectively static HTML documents. The origin consists of an Apache server farm with no dynamic modules. Stories generated from the CMS are converted into HTML fragments containing the story body, headline, associated links, and other display metadata for the user. These fragments are then wrapped by a "story wrapper" template, which uses Server-Side Includes (SSI) to inject story variables into the appropriate areas for display to the end user. (See Figs. 2 and 3.)

```
<!DOCTYPE html>
<!--#config errmsg="" -->
<!--#if expr="${QUERY_STRING}" -->
<!--#include virtual="${QUERY_STRING}" --
>
<!--#set var="newsStory" value="true"-->
<!--#endif -->
<!--#if expr="!${storyBody}" -->
<html>
<head>
<meta name="robots" content="noindex, no-
follow"/>
<meta http-equiv='refresh' con-
tent='0;url=http://www.cbc.ca/404.html'
/>
</head>
<body>Page Does Not Exist - 404</body>
</html>
<!--#else -->
<html>
<body>
<div id="storyhead">
<!--#include
virtual="/cms/html/head/${storyHead}" -->
</div>
<div id=”storybody”>
<!--#include
virtual=”/cms/html/body/${storyBody}” -->
</div>
</body>
</html>
<!--#endif -->
```

Fig. 2: Simplified example of story "wrapper" that includes a story headline & body included by way of an HTTP query string. Rewrite rules (see Fig. 3) ensure that the query string is hidden and remapped to a pseudo-filesystem path.

```
RewriteRule ^/news/story/(.*)
/i/news/v10/storytemplates/default.html?/
cms/html/story/$1 [L]
```

Fig. 3: Example of rewrite rule illustrating how news stories that appear to be static HTML documents under /news/story are actually remapped for processing by the story wrapper mechanism.

While readers may criticize SSI as a primitive technology, lacking features available in more complex languages like PHP or Ruby, SSI is lightweight and fast, runs within Apache, scales extraordinarily well under

high load, and provides good security, provided that the ability to execute arbitrary scripts is disabled.[8] The limited feature set also makes it very difficult for developers to make mistakes by introducing external dependencies, e.g. XML transformation, database connections, etc. Finally, the simple feature set has an unintended, but desirable side-effect: it mandates the use of the content management system as the canonical source of all information associated with a news story. Reporters cannot manipulate the content displayed to end-users without entering it in the CMS, which has an audit trail for legal purposes, such as libel defense.

6. The Origin's Stable; What Do We Do With the CDN?

Once the origin was stable, the team was able to carefully consider the complex nest of origin server and CDN tuning rules.

The authors decided to begin by conducting a thought experiment. What would be the consequences if there was no tuning at all and the origin was exposed directly to the CDN? Obviously, this would expose the origin to high risk under high traffic and defeat the purpose of having a CDN, but it would significantly simplify the configuration. With this thought experiment as a foundation, the team began to really think about the company's conflicting requirements from section 4 and prioritize them. Should all other factors really be sacrificed for the sake of site freshness, even for a news organization? The authors agreed that it shouldn't, and decided upon the following order of priorities:

- Origin stability
- Content freshness
- System complexity
- Cost

In other words, under no circumstances should the origin fail, even under a crushing traffic load. After all, content delivery rules that optimize for content freshness are pointless if one's website is down. Next, the tuning rules must be implemented in such a way that provides the freshest content possible. Third, the business rules must be simple enough that even a junior operator can understand – and if absolutely necessary, adjust them, should a major news event strike in the middle of the night and the origin require extra protection.

Readers may criticize CBC for placing cost last, particularly since the corporation is a public benefit corporation, supported by taxpayers. However, the company mitigates this prioritization in a number of ways:

- Using a local peering exchange (Toronto Internet eXchange[9], or TorIX) to connect to the CDN, whose ingress servers are also at TorIX. In this way, transit bandwidth from the origin to the CDN is effectively free;
- Negotiating with the CDN so that midgress bandwidth is free. That is, bandwidth used to send data on behalf of CBC internally in the CDN's infrastructure is not charged to CBC;
- Choosing a 95th percentile-billing model for HTTP content to mitigate traffic spikes causing high one-time costs;
- Configuring the origin to send appropriate cache-control headers to minimize the amount of content sent to the CDN. (See section 7.)

7. Implementation Parameters

With a general list of priorities in mind, we can discuss the specific parameters that CBC uses to optimize its use of the CDN.

7.1. Set a global site TTL

Using the principle of "sometimes the defaults are good enough", the authors set a global site TTL of 20 seconds for almost all objects except HTML (`Content-Type: text/html`); HTML has a global site TTL of 120 seconds. In conjunction with the other configurations below, the authors have found this to be adequate in handling 99% of all traffic situations. If a major news event occurs, operations staff will carefully monitor the performance of the origin and increase the global TTL if needed. (Such a change could theoretically be automated.)

7.2. Heavily leverage If-Modified-Since

As explained above, after the expiry of a TTL, Akamai edge servers will issue GET requests to the origin with an `If-Modified-Since` (IMS) header set to the time at which it last retrieved the objects in question. The origin is configured to correctly send `HTTP/1.1 304 Not Modified` if the object has not changed, thereby ensuring that object bodies are not sent unnecessarily.

The authors have found this simple configuration to be extremely effective at achieving a high origin offload: approximately 70% of origin requests result in a `HTTP/1.1 304 Not Modified`. (See Fig. 4.)

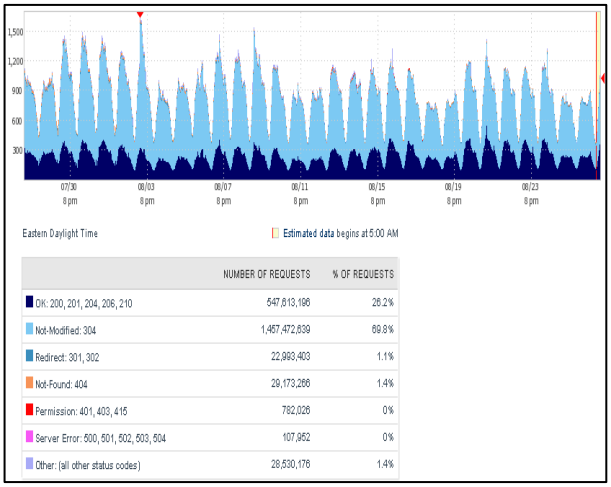


Fig. 4: Distribution of typical HTTP access codes over a four week period, showing that the majority of user requests are serviced by 304 Not Modified responses (in light grey).

7.3. Organize the Filesystem by Object Mutability and Override Global Site TTLs by Content-Type

By organizing the filesystem with top-level directories that correspond approximately to the update frequency of objects inside them, content-type-based overrides can be set by directory. For example, CBC has a top-level includes directory under which global assets like site icons, JavaScript and CSS are deployed through a rigorous change control process. Although CBC could just let Akamai use the global site TTL and issue many unnecessary but harmless `If-Modified-Since` requests, the authors have instead set a TTL by content-type to something relatively high, such as one (1) hour, given that the objects will not change multiple times within that time window.

```
<Location "/includes">
  ExpiresByType "text/css" "access
plus 1 hour"
  ExpiresByType "application/x-
javascript" "access plus 1 hour"
  ExpiresByType "image/gif" "access
plus 1 hour"
</Location>
```

Fig. 5: Illustration of TTLs set by filesystem path and content type.

7.4. Enable Last Mile Acceleration and Origin Compression

Last mile acceleration means that the CDN's edge servers will attempt to use gzip compression to send content to end-users. Origin compression means that the origin will send compressed objects to the CDN. Both are enabled; this way, the CDN merely needs to send payloads directly to the end-user without needing to recompress them.

7.5. Enable HTTP persistent connections and set appropriate timeouts

Some CDNs will attempt to keep a pool of connections open to the origin to avoid incurring the overhead of connection setup and teardown on each user request that results in a cache miss. This requires not only the use of of persistent connections (pconns) in the origin webserver, but also careful tuning of pconn timeouts to match those configured at the CDN.

The Akamai configuration documentation[10] emphasizes that the timeout configured in Dynamic Site Accelerator (DSA) must be shorter than the maximum timeout configured at the origin, to avoid an in-progress pconn being terminated. Since the default value in DSA is 300 seconds, CBC sets the `KeepAliveTimeout` value in Apache to exactly 301 seconds.

7.6. Understand and properly configure all cache-control headers

It's critical to understand and properly configure all cache control headers when interacting with a CDN. The following are critical:

Cache-Control

The global time-to-live value is specified in the `max-age` parameter. `public` should also be specified to ensure that downstream caches cache the content.

Expires

The date the object expires. This may be used by a CDN in preference to the `Cache-Control` header. Generally its value should be the current time (value of the `Date:` HTTP header) plus the TTL.

Last-Modified

As described above, the `Last-Modified` header is sent with all objects with the exception of dynamic content and server-side HTML.

ETag

Entity Tags[11], or ETags, are a hash of the file's inode and last modified time, and can often be an excellent hint to a CDN as to whether or not a cached object is dirty. They should be used with care[12], however, for the following reasons:

- They are incompatible with SSI documents. ETag will be set to the "hash" of the SHTML wrapper even though SSI-included objects might have changed.
- ETags work perfectly for non-SSI documents in a deployment scenario where all webservers are connected to shared storage, thus ensuring consistent object inodes across the cluster, so

long as the webserver's operating system respects the shared storage system's inode scheme. However, in a scenario where each webserver serves objects from its own local disk with different inodes between machines, ETags are not suitable.

8. General Lessons Learned

The authors offer the following general advice by way of summarizing the specific, technical implementation details outlined in section 7.

8.1. Keep caching rules simple

Sometimes the defaults are good enough. For example, many origin HTTP servers already set headers (`Cache-Control`, `Expires`, `ETag`) that are usable hints for downstream content delivery networks, caches, or proxies. The default behavior of these headers is often adequate, but should be verified by system administrators to ensure suitability.

8.2. Tune at the origin first rather than at the CDN

Many CDNs provide the ability to tune content freshness at the "edge", or in the vendor's delivery network. The authors shied away from this approach, at least until origin tuning options were exhausted. There are several reasons for this, aside from the obvious downside of vendor lock-in. First, implementing tuning features directly in the vendor's network increases the propagation time for any changes to those features. For example, Akamai stores customer configurations in XML files ("metadata") that have to be carefully managed and go through a rigorous workflow before they can be deployed to Akamai's entire network of 60,000 servers. This is an appropriate level of paranoia for many features, but excessive in an emergency situation when TTLs need to change immediately due to high traffic. Making TTL changes at the origin and then signaling to the CDN that future requests should be treated differently is the fastest way to propagate those changes across the CDN's network.

8.3. Understand and categorize content before tuning

When tuning – in the form of custom TTLs – is necessary, it's most efficient to group that content together in some way so that one or two rules may suffice. For example, the majority of the TTL rules implemented at CBC are by MIME (content) type, in preference to rules based on filesystem paths.

8.4. Understand what "TTL" actually means

There is often a major misconception among system administrators and users as to the meaning of "TTL", at least insofar as it applies to Akamai Dynamic Site Delivery. "TTL" does not mean the interval at which the entire object body will be re-fetched from the origin; it simply means the interval at which the CDN cache will check the origin for object freshness. If the cache determines, from origin HTTP headers, that the object has not been modified from the copy it has in cache, it will not re-fetch the object body and instead reset the TTL timer. Therefore, it is actually safe to set the site TTL quite low for a site on which objects change relatively infrequently; it can be some fraction of the average modification time of objects.

9. Tools Used To Debug and Improve Cache Performance

CBC has found the following resources useful in examining system behavior, debugging problems, and providing statistics on how performance can be improved:

- The Akamai Luna Control Center (formerly EdgeControl) where system administrators can generate reports in real-time as well as run ad-hoc reports on past traffic.
- The "origin OK volume per URL" report is handy in determining which objects are being fetched from the origin.
- Error reports: To improve performance, the "top URLs, by number of errors" report is useful for tracking down and identifying the top broken links that cause origin requests.

System administrators and developers also use a number of Firefox plugins to troubleshoot CDN behavior by examining and/or modifying HTTP headers:

- Firebug[13] allows users to view HTTP request and response headers to troubleshoot caching problems.
- Tamper Data[14] can also be used to inspect and modify HTTP/HTTPS headers.
- Akamai Headers is a plugin that system administrators have used in the past to decipher Akamai specific information such as the cache key and which edge server responded to your request. However, this plugin is no longer actively maintained by Akamai and may not be suitable for general use.

10. Outcomes

CBC has greatly increased the performance and maintainability of its content delivery network by redesigning its origin architecture and simplifying the configuration of both its origin servers and the CDN. The business requirements are achieved with fewer than 25 lines of tuning in Apache. All business rules are implemented at the origin, meaning that any changes are instantaneously propagated to the CDN.

The entire solution is extremely cost effective from a capital expenditure viewpoint. CBC ran its entire origin with only six webservers between 2003-2010, and today the origin consists of a nine-server cluster of commodity servers with only 40% CPU utilization across all machines.

11. Future Work

Dynamic features, are slowly being added to CBC's website, requiring a careful rethinking of the origin architecture to ensure that we do not return to the mistakes of the past. Some dynamic functionality has been implemented by using Edge Side Includes, which provides a more expressive and powerful processing language over SSI.

At the present time, ESI processing is done by the CDN, but CBC is considering implementing an origin ESI processor and cache, such as Varnish[15], to be able to achieve even more control over caching rules and tuning.

12. Acknowledgements

The authors would like to thank their colleagues in the Digital Operations group at CBC, and in particular David Raso for designing and implementing the Story Wrappers system.

13. References

[1] Dilley, J. et al. Globally distributed content delivery. *IEEE Internet Computing*, September-October 2002,
[3] Noam, E. M. Straining communications systems to the limit. *The New York Times*, September 24, 2011, pp. 4-C.4.
[4] Akamai Dynamic Site Accelerator. Web, September 2012. http://www.akamai.com/html/solutions/dynamic_site_accelerator.html
[5] Nottingham, M., et. al. Edge Side Includes Language Specification. Web, September 2012. http://www.w3.org/TR/esi-lang
[6] Government of Canada. Access to Information Act. Web, September 2012. http://laws-lois.justice.gc.ca/eng/acts/A-1/index.html
[7] Economou, G. How Akamai Maps the Net: An Industry Perspective. Web, September 2012. http://www.akamai.com/dl/akamai/economu_mapping_the_internet.pdf
[8] Security Tips – Server Side Includes. Web, September 2012. https://httpd.apache.org/docs/current/misc/security_tips.html#ssi
[9]About TorIX. Web, September 2012. http://www.torix.ca/about.php
[10] Akamai Edge Server Configuration Guide. Proprietary information of Akamai Technologies. May 2012.
[11] Fielding, R., et. al. RFC 2616: Hypertext Transfer Protocol (HTTP/1.1). Web, September 2012. http://www.w3.org/Protocols/rfc2616/rfc2616.html
[12] Yahoo. Best Practices for Speeding Up Your Website: ETags. Web, September 2012: http://developer.yahoo.com/performance/rules.html#etags
[13] FireBug. Web, September 2012. https://getfirebug.com/
[14] Tamper Data. Web, September 2012. http://tamperdata.mozdev.org/
[15] Varnish Cache. Web, September 2012. https://www.varnish-cache.org/

Building a 100K log/sec logging infrastructure

David Lang - Intuit

Abstract:

A look at the logging infrastructure that one division of Intuit built that included the requirement to handle 100K lines of logs per second with the logs being delivered to several destinations (including proprietary appliances).

This paper will cover the options considered, the choices made and the problems we ran into. The most unusual and interesting topic discussed is the method selected to distribute the logs to all the different destinations, able to deliver the log message to several different load balanced farms of servers with only one copy being sent over the wire.

Introduction

Digital Insight (acquired by Intuit in 2007) built a new logging infrastructure in 2006. It replaced the prior logging solutions that had been implemented by individual teams that had then evolved . We provide Internet banking services to approximately 2000 banks and credit unions, providing everything from the Internet facing home-page to the connection to the bank's main systems. Over time and acquisitions, the environment had became very complex with 135 different networks, approximately 300 firewalls organized into about 100 sets, and over 1000 routers and switches. We needed a logging infrastructure to support the Security/Audit requirements of being able to go back in time and figure out what happened even months after the event when someone complained and as we looked at requirements we found that something that would satisfy these requirements should be very close to satisfying operational requirements as well, so we enlarged our scope. The final requirement document can be summed up as:

- Gather all logs generated by any software or hardware in the company.
- Have an alerting engine that generates alerts based on individual or combinations of log messages.
- Allow for rapid ad-hoc searching of the logs, both for fraud investigations and for troubleshooting.
- Maintain an archive of logs for many years (data retention policy set by the Legal department and driven by the need to provide logs of financial transactions to banks)
- Generate periodic reports summarizing data in the logs
- Be able to run for at least three years without needing any architectural changes. Proactively identify what the expected bottlenecks would be, and produce plans to address them.

At the time we had been growing rapidly, with the historic Internet traffic and log traffic approximately doubling every year for the 7 of the prior 10 years. As a result we estimated that to have the new logging system be able to last at least three years, it needed to handle approximately 100K logs/second.

In the following years the peak traffic and logs 'only' grew at about 60% per year, and the traffic became more concentrated, so the total log volume has only grown about 40% per year. In addition, about 3/4 of the possible logs still aren't getting fed into this system due to other priorities preempting the logging work, so the log volume did not grow to the design level. There have been peaks of logs where we have received >92K logs in a single second and the architecture has held up as expected. Normal peaks routinely generate 30K logs per second.

Architecture Design and Software Selection

The thinking for this project started in 2005 when we considered just adding Sensage to our existing logs to make it faster to search them. Initially Sensage quoted us a price, but when we went back to them a while later, they revised their price up by a factor of 4x. As a result, this moved from being a simple department purchase of software to a $1M USD project requiring much higher levels of approval.

We created a RFP and sent it out to several vendors, but in addition I was assigned to evaluate the options to build a vendor-neutral solution ourselves. This vendor-neutral solution could use whatever software we wanted, but it needed to be designed in a way that we would not be dependent on any one vendor. Any vendor's product could be swapped out for a different vendor's solution for that component without having to make major changes outside of the parts being changed.

We evaluated Arcsight, Sensage, Splunk, Nitro Security, and Greenplum as possible vendors. After evaluating the different responses to the RFP we decided to go with the vendor-neutral option. At the time only Arcsight claimed to be able to meet our requirements, but they wanted to charge us for each website that we host, which resulted in a pre-negotiation price of $40M being quoted. The other vendors either couldn't handle the 100K log/sec load, or couldn't handle the alerting/reporting functions. Splunk came the closest among the other options, but at the time their alert capability consisted of scheduling saved searches.

The architecture we ended up settling on uses *rsyslog* for the log gathering and transport, delivering the logs to multiple farms of servers simultaneously, with each farm focused on providing a specific set of functions.

Gathering Logs

We started by saying that most of the logs that we needed were already generated through syslog and so we decided to see if syslog could satisfy our requirement for log transport. Since many of our devices generated UDP syslog, there was a strong desire to be able to be able to support UDP syslog at least on the initial leg of logging, switching to TCP syslog or other protocol only if UDP was not reliable enough.

To test the syslog implementations, I created sample log messages using logger and captured the resulting network traffic with *tcpdump*. Then using *tcpreplay* to replay the traffic we performed tests with *sysklogd*, *syslog-ng*, and *rsyslog*. The tests consisted of setting *tcpreplay* to re-send the known traffic at a set speed. We then examined the logs written by each syslog implementation to see how many log messages were received. We then ramped up the message rate. This was not a formal study, but the rate of message loss grew so rapidly that a graph of the percentage of logs that were received would have looked like a cliff, so we just started talking about the approximate message rate at which the collapse happened. Over time most of the records of the testing have been lost so all that we have to go by is the records left in e-mail discussing the options. The initial responses were disappointing as none of them seemed to be able to handle anything close to the 100K logs/sec load we were predicting.

The *sysklogd* daemon we tested had already been tweaked (disabling name lookups, eliminating system calls and a streamlining a few other areas), but since it is single-threaded and had to write one message before starting to process the next, it started losing logs at a couple of thousand logs/sec, gradually losing higher percentages of the logs as traffic volumes increased

Syslog-ng seemed to hit a wall around 1K logs/sec and just dropped messages above this rate. I never was able to understand this. When I asked for help, the general answer that I got was to just use TCP.

Rsyslog handled short peaks up to about 30K logs/sec as it processed the incoming messages into a memory queue, but could only write out a few thousand logs/sec, so if the traffic spike was longer than the memory could handle it started losing massive amounts of logs.

Since *rsyslog* was the best of the available options, I investigated what was going on with it in more detail and immediately noticed the huge number of system calls that it was making per message, including four *gettimeofday()* calls per log message to track the message as it moved through *rsyslog*. The maintainer, Rainer Gerhards, was very responsive to suggestions for changes and performance rapidly improved. As the performance increased, the bottleneck became the internal locking of the message queues. Intuit sponsored some development to to add the ability to process multiple messages at once to reduce locking on the internal queues and be able to do database inserts of multiple records at a time. and within a few months we were seeing tests where *rsyslog* was absorbing spikes of 378K logs/sec (effectively Gig-E wire speeds with the 256 byte log messages that measurements on existing logs showed to be about average for out environment) and writing to disk at >78K logs/sec with no

dropped log messages. Since that time performance improvements have continued with some people now reporting success with end-to-end tests over 1M logs/sec.

Transporting Logs

Due to the large numbers of networks that we have, and the fact that many of these networks were isolated by proxy firewalls, we decided to implement a set of syslog relay servers. We built syslog relay servers in HA pairs and put an interface for a relay server on 90 of the major networks and accepted the risk that unreliable UDP syslog messages may be blocked by the router choke points from the other networks.

It turned out that another use for these relay servers is to clean up 'non-standard' syslog messages. Many devices send syslog messages that don't quite match the RFCs and as such cause problems when they are parsed and analyzed in combination with other messages. One example of this is that Cisco routers can be configured to send logs in the format

<pri>timestamp IP tag: message

or

<pri>timestamp name : tag: message

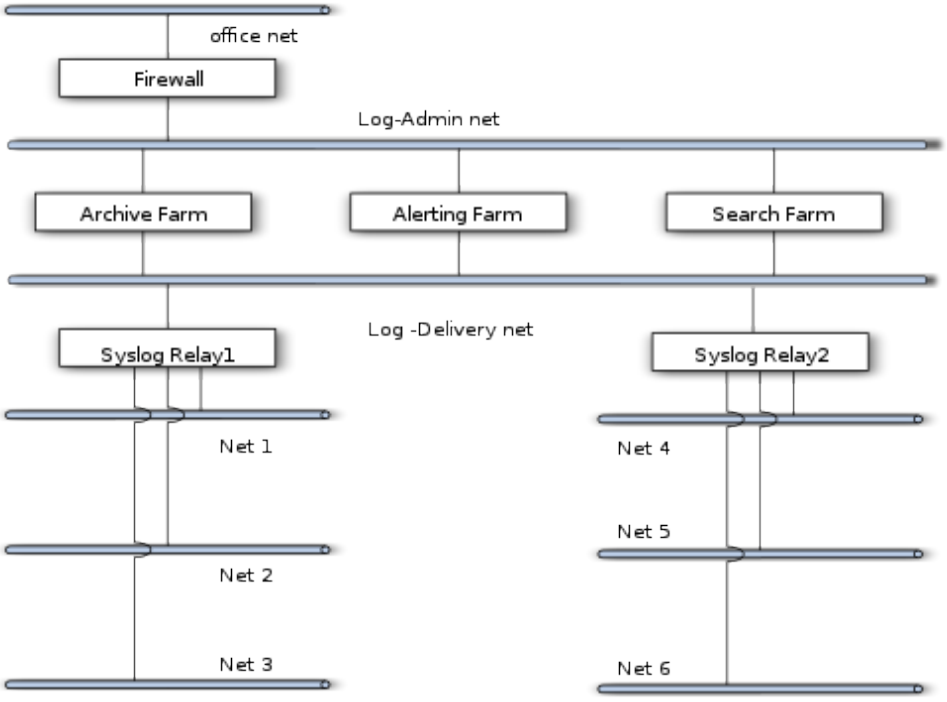


Illustration 1: Simplified Network Diagram

If you try to process these messages normally you end up with the syslog tag being just ':' instead of the %ASA-#-###### type of tag that is far more useful for filtering. What's worse is that if you have some Cisco devices configured to log by IP and some that log by name, you end up with the data fields misaligned on logs that are otherwise identical.

Initially we used the *rsyslog* custom formats and filtering logic to

detect and correct these log messages. Recent versions allow for custom parser modules to be written and loaded to fix the logs as they arrive on the system. I wrote and contributed several parser modules to deal with the broken logs we have encountered.

Delivering Logs

With the large volume of logs, we recognized early on that most of the processing of the logs would require more than one machine, and so we needed to figure out how to reliably deliver the logs to multiple destinations, and how to load balance across multiple servers for one destination. One issue that we realized early on was that at the data volumes that we were talking about supporting, sending multiple copies of the logs over the wire was going to start running into performance problems. With a goal of 100K logs/sec and Gig-E wire speed being just under 400K logs/sec, just sending four copies of everything would exceed network capacity. If we used a hardware load balancer, we would need multiple load balancers to avoid the interface on the one load balancer becoming the bottleneck. Moving to 10G Ethernet was considered as an option, but at the time the cost of of 10G Ethernet was several thousand dollars per port.

Instead we ended up using Multicast MAC traffic. Under Linux this is called CLUSTERIP. Normal IP Multicast uses the 224.0.0.0/8 address range and requires that the software involved be multicast aware. Multicast MAC on the other hand is a feature of Ethernet and works underneath IP. Multicast MAC sets the low bit on the high octet of the MAC address to 1 and the ethernet switches recognize this and send the traffic out to multiple ports on the switch. CLUSTERIP uses this by having the receiving system hash the connection information (one or more of Source IP, Source Port, Destination Port) and then divide the resulting hash into a number of buckets. It then passes any packets that match the buckets assigned to the local machine up the stack. This requires no changes in either the sending or receiving software. Everything is done in the network stack.

Example:

Let's say that we have the following three connections:

1. source 192.168.1.1 port 1025, destination 192.168.1.5 port 514 hashes to 13
2. source 192.168.1.1 port 1026, destination 192.168.1.5 port 514 hashes to 14
3. source 192.168.1.2 port 1025, destination 192.168.1.5 port 514 hashes to 15

We then say that there are 3 buckets in the current cluster, so we use the modulo function to assign these connections to nodes

1. hash 13 % 3 = node 1
2. hash 14 % 3 = node 2
3. hash 15 % 3 = node 3

The system that is configured to be node 1 of 3 then will process connection #1, the system that is configured to be node 2 of 3 will process connection #2, and the system that is configured to be node 3

of 3 will process connection #3. As long as you have a system configured for each bucket, all connections will be handled by some system.

This is done in IPTables with a command like:

```
/sbin/iptables -I INPUT -d 192.168.1.5 -i eth0 -j CLUSTERIP --new --clustermac
01:02:03:04:05:06 --total-nodes 3 --local-node 1 --hashmode sourceip-sourceport
```

This works with any sort of traffic (the port numbers obviously are only applicable to TCP or UDP traffic), with the only significant drawback being that all the traffic for the cluster will hit the kernel of each box before being dropped.

I recognized the fact that the switch distributing the traffic doesn't know or care which of the systems are actually processing the traffic. For connectionless protocols like UDP syslog, this then means that you can define your IPTables rules so that more than one box handles any given packet.

So if you configure two different systems with the rule:

```
/sbin/iptables -I INPUT -d 192.168.1.5 -i eth0 -j CLUSTERIP --new --clustermac
01:02:03:04:05:06 --total-nodes 1 --local-node 1 --hashmode sourceip
```

both systems will receive and process all traffic sent to 192.168.1.5.

In addition to this you can have the three machine cluster mentioned above, and each machine on that cluster will receive 1/3 of the traffic. There is no need to have the same number of boxes in each cluster, and the systems sending the traffic don't need to know how many boxes are in each cluster, or even how many clusters there are.

This allows us to add a farm of servers to the network by just configuring this multicast MAC on the system and then remove it later if desired without having any effect on the other systems.

By using this approach, the limiting factor is the port speed of the receiving boxes, in this case 1 Gb/sec or just under 400K packets/sec. This met our three year goals with about 4x headroom and the obvious upgrade path of moving to 10G ethernet when we ran into the limit (with the expectation that in 5 or so years when the limit would be hit, the cost of 10G equipment would have dropped significantly)

There were a lot of questions about the reliability of this approach, especially since it required us to use 'unreliable' UDP for the logs. As a result we ended up doing long-duration testing with multiple sources sending logs to multiple sets of destinations and we found that with a relatively low-end Cisco switch (3550) we were able to send several trillions of log messages in wire speed bursts of up 120GB per burst with zero lost messages as long as we paused between burst to let the slower file I/O that *rsyslog*

had at that time catch up. This testing was possible due to some machines with 128G of ram that were waiting to be installed, that we hijacked for a couple of weeks for this testing.

Since we wanted to avoid any possibility of other traffic interfering with the log delivery, we designed the core logging servers to be dual homed, with one network being used only for log delivery, and the other being used for queries and administrative traffic.

Log Analysis

With this capability in hand, we then built out several farms of servers to receive and process the logs.

Archiving:

We are using a simple *rsyslog* server pair writing to flat files that then get compressed and rolled to long-term storage (initially a 16x1TB RAID 6 array)

Reporting:

We wrote a series of scripts (currently a combination of Bash, Awk and Perl) to implement Artificial Ignorance filtering of the logs.

This consists of:

- Filter out messages, but count messages that are known to be uninteresting (the number of times an uninteresting thing happens may be interesting)
- Split logs which you recognize off to separate scripts to summarize them.
- Then sort the remaining messages by the most common messages and generate a report

This is still being done on the same farm as the Archiving. Periodic re-writes and optimizations have allowed the reporting to be completed in a reasonable time frame, but as the number of logs increases, this will eventually need more servers and we will then have to decide between creating a dedicated farm of servers for the reporting, or splitting the archiving across multiple servers.

Alerting:

We had significant disagreements internally over the choice of alerting tools to use (the “buy vs build” discussion). Management strongly favored of the “buy” approach but decided the marginal cost of buying two more servers to allow the people advocating the 'build' approach to implement Simple Event Correlator was small enough to allow us to both implement Simple Event Correlator and buy

Nitro Security's log analysis and alerting product.

We discovered that Nitro Security not only could not handle the load, but also that it could not handle the concept of syslog messages being relayed by other systems. It classified log entries based on the IP address the received packet came from, not the system name in the syslog header. Once it classified a log message, it only applied the 'applicable' analysis to the message. This meant that if the first message that arrived from a syslog relay box happened to be a Cisco ASA firewall, Nitro would decide that all messages from that IP were Cisco messages, and only apply the Cisco analysis rules to it. To try and work around this, I implemented and contributed a module that allowed *rsyslog* to forge the source IP address of UDP packets and implemented a set of systems to accept the messages and relay them to these appliances. While I was doing this, we implemented a work-around of implementing *rsyslog* filters on various other systems to relay logs of a specific type from a specific system. There were still serious performance problems, and even this work-around meant that Nitro believed that it had one system of each type reporting to it. Due to these other problems, we never got around to deploying the *rsyslog* forgery module in this environment.

We initially implemented Simple Event Correlator on a pair of 'high end' systems, 4x dual-core processors with 128G of RAM. Those boxes have kept up with the load with our current rule sets, so we have not needed to expand the farm. We have *rsyslog* filter logs of different types to different instances of SEC, each of which only sees a portion of the logs and only needs to deal with a portion of the rule set. This also has the advantage of allowing us to leverage additional CPU cores in spite of the fact that SEC is a single-threaded Perl program. We recognized that this approach had limited scalability, and we initially were thinking in terms of setting up memcached servers to allow for state to be held independently of the servers. However, in 2010 Paul Krizak from AMD submitted the LISA best paper “Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)”[1] that talked about an approach of using syslog-ng to filter the logs going to several rule engines (perl scripts) and then taking the output of these rule engines and passing them via syslog to another rule engine for generating the alerts. Since then our scalability plan has evolved to follow a similar approach, using *rsyslog* to filter alerts by type, going to many instances of SEC, and then SEC setting global state by generating specially formatted syslog messages that could then be correlated by a 'master' SEC instance.

Searching:

For rapidly finding things within logs, we evaluated Sensage, Splunk, and considered rolling our own with Greenplum (a PostgreSQL derivative that can scale horizontally), but ended up deciding to use Splunk. Splunk provides a good user-friendly search interface with a pricing model of charging per gig of logs processed daily instead of per system that allows us to scale the hardware up to get better search performance without costing more than hardware. Sensage provided good scaling, but was very expensive and we would have had to write our own front-end for it. The PostgreSQL approach would have required us to become expert DBAs as well. Those of us with programming skills on the team had already shown that we were not particularly suited to user interface design.

1 http://static.usenix.org/events/lisa10/tech/full_papers/Krizak.pdf

Initially we built a by-the-book Splunk cluster with two forwarders receiving logs, feeding to a farm of four indexers, each with two dual-core CPUs, 16G of RAM, a mirrored pair of 300G 15K rpm drives, and a RAID 10 array of 10 1TB SATA drives. We ran in this configuration for a year and found the query performance was getting unacceptably slow, taking over 10 minutes for some queries to complete. We also had problems restoring data when one system got corrupted.

It was clear that we needed to upgrade our cluster, so we called in Splunk Professional Services and worked with them to do several weeks of performance testing on different hardware (making use of the year's worth of data and real-life queries that we now had).

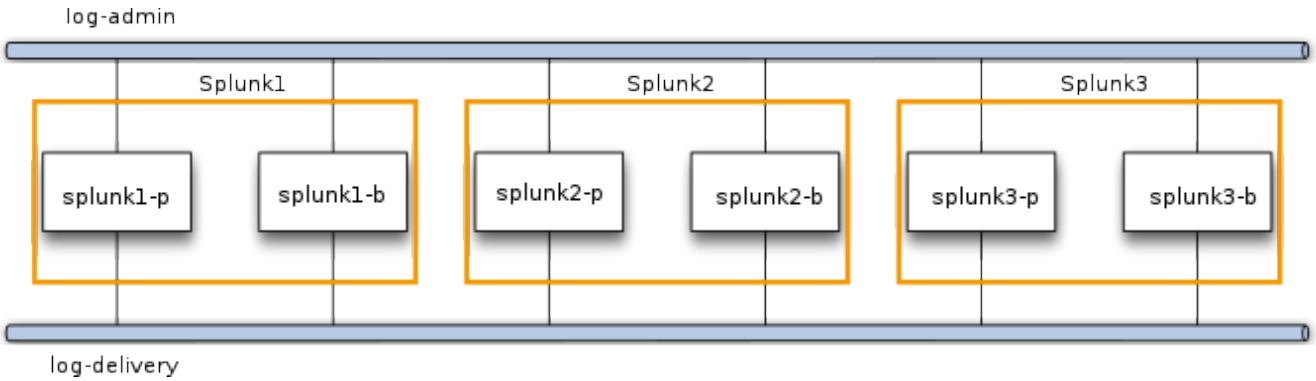


Illustration 2: Splunk Cluster Architecture

We also used their assistance to design and test (but have not yet fully implemented) a work-around for the lack of High Availability support in Splunk. The approach was to arrange the Splunk boxes into the equivalent of a RAID 10 array by configuring the machines into failover pairs. Each pair would be one logical node handling a slice of the traffic. Both the primary and the backup would be configured as the same CLUSTERIP node so they would both receive the same traffic. *Rsyslog* receives the logs and writes them to a file and every minute a cron job rotates these files and hands them off to Splunk. When it is time to roll the buckets in Splunk, each box checks to see if it is primary or backup of its pair. If it is primary, it copies its local copy of the bucket to the backup box, and if its backup, it deletes the bucket. This is not a perfect setup as the logs won't roll at exactly the same instant on both boxes of a pair, and it takes time to roll and copy the logs, so when a failure happens some logs will be lost or duplicated. The advantage of this approach compared to just letting the primary and the backup run independently (which would not have this loss/duplication potential) is that this keeps the log archives consistent between the primary and backup, so that after service is restored on a down box, its archives can be brought up to date with a simple rsync.

Another option that we considered, but haven't yet implemented, was to distribute the search load across both the primary and backup machines of a cluster. This would require running additional Splunk instances on both systems and having the bucket rotation script distribute the buckets across the

instances.

In our performance testing we discovered that for workloads that are close to read-only, as the Splunk searches are, RAID6 is just as fast as RAID10 as both can keep all spindles in use. We also upgraded the second-generation boxes to quad-core CPUs, 64G of RAM, a 64G Intel X25E SSD, and RAID 6 array of 16x 1TB SATA drives. The combination of the more powerful systems, as well as moving from four active indexers to 10 improved the performance of Splunk by about 16x. According to our testing, using the backup boxes for the search load will comfortably double the performance.

Lessons Learned and Problems Encountered

Overall this architecture has held up well. Other than upgrading the Splunk cluster, there have been no architectural changes to the base system. With the exception of normal patching, upgrading, and fixing failed boxes, the systems have pretty much "just worked". Other than the Splunk cluster, the only hardware upgrade that has been required was to upgrade the RAM on the archiving/reporting server as the working set for the nightly report had grown beyond the original 16G. Currently the box has 64G in it and the working set looks like it's about 40G.

The log reporting scripts have had extensive additions, and have been restructured or re-written when the log analysis runs start threatening to take more time to run than the time between runs. We've reworked them 3-4 times so far. and they have been a great example of "throw something quick-and-dirty together and worry about optimizing it later"

Since we have been using and implementing some of the latest features in *rsyslog*, this has resulted in us running development and git snapshot versions much of the time. This has occasionally resulted in a few problems and late nights, but overall things have worked well.

The only real technical "gotcha" we have run into is that the logs generated by the systems receiving the multicast MAC packets must not be sent to the common address. Otherwise, the Linux kernel "optimizes away" sending the packets over the wire, which prevents the other farms from receiving the messages. So the systems receiving the multicast logs must instead be set to relay-locally generated logs to one of the relay boxes for the relay boxes to bounce back to the farm address.

The multicast MAC logging approach has worked spectacularly well, and we've added and removed several products that people wanted to use over this time. New admins tend to be concerned about it, but then relax fairly quickly after they see it in operation for a bit.

The biggest problem has been the political balance between giving everyone access to logs that may

contain sensitive information, and letting people who can benefit from the system get access to it.

The security group had decided several years earlier to use Debian as our Linux distribution, and since we also used our standard image for firewalls, it was a very stripped down version which included a half dozen custom compiled packages for cases where we wanted different defaults, or where newer versions had features or bugfixes that we needed that weren't backported to the versions that Debian provided. The standard when we started was Debian 3.1 32 bit, and we moved to the 64 bit build very early on in the process and were running 64 bit kernels, even with the 32 bit builds.

Summary.

Creating a high volume logging infrastructure takes attention to detail, but the tools that are available and a standard part of all of the Linux Distributions now make it possible to build a logging infrastructure that can handle very high volumes of logs without locking yourself into proprietary solutions, and without writing a lot of custom, performance critical programs.

About the Author.

David Lang is a Staff IT Engineer at Intuit, where he has spent over a decade working in Security Department for the Banking division. He was introduced to Linux in 1993 and has been making his living with Linux (and using it as his desktop) since 1996. He is a Extra Class Amateur Radio Operator and served on the Civil Air Patrol California Wing Communications staff, where his duties included managing the state-wide digital wireless network. He has been running the wireless network at the Southern California Linux Expo since 2010. He is also active on various Open Source mailing lists. He can be contacted via e-mail at david@lang.hm, by phone at +1 818 292 7015

This paper and related materials are available at http://talks.lang.hm/events/LISA_2012/logging

Building a protocol validator for Business to Business Communications

Rudi van Drunen, Competa IT B.V. (r.van.drunen@competa.com)
Rix Groenboom, Parasoft Netherlands (rix.groenboom@parasoft.nl)

Abstract

In this paper we describe the design and implementation of a system essential to enable the deregulation of the energy market in the Netherlands. The system is used to test and validate secure communications using XML messages through the AS2 standard between the business partners in the market. The tool is comprised of an Enterprise Service Bus component, a service virtualization component, a database with business logic and an user interface added. The version 1.0 of the system was built in less than one month.

1. Introduction

The energy market in the Netherlands has been deregulated. To facilitate this a communications structure is designed in which the different energy suppliers are able to communicate to each other. This communications structure and protocol is critical in the supply of energy (electricity, gas) from supplier to consumer. To validate these protocols a simulation and validation environment is required, and used for certification of the different market parties. This experience report discussed the implementation of such an certification environment for the gas transportation; more of these tools will be required for other market processes and will be constructed in a similar fashion.

The exchange of messages in the protocol is built up in different layers. The transport medium is the Internet, and every supplier has connected their production systems to the Internet, to reach all others. In the next layer the transport protocol is http, secured by ssl, so we're using standard https here. Then the data protocol is AS2, which provides a way to send (XML) data in an encrypted manner using S/Mime to an authenticated receiver. The actual payload is an XML message that should adhere to the Dutch energy interconnection standard.

The goal is to provide a test environment (in the project called Testtool) that can be used to certify over 100 market parties to see whether they adhere to a new XML definition that is being rollout during a migration process taking about one year. There are over fifty application and protocol level test-scenarios that need to be checked for each party before they are certified and can participate in the new communications infrastructure.

2. Architecture

To validate the complete communication stack the engine should be able to validate and test the communication on every level. Building this engine from scratch is a major undertaking, and will not be flexible. Therefore we choose a number of building blocks to technically implement the validation engine.

The https and AS2 communications are being handled by an Open Source Enterprise Service Bus (ESB) solution (UltraESB[1]), which then hands the XML payload to a product called Virtualize[2]. Virtualize is used as a service virtualization engine which tests the validity of the XML message, handles responses and stores the data in a database. The actual validation (as part of the certification process) is done using the data in the database and checking on the right contents and sequence of the XML payloads. The database is a MySQL database that stores all messages and metadata of the messages and the sender / receiver. All of this is driven by a number of Java classes that can be controlled by a Web GUI.

`Figure 1` shows the global architecture of the Testtool. The different messages (notification (NOT) and Message Delivery Notice (MDN) as Acknowledge) are asynchronously for external communications and synchronously for internal communications. The external party is simulated in the test environment as depicted using the Meldelson[3] tooling.

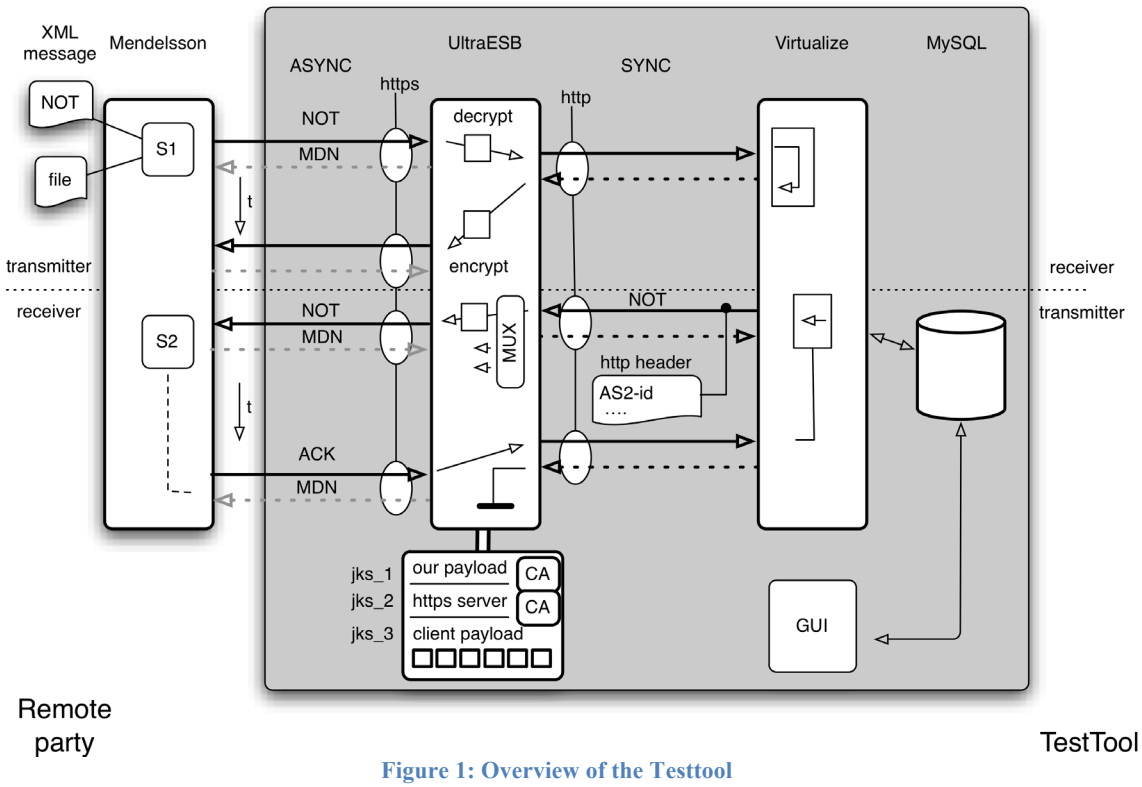


Figure 1: Overview of the Testtool

2.1 AS2

The protocol on AS2 level allows for multi level encryption of the data on the wire, as well as validation of the communication partners using certificates. Next to that communications can take place in a synchronous and asynchronous manner.

The AS2 message enters the system through a https connection. A standard server certificate, issued by a trusted certificate authority is used to validate the sender and decrypt the ssl transmission. The setup corresponds with the way an ssl-enabled webserver is set up. Important is that the Testtool, as well as the partner, can work both with one and two-way SSL.

As the ESB solution incorporates the webserver, the server certificate, as well as the (path to the) root CA needs to be present in the Java keystore it uses (a so-called `identity.jks`). This setup is shown in Figure 2.

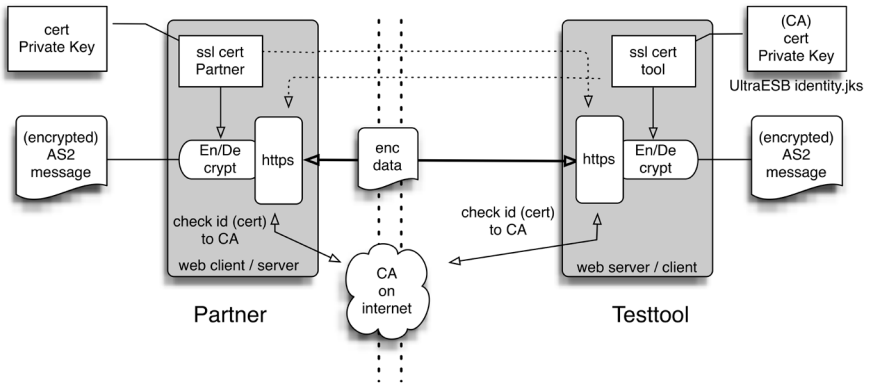


Figure 2: ssl setup for AS2 messages

The OpenSource ESB we employ in this project is UltraESB. This engine has the ability to handle the (AS2) communication with the partner in a synchronous and asynchronous manner, while it communicates in a synchronous fashion to the backend (in this case the Virtualize engine). Configuration and adding functionality to UltraESB turned out to be simple by changing the configuration files that are a mix of Java and XML.

As the protocol supports bi-directional communication we need to not only accept messages but also be able to send them. We needed to add extra functionality to the ESB to facilitate sending messages to partners, when the test environment is initiating the communication. We did this by adding functionality to select the correct certificate that belongs to the business partner with whom we need to set-up the AS2 communication. A custom http-header field: `dest-endpoint` is used to communicate this information from Virtualize to UltraESB.

Second, when we are receiving the, in this case asynchronous, acknowledgement message we should end the handshake directly and not send a formal reply from the backend to the business partner again. In that

situation, the ESB would not forward the ‘response’ message, but direct it to "ground". We used this approach to have a clear separation of concern: the ESB deals only with the communication, only the Virtualize components are ‘aware’ of the type of message. If it is an incoming notification message that needs acknowledgement, Virtualize responds. If the incoming message is an acknowledgement we do not have to respond Virtualize adds the “ground” designator to the response message.

As there were no throughput requirements, as the system was to handle one communication at a time, with a payload maximum of about 1 Mbyte. We did not expect issues with buffering, and relied on internal buffers of the ESB. In practice this has proven to be sufficient.
The ESB allowed us to set the maximum payload size, which will be set to 10 Mbytes in the production version. Together with the appropriate settings of the java VM (max heap size) we were able to handle this maximum payload size without problems.

2.2 PKI

To authenticate and encrypt / decrypt the XML messages on AS2 level a Public Key Infrastructure is being used as shown in Figure 3. Every partner in the energy market in the Netherlands has a certificate that is signed by a root that is public for the Dutch energy market, and is maintained by the government. This PKI is used in the ESB software to maintain authentic and safe communications. The path to and the root CA, as it is not a standard trusted CA as well as the certificates of every business partner must be known to the ESB application. The local certificate must be made known to every partner as well. The ESB certificates are maintained in a different Java keystores and controlled by a GUI.

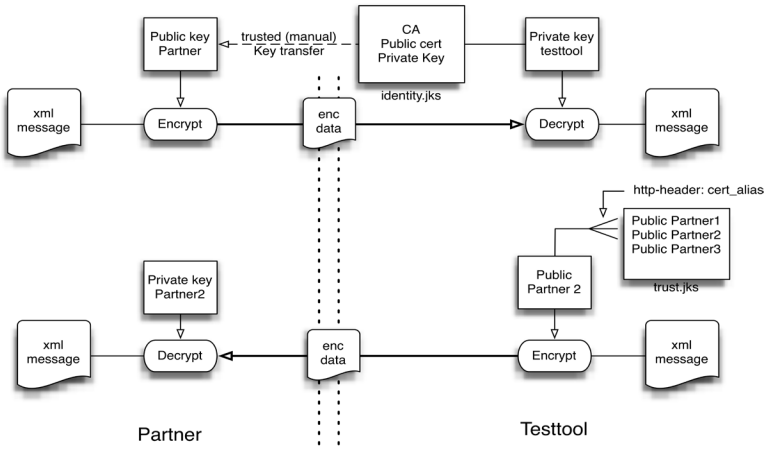


Figure 3: Message level encryption using PKI

If an AS2 message enters the ESB, it will be authenticated by checking the certificate against the CA, and then decrypted with the local (private key) certificate. If the ESB needs to send a message, it encrypts it with the public

key of the partner and sends it on the wire. The Virtualize component used 3 http-headers to tell the ESB which certificates and endpoint to use.

For testing this process, we have used a separate AS2 client (Mendelson) that we can load with the certificates of the business partner so we can simulate and test the AS2 communication before bringing the application 'live'. This helps in the troubleshooting of the correct loading of the certificates.

2.3 Virtualize

The simulation of the service that handles the communication between the partners is implemented using Parasoft Virtualize. The Virtualize toolkit was readily available and had all the functionality we needed for the backend. Next to that the customer had ample experience with other Parasoft tooling. In this setup Virtualize accepts XML messages on a Tomcat queue, checks the messages against the XML schema definitions and validates them. Based on the message content, Virtualize generates the response messages for a specific partner and sends it on to the ESB as http message. To let the ESB know which partners information to use in forwarding the message, Virtualize adds three custom http-headers:

`dest-endpoint`	AS2 endpoint where the ESB should deliver the message
`cert-alias`	The alias of the certificate in the keystore in the ESB that should be used for the encryption of the MIME payload
`as2-id`	The AS2 identifier of the business partner

Virtualize supplies the parameters to these headers based on the business partner identifier and the information stored in the backend database.

To be able to mimic asynchronous behavior, we need to be able to have a message that is generated in Virtualize not to be forwarded to the partner as described before we add the "ground" designator as value to `dest-endpoint.`

Using the scripting facilities of Virtualize we created python methods that are fired off when specific messages arrive. These scripts log the (validated) message and metadata in a database to be inspected on the contents, and respond though the ESB in a correct AS2 message to the client.

2.4 Database

In order to check the contents of the XML and its validity, the message is logged in a database. Next to the message itself, meta information on the business partner and timestamps are stored. This information than can be used to actually check the sequence and detailed contents of the XML message and report to the partner.

Besides storage of the incoming messages, the database also contains the different messages the tool should be able to send for certain test-scenario. This requires a trade-off between storing complete messages or only the relevant payload information and let Virtualize compose the correct XML message with right information. The database offers enough flexibility to support both approaches.

2.5 Database application

The application to validate the message transfer is a Java application that supports the process of validation of a partner, in which a number of test cases need to be performed. Each test case is a scenario in which a number (>= 1) of messages containing specific information need to be sent to or received from the test tool. The application can evaluate the database contents and report on the correct sequence and contents of the messages, and then flag a test (case) as succeeded or not.

Next to this functionality the database application implements a portal for maintaining clients, and querying the database for certain test scenarios executed by partners, or provide a partner with the ability to have the test system initiate a test in the direction of the business partner.

3. Experiences / Usage

The described project has a high business value, as partners are obliged to certify themselves before being allowed to do business on the Dutch energy market. Certification ensures the peer-to-peer communication, and thus the national protocol will work. As market introduction depends on certification of the partners the deadlines are pretty tight. The development team managed to build a version 1.0 of the system within a month. This can be regarded as a good accomplishment, also given the fact that the messaging specification on a functional (which test-scenario's to support) and technical level (final versions of the XML schema definitions) was still evolving during the development.

Problems encountered on the way were partly due to the different ways the business partners have set up their production systems and the PKI. During the process of testing we encountered problems in validation of the certificates as different partners had certificates signed by an older version of the root certificate, and thus were not able to be authenticated. The error messages from UltraESB did not point us in the right direction right away. This problem was solved in the checklist and standard operating procedure.

Other issues that we have seen were due to the fact that the system was required to run on a Windows platform that was present as a VPS in the cloud. The acceptance system and the development systems were set up for a single developer use so making parallel development more complicated.

Technical issues were seen in the management of the Windows firewall through the Java application, as we wanted to close the system off for all other IP numbers than those of the partners that registered for the validation. Managing the Windows firewall rules was difficult, so we implemented access lists on the Apache webserver as part of the ESB. Other difficulties were encountered in the file format of certificates and the multiple different keystores. Also the inner workings of some of the products used were not as expected, and needed adaption in the user code.

The next (2.0) phase of the project, currently on its way features the building of several web interfaces to facilitate the management of partners and the certification / validation process, in such a way that partners can start up their own validation process and see the progress of their certification.

4. Lessons Learned

A number of lessons can be learned from this project.

First of all, separation of the three main functionalities of Testtool, has been key to the success of this development. Using an of-the-shelf ESB to handle the (AS2) communication, the Virtualize engine to handle the XML payloads including technical validation of the message and the database application to support the actual validation process. This clear separation in architecture has proven to be important throughout the development process, although fine-tuning the interface between these three components required frequent detail adjustments.

Second, the modular design of Testtool, turned out to be very helpful to handle the changed specifications. Virtualize was programmed using a draft set of XSDs that in a later stage where replaced by the final versions. The core functionality depended only key (stabled) elements in the header of the XML message. Similarly, incomplete sets of the certificates influenced only the testing of the AS2 connectively not the overall development of the Testtool.

Another important issue in this project is that a solid PKI should be one of the requisites for starting. Designing a PKI and setting it up during a project that technically uses the PKI components yields to issues in communication.

An observation during the project resulted in the best practice of having infrastructure development to be ahead of software development. This will result in having a solid base to plan tests. In this project the infrastructure was designed and built alongside of the software environment, which resulted in some planning issues, especially with the tight deadlines we were facing.

Finally, the complete application could have been built from scratch, but as this was not an option within the project, readily available and re-usable “off-the-shelf” components were found in UltaESB and Virtualize. Even the custom build Java GUI will be re-usable for other certification environments that are

required in the future. The combination gives a flexible setup to handle changes in transport protocol, message formats and test-scenario's.
The use of the UltraESB and Virtualize combination turned out to be the right choice for building this tool.

5. Conclusion

Without a validation system and the system driving it, it is not possible to have a large number of partners using a complex protocol securely and reliably amongst each other, a requirement in the Dutch deregulation of the energy market. The validation service has moved to full production successfully. Whenever a new partner wants to join the energy market the validation and certification of their backend communication systems has become a simple job. This system, quickly built out of a number of off the shelf components, provides a quick way to validate communications, and can be reused for other protocols in the near future.

6. Acknowledgements

The authors want to thank all people involved in the project. Next to the customer and their very helpful staff, Richard Jaspers of Competa IT and Georg Tornqvist of OELAN are thanked for their work in implementing the software needed. Thanks go to Dirk Giesen of Parasoft Netherlands for his guidance and expert advice.

[1] Ultra ESB: http://www.adroitlogic.org/
[2] Virtualize: http://www.parasoft.com/
[3] Mendelsson: http://as2.mendelson-e-c.com/

Progress of DNS Security Deployment in the Federal Government

Scott Rose
NIST

Abstract

In 2008, the US Federal government mandated that all Federal Executive Branch owned DNS zones must deploy DNSSEC. Initial deployments lagged and often error prone, and in response, the DNSSEC Tiger Team was formed to aid deployment and develop a system to monitoring system. The results showed a significant increase in deployment as well as a reduction in errors. When errors were detected, the time it took to resolve the problem was also reduced.

This paper discusses the history of DNSSEC in the gov domain, the types of errors seen, and how they were reported. This paper concludes with a set of lessons learned that would apply to other large domains or groups wishing to make DNSSEC a requirement for operation in members' zones.

1. Introduction

The DNS Security Extensions (DNSSEC) is a collection of extensions to the DNS to provide source authentication and integrity protection. DNSSEC does this by adding digital signatures to DNS data, which clients can validate using public keys also stored in the DNS. The DNSSEC specification was initially published in 2005 by the IETF in RFC 4033[1], RFC 4034[2] and RFC 4035[3].

DNSSEC is deployed on a per-zone basis and uses the existing DNS hierarchy to establish chains of trust from parent zone to child zones. Parent zones vouch for the DNSSEC status of delegated child zones by using a special Delegation Signer (DS) Resource Record (RR). The presence of this RR gives the client information about the key used by the client. Lack of this RR means that the child zone is either not signed, or in the process of deployment and the child zone administrator does not consider their deployment to be production ready. Often the last step in deployment of DNSSEC in a given zone is to upload the key information for the zone to its parent zone. This is done out of band of the DNS, usually through a registrar web portal or similar. For example, the registrar website for the gov Top-Level Domain (TLD) is https://www.dotgov.gov/ and allows registered administrators to upload and request key data to be published for their delegation (e.g. dnsops.gov).

This linking of security from parent to child makes the deployment of DNSSEC at the DNS root and TLD's important, as clients with the DNS root public key and/or TLD public keys would be able to validate the widest set of possible DNS responses. Child zones under TLD's can sign their zones and simply upload their key material to their parent zone without having to go through the effort of publishing their public keys for all clients to obtain.

2. Background and Deployment Drivers

Though the current specification was published in 2005, initial deployment of DNSSEC was scant, with few zones being signed. DNSSEC got a boost in interest with the disclosure and publication of the so-called Kaminksy attack, presented at Def-Con in August 2008[4]. Also that month, the US Office of Management and Budget (OMB) issued OMB-08-23 "Securing the Federal Government's Domain Name System Infrastructure" [5] which set deadlines for the deployment of DNSSEC by Federal agencies. This was often referred to as the "OMB DNSSEC mandate". The deadlines given in the memo were for the gov TLD used by the Federal government to deploy DNSSEC by January 2009 and every Federally owned second level zone be signed by December 2009. Federal agencies only make up roughly 20-25% of the gov TLD, the remaining delegations belonging to state and local governments, and American Indian tribes (Native Sovereign Nations) neither of which falls under the mandate.

Parallel to the OMB mandate was the addition of DNSSEC to the Federal Information Security Management Act (FISMA) controls. FISMA requires each Federal agency, and private entities that posses and process Federal information to have an established IT security policy for each system, and includes a set of checklist items (called controls) that are recommend or required for Federal systems, depending on the risk factor. Deployment of

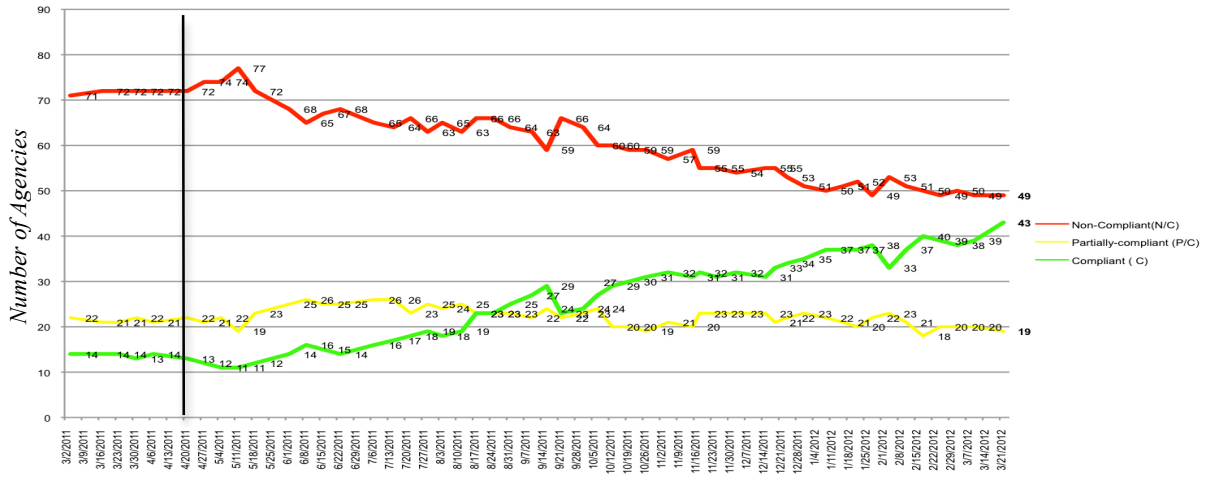


Figure 1 DNSSEC Deployment by agency in the gov TLD per week (taken from a sample DHS compliance report)

DNSSEC was added to the controls for Federal systems to deploy, which applied to all Federal zones, not just second level zones as mandated in OMB-08-23

3. USG DNSSEC Tiger Team

Even with the OMB DNSSEC mandate and FISMA drivers, deployment of DNSSEC by Federal agencies was slow. The gov TLD was DNSSEC signed by February 2009, missing the deadline by 1 month, but only a fraction of agencies signing their zones within the agency deadline [6]. This is partially due to a lack of coordination within the Federal space, as agencies have traditionally been very independent in establishing and managing their DNS. The operation of DNS by Federal agencies varies from internally operated by Federal employees (or contractors) to outsourcing to commercial hosting services, so no "one-size-fits-all" approach could be applied across all agencies.

In response to the slow rate of adoption, the Federal CIO Council chartered the DNSSEC Tiger Team to address the issue. The Tiger Team initially met in April of 2011 and met monthly to coordinate efforts to discuss deployment barriers and address concerns. The Tiger Team consisted of volunteer participants from various agencies and members of DHS Federal Network Security (FNS) tasked with monitoring deployment of DNSSEC in the gov TLD. NIST participated in the Tiger Team first as subject matter experts, and later as co-chair of the Tiger Team.

The Tiger Team helped increase the number of signed zones through a program of training, communication and monitoring. The team collected a set of training material for distribution to all Federal DNS administrators, as well as funding a computer based training course made free for all government employees. An internal, government only email discussion list was created for administrators to discuss issues, roadblocks and tips. Finally, a DHS program to regularly monitor DNSSEC deployment was established that would send weekly reports to administrators (via the email discussion list) and monthly reports to agency CIO's.

As seen in Figure 1, the Tiger Team has had a measureable impact on the number of signed zones. In the figure, the black vertical line in April indicates when the Tiger Team first convened. A few weeks after the Tiger Team formed, the DNSSEC compliance monitoring and reporting program started at DHS. From then, the number of DNSSEC compliant agencies increased. On March 26th, 2012 (the latest date in Figure 1), 910 unique Federal zones were signed and chained from the gov TLD, or 54% of all Federal zones.

One other interesting thing to note is that the total number of Federal zones in the gov TLD have also decreased during this period. There are two factors to this trend: First, OMB memo 11-24 [7] issued a call for agencies to reduce the number of Federal domain names and websites in order to reduce possible citizen confusion. Secondly, the DNSSEC requirement gave many agencies a chance to re-visit its DNS infrastructure and offered an excuse to take inventory and remove zones that were no longer needed or desired.

4. Types of Errors Seen, and Efforts to Remediate Them

DNSSEC involves a new set of operations to traditional DNS operations. Rushing to deploy can result in errors, which can be worse than not deploying. NIST performed daily scans of the known Federal gov name space and recorded any seen errors that would cause a client to reject DNS responses from a zone. These errors can be broken down into five basic categories:

- **NoSigs**: The parent zone claims the zone is signed (i.e. a DS RR is present), but the zone does not have DNSSEC signatures over the zone data. Clients would expect signatures, and failure to obtain them in responses results in an error.

- **ExpiredSigs**: The parent zone claims the zone is signed, and the zone has signatures over its data, but the signatures have expired. The client would reject these responses as invalid.

- **SigsPriortoInception**: The parent zone claims the zone is signed, but the signatures in the zone have a creation date set in the future. This means that a validating client would reject these signatures, as they are not valid. This error is likely due to a clock error on the system used to generate signatures.

- **BadKeyRollover**: The zone has recently changed its keyset, but the parent zone was not informed and continues to publish the old DS RR. Clients see a key mismatch between a trusted source (the parent), and the zone, and a validation error results.

- **DSPointstoPre-publishedKey**: The zone is in the process of changing its keyset, but the parent zone has a DS RR for the (possibly) new key instead of the current key. This error is rarely seen, and was largely due to an appliance implementation error that has since been patched, but still seen in production services.

Other errors were seen during this period, but they were not directly DNSSEC related, so they were not included. These errors were typically network related issues, or system problems that rendered the entire zone unreachable for all clients, not just for those performing validation.

Not having a DS RR in the parent zone is not considered an error, as this will not result in the response being rejected by a validating client. These zones are often called "islands" since they are signed, but often can't be validated unless clients have some means of obtaining the zones' keys in a trusted manner.

Figure 2 below shows the DNSSEC errors seen per day during the last five months as Figure 1. The errors are color coded to show each category.

From the above figure, it is clear that while the number of errors changes over time, the two categories that make up the majority of the errors are ExpiredSigs and BadKeyRollover. That is likely due to the fact that these are often manual operations done by administrators or (in the case of BadKeyRollover) require a human to perform at least part of the operation (i.e. interact with a registrar).

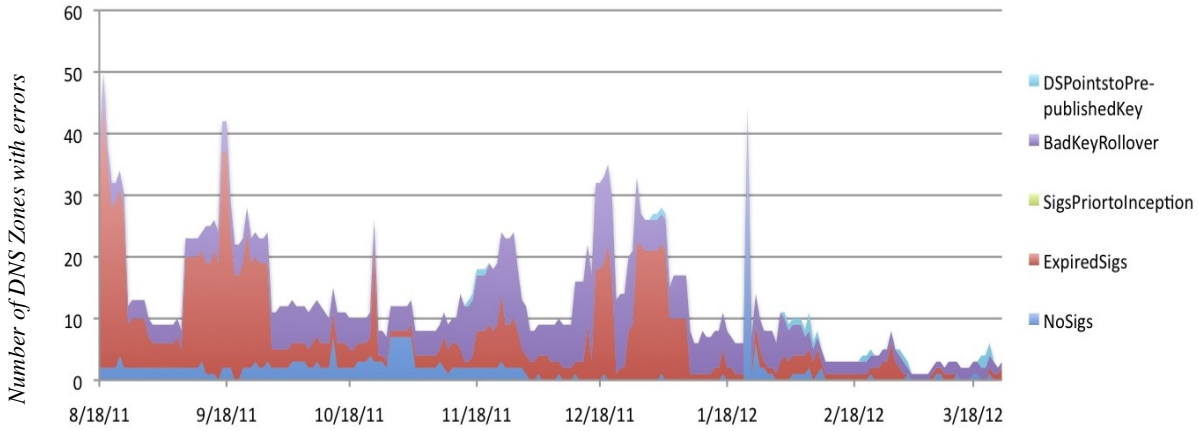


Figure 2: DNSSEC errors seen per day in gov domains

Further evidence that these errors are due to the dependence of manual operations is the spike seen during the holiday months (i.e. Aug/Sept and Nov/Dec). During these months, DNS administrators are often on leave, or dedicating their time to end-of-year tasks and DNSSEC maintenance operations (like DNSSEC re-signing) are overlooked. The number of errors seen drops post December, indicating that administrators are back at work and resolving the issues seen over the holidays.

It is also interesting to note that while the overall number of signed zones have increased (Figure 1), the number of errors seen have stayed the same or decreased. The monitoring and reporting system in place is often credited with this trend. As both IT management and administrators are informed as to the state of their DNS, errors are being caught sooner and problems resolved quicker.

5. How Monitoring Helps

It has been shown that the DHS monitoring program has increased the number of signed zones in the Federal DNS name space, but also has helped shorten the time between the initial failure and remediation of the problem. Analyzing the number and types of errors in the April of 2011 (after the Tiger Team was formed but before the monitoring program started), shows the following breakdown of errors as seen in Table 1.

Error seen	Num. Errs	Min. Days	Max Days	Avg. Days
NoSigs	41	1	20	2
ExpiredSigs	21	1	27	6
SigsPriorto-Inception	1	9	9	9
BadKeyRollover	3	1	27	14
DSPointstoPre-publishedKey	6	1	27	9

Table 1: An analysis of errors seen in March 2011

The large number of errors and the wide range of days until the errors are addressed reflect the wide range of quality in DNSSEC operations in the Federal name space. Interestingly, the majority of the errors are ExpiredSigs and NoSigs. It is believed, based on some anecdotal reports that the NoSig errors are due to failures in configuring name servers to properly serve DNSSEC responses. BIND (the most widely used software for authoritative DNS servers) requires manual changes to its configuration file before loading and sending DNSSEC replies. DNSSEC processing is turned off by default.

The length of time required to resolve some of these errors seems very long. In the most extreme cases, problems were not addressed for weeks. This is likely attributed to the lack of outside clients asking or checking for DNSSEC signed responses. It is known that very few (if any) clients were performing DNSSEC validation in March of 2011. Even today, only one major US ISP (Comcast) provides DNSSEC validation for all customer DNS queries. There is also the problem of how to report issues when DNSSEC validation fails. Unlike problems with web pages or other services, there is no common standard "dns@example.com" type email addresses to report problems. The RNAME field in the SOA RR is cited as the place for this address (RFC 1035 [8]), but it is often not used correctly or redirects to a mailbox that is rarely (if ever) checked. Often, problems are resolved by asking on email distribution lists for a point of contact, which often relies on finding the right audience or searching for a help desk at the zone's registrar or hosting service.

Performing the same analysis in April of 2012 shows a marked improvement; not just in the number of errors, but also in the time it took for the errors to be resolved. The improvement can be seen in Table 2.

Error seen	Num. Errs	Min. Days	Max Days	Avg. Days
NoSigs	6	1	1	1
ExpiredSigs	4	1	4	2
SigsPriorto-Inception	0	0	0	0
BadKeyRollover	2	3	7	12
DSPointstoPre-publishedKey	3	3	3	3

Table 2: An analysis of error seen in March 2012

This improvement can be attributed to the monitoring program that not only regularly checks the DNSSEC validity of zones, but also reports directly to agency administrators and managers the status of their zones.

This feedback is seen as critical to the success of deployments. The second factor is the increase in knowledge sharing between DNS administrators that have resulted in improved operations in agencies. More agencies are automating regular DNSSEC operations such as resigning and portions of the key rollover process. This is done using a range of options from simple scripts and open source tools to purchasing automated appliances or outsourcing of operations to contracted parties.

The current situation is not immune to problems, however. The most famous example is the incident in January of 2012 when Comcast customers could not reach nasa.gov servers because the nasa.gov zone incorrectly performed a key rollover. The after action report from the Comcast perspective was published in their blog [9] which contains a detailed description of the problem and gives details about how they react to DNSSEC validation failures in order to maintain service for their customers.

6. Lessons Learned

The experience with the Tiger Team in deploying DNSSEC across the Federal Executive branch highlight several key lessons to consider for large organizations and communities when they seek to deploy[10]. Perhaps the most important lesson doesn't directly involve DNS at all; that is, knowing whom in another organization is responsible for DNS (and by extension, DNSSEC), and who to contact when something goes wrong. The first issue the Tiger Team had was identifying responsible points of contact (either administrators or network managers) in each agency. This is especially difficult in agencies that outsourced their network operations.

This becomes more important if an error is detected. By policy, the gov zone does not have a thick WHOIS, and does not list email addresses of points of contact. There is a registrar help desk that is publically accessible (at http://www.dot-gov.gov/), but few Internet users and network managers outside of the government know of its existence. Outside of the government, a delegation's WHOIS information may be out of date, or not list the actual day-to-day operator of the problem zone. Domain name owners should take every step to insure that their WHOIS information is current as well as having a valid email address in the SOA RR that is monitored by operations staff.

However, often an end-user does not seek out points of contact to report errors, so zone administrators must be pro-active. The Comcast-NASA.gov incident illustrated the point that the majority of end users are not aware of how DNS or DNSSEC works, and will instead vent their frustrations on social media sites first, and their ISP's help desk second.

The second lesson is that it is easier to detect one's own problems than to react to them when learning from outside sources. Regular monitoring from an outside point of view can alert administrators to a potential problem before they become a larger issue. Some external sites provide a snapshot view of the DNSSEC status of a given zone, but it is trivial to set one up to monitor a zone's own DNSSEC status. This could be extended to warn of potential failures (i.e. signatures that may expire soon) as well.

The third lesson is it helps to have a forum for community members to hold candid discussions on roadblocks, challenges and ask questions about DNSSEC deployment. NIST created the "gov-dns" mailing list on behalf of the DNSSEC Tiger Team to be used by USG administrators or contractors directly supporting a zone in the gov TLD. The purpose for the restriction was to give administrators a forum to ask questions about Federal policy (e.g. key size and rollover frequency) or questions they may feel uneasy asking in a public DNS operations forum (i.e. questions about a specific DNS server implementation). Admins who noticed a particular issue with a .gov zone would also use the list to call attention to the issue if they could not reach a zone POC directly. While not perfect, it was one of the ways USG zone administrators were able to coordinate responses to DNSSEC errors. The forum was also served as the outlet for summary compliance reports from DHS to administrators to show progress of DNSEC deployment in the gov TLD.

Lastly, DNSSEC requires a different operational approach than traditional DNS, as there are more time dependent operations such as re-signing of zone data and routine key changes (depending on an organization's security policy). These operations can be easily automated and tools are available (both open source and COTS) to aid administrators. These tools, coupled with tasking backup administrators for coverage during holidays or leave periods will reduce the risk of the most frequent types of DNSSEC errors encountered (e.g. ExpiredSigs).

7. Current Status and Future Efforts

The USG DNSSEC Tiger Team is no longer meeting,

but the monitoring program set up by DHS FNS continues to scan Federal domains and issue reports. The rate of DNSSEC deployment continues to increase and the frequency of errors continues to remain low and stable at less than ten zones identified as having errors seen on a daily basis (1%-2% of all signed Federal zones).

DNSSEC is seen as an enabling technology, not just a means to protect traditional name to address translation. Secondary to DNSSEC deployment, the USG Tiger Team sought to increase the use of various email authentication techniques that rely on the DNS in some way, such as Sender Policy Framework (SPF) [11] by Federal agencies. These technologies publish email policy information in the DNS, which can be digitally signed by DNSSEC just like any other type of DNS information. It is still too early to conclusively say how a signed DNS infrastructure can be used to build trust in other applications and services.

8. Acknowledgements

The author would like to acknowledge key individuals who were vital to the increased deployment of DNSSEC in the gov domain. These individuals are Earl Crane, DHS and chair of the DNSSEC Tiger Team, Valeri Stoyanov of DHS FNS and the other members of the USG DNSSEC Tiger Team.

9. References

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, DNS Security Introduction and Requirements. RFC 4033, March 2005.

[2] R. Arends, et al. Resource Records for DNS Security Extensions. RFC 4034, March 2005

[3] R. Arends, et al. Protocol Modifications for the DNS Security Extensions. RFC 4035, March 2005.

[4] D. Kaminksy. BackOps 2008: It's the end of the Cache as we Know it. http://www.slideshare.net/dakami/dmk-bo2-k8

[5] Securing the Federal Government's Domain Name System Infrastructure. Office of Management and Budget Memoranda 08-23. http://www.whitehouse.gov/sites/default/files/omb/memoranda/fy2008/m08-23.pdf

[6] William Jackson. DNSSEC spreads slowly through government domains. *Government Computer News Magazine*. Sept 23 2010. http://gcn.com/articles/2010/09/23/governmemt-slow-to-deploy-dnssec.aspx

[7] Implementing Executive Order 13571 on Stream lining Service Delivery and Improving Customer Service. Office of Management and Budget Memoranda 11-24. http://www.whitehouse.gov/sites/default/files/omb/memoranda/2011/m11-24.pdf

[8] P. Mockapetris. DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION. RFC 1035, November 1987.

[9] Analysis of DNSSEC Validation Failure Comcast – DNS Engineering http://www.dnssec.comcast.net/DNSSEC_Validation_Failure_NASAGOV_20120118_FINAL.pdf

[10] CONSIDERATIONS AND LESSONS LEARNED FOR FEDERAL AGENCY IMPLEMENTATION OF DNS SECURITY EXTENSIONS AND E-MAIL AUTHENTICATION. Federal CIO Council Whitepaper. Nov. 2011 http://www.cio.gov/documents/DNSSEC_and_E-Mail_Authentication_Considerations_and_Lessons_Learned.pdf

[11]M. Wong. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408. April 2006.

Building the Network Infrastructure for the International Mathematics Olympiad

Rudi van Drunen, Competa IT, (R.van.Drunen@competa.com),
Karst Koymans, University of Amsterdam, (K.Koymans@uva.nl)
The Netherlands

Abstract

In this paper we describe the network infrastructure we designed, built and operated for the International Mathematics Olympiad in the Netherlands in 2011. The infrastructure was pragmatically designed around OpenVPN tunnels in a star topology between the various venues. VLANs were extensively used to separate functional groups and networks. The actual construction of the event network took about 3 days and was needed for only 2 weeks. The architectural, setup, building and operational aspects of the network are described and we include some lessons learned.

1. Introduction

The International Mathematics Olympiad (IMO) was to be held in the Netherlands during a week in the summer of 2011. During this week 600+ scholars plus about the same number of support staff were invited to the Netherlands to do 2 days of mathematics contests. In the weeks before the event. the assignments had to be created and selected by the team captains and jury at an undisclosed place. After the event, all work needed to be scanned, transmitted, corrected and the results and final classification needed to be ready within 2 days.

To support this organization a complex and temporary (as the Olympiad is held in a different country every year) IT infrastructure was needed. At all locations we needed to set up infrastructure in such a way that a complete internal private network for the Math Olympiad was created. We had 4 locations and a central datacenter to be connected for the different services. Owing to the security measures as a functional requirement, we had multiple different networks across the infrastructure. Further, we wanted one single point of access to the Internet for access control.

While all of the networking was needed for about 2 weeks, we only had about 3 days to build the entire setup at the various venues (hotels, sports accommodations) throughout the Netherlands. It was a major technical and logistics undertaking, all done with a group of volunteers. In this Practice and Experience Report we will describe the design and setup of the site interconnection networks.

2. Requirements

The functional requirements for this ad-hoc setup were quite simple: the Mathematics Olympiad needed networking facilities to support the contest in the form of internet access and access to the main logistics and administration system and its backup. Security was defined in terms of access control to different pieces of the data for functional groups of users. Additionally there were a number of “nice to have” features defined, such as video streaming and internet café style setups at various locations. A set of strictly defined technical specifications did not exist.

3. Architecture

We co-located the servers at a datacenter of the masters program of System and Network Engineering at the University of Amsterdam where multiple high speed internet feeds were readily available. In this datacenter we located all central services for the IMO.

The central services, such as the main system that supports all logistics and operations, were located here and were reachable over the Internet for certain functionality such as presentation of results. It was also required that these services be reachable from the internal IMO networks for preparing and selecting the assignments, translating assignments to each of the 60+ official languages, and other tasks. At the different sites, the networks were separated into functional VLANs, which were distributed throughout the different venues using VLAN capable switches.

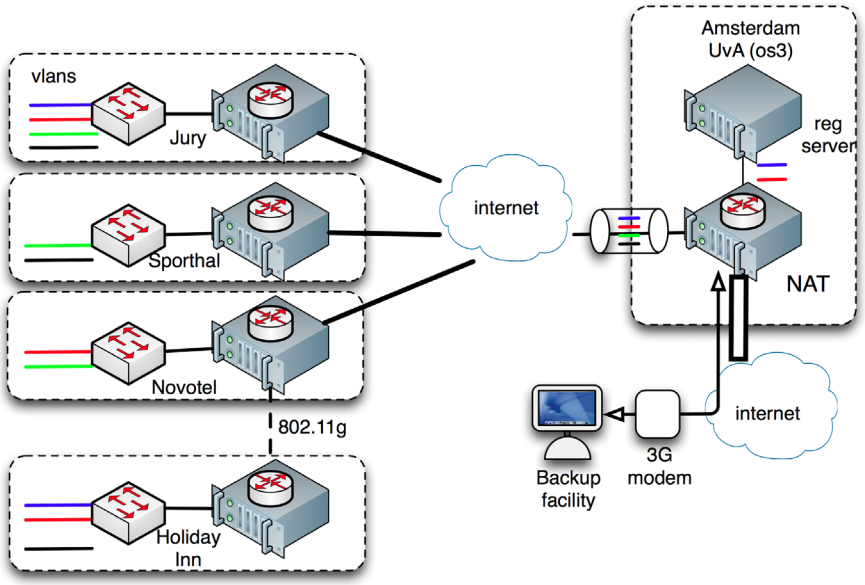


Figure 1: Schematic network overview

All sites were equipped with a VPN tunnel to the central datacenter. All tunnels were terminated at one point. At this termination point we are able to control routing between the different sites and networks. Also, the Internet connectivity was controlled at this point. A schematic overview of the setup is depicted in figure 1.

3.1 VPN setup

To establish the VPN tunnels between the different sites we used OpenVPN. OpenVPN is a SSL VPN solution that works on the application layer, in contrast to other solutions like IPSec where the VPN is created in the network layer of the stack. A specific and important advantage of OpenVPN is that no extra difficulties are caused by the use of network address translation (NAT). We had to cope with very different Internet feeds at the different venues we were using. While NAT traversal was available for IPSEC, the extra configuration complications favored avoiding it entirely. Having had previous bad experiences using IPsec from a system behind a NAT gateway we decided to use OpenVPN. Another advantage of OpenVPN was that it was easy use a non-standard port if needed, to circumvent local firewalling: for example 443 (https) instead of the standard 1194. Because of the fact that OpenVPN runs in user space, very high throughput (> 1Gbit/s) can cause context-switching problems, but the total (VPN) network bandwidth that we would need at any location (including the central site) would not have exceeded 600 Mbit/s in our rough estimations, taking into account the bandwidths we got at the different venues.

3.2 VLANs and IP Number plan

For the internal IMO infrastructure we designed an IP numbering scheme that used RFC 1918 private addresses. We put different functional groups of users in different VLANs: contestants, jury, officials etc. The IP ranges on these VLANs corresponded in one of the octets to the VLAN number to ease identification and troubleshooting, for example VLAN 15 corresponds to subnet 10.15.0.0/12. It proved to be very helpful to identify the network (VLAN) from the IP number issued from DHCP.

3.3 Distribution

All sites had a VPN terminator machine that both served as router and provided for basic network services for the IP ranges on that particular site. The VPN terminators provided network services such as DHCP and DNS (relay) to the local networks. This prevented broadcasts (DHCP) over the VPN and created a fallback scenario; when the VPN tunnel to the central site had problems, Internet connectivity to certain VLANs could be restored easily by using the local Internet connection. The internal network interface on the VPN terminator machine, equipped with a number of virtual interfaces, one for each VLAN, carried a trunk that brought all VLANs to a switching fabric. This setup of switches separated the different VLANs to the users.

4. Implementation

We aimed to implement all network infrastructures as transparently as possible, using mostly Open Source solutions. Since the number of devices in the network infrastructure was rather limited, and the time to build and operate was really short, the decision was made to do a “one-off” and pragmatic design. We refrained from setting up configuration management and a deployment environment, and we defined network monitoring as a nice to have, but not essential to the operation. For example the VPN terminators were “hand configured” in a standard way documented on the ICT project wiki.

4.1 Background

The VPN implementation was done on FreeBSD 8.2. We chose FreeBSD because of a good network stack, easy deployment and a proven OpenVPN port. Each site was equipped with a FreeBSD machine running OpenVPN. We had a hot-spare machine on the central site to be able to cope with disaster during the event. All machines had 2 network interfaces: one connecting to the Internet feed on the location and one on the internal networks (switch fabrics that were installed).

4.2 Central Site

The central site machine was running 4 instances of the OpenVPN daemon, each running on a different port at the external interface. We needed multiple daemons running at once since the OpenVPN daemon was not multithreaded, and to make efficient use of the multicore architecture of the central site. This resulted in a clear configuration; each daemon maintained a distinct connection with a separate configuration file and virtual endpoint interfaces on the machine. This simplified the routing setup and debugging. All VPN tunnels were using self-signed certificates with a private CA for authentication and encryption, installed manually. The bandwidth needs were not so large that we needed to take special measures here. On the central site we had access to 1 Gbit/s full duplex, where on the other sites bandwidths varied between 40 Mbit/s and 100 Mbit/s.

The central machine also provided the Internet connectivity. To connect to public external addresses from the internal provided network ranges we used NAT on the central node. Since we had an estimated 400 international people concurrently using the network, many of whom might be using tools such as Skype, or browsing the Internet, we needed to cope with a large number of outgoing sessions. This discouraged the use of user-space `natd`(8), so we used `ipf` and `ipnat`(5) providing an in-kernel solution that was able to handle sessions more efficiently. This also provided the ability to add multiple external IP addresses to be used as source IPs for the NAT sessions so that we would not run out of source port numbers.

4.3 Remote sites

All remote sites had the same basic FreeBSD setup, where just the OpenVPN configuration files, certificates, DHCP, DNS and routing configurations differed. Also the network setup was kept as simple as possible to prevent errors. When we started the VPN, the endpoint machines were not reachable over the Internet anymore, but only through the VPN tunnel from within the IMO network cloud. So, to prevent lockouts, an extra firewall / routing entry was added on each remote site to be reachable over the Internet without using the VPN tunnel. On the remote sites the internal interface of the VPN terminator carried a trunk with a number of VLANs that were split using switches.

4.4 Switches

To have full control over the network environment, the switches that we deployed were all managed by the IMO IT team. To prevent issues we decided not use the existing switches that the venues had in place. This reduced the number of potential building and operational problems for preexisting, disparate switches enormously, trading them for procurement, configuration, and maintenance of new switches. This reduced the number of potential problems (building and operational) enormously, but faced us with extra tasks to have switches available, configure them and maintain them. The only infrastructure we used that was provided by the venues was the Internet uplink and the dry copper (and dark fiber) to the rooms that they had installed. Use of the existing infrastructure, even just the cabling, proved to be challenging because of the way it was installed (see Figure 2).

Most end user equipment was connected to wired Ethernet. At some locations (local) WiFi was used to extend the wired infrastructure. In order to not run into extra complexity, we deployed simple (unmanaged) access points connected to the wired network when needed. We thought of reflashing the APs with an Open Source distribution, but as we used them simply unmanaged it did not add any value, so we used them with the manufacturer`s firmware

5. Deployment

As a team of about 6 people partly in their spare time, installing the complete infrastructure was a major logistics operation. Key here was that we had ample space in a datacenter of the University of Amsterdam which allowed us to have all VPN terminator machines (central and all remote sites) set up in a lab environment before actually shipping them to the sites.

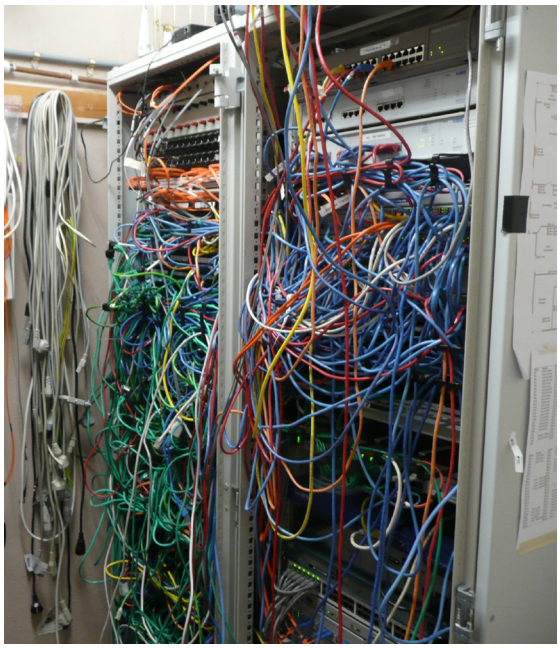


Figure 2: Existing Cabling

Figure 3: Testing in the data center

On an afternoon, about a week before the event, we built the complete server and switching environment in the datacenter to test all configurations and routing. This included a wireless link we were to use between two buildings.. The relatively small setup (5 locations) allowed us to do this just before shipment of the equipment to the remote locations. This exercise proved to be a major advantage, because during these tests a number of configuration issues showed up such as mistakes in IP numbers and configuration errors in the wireless bridge Figure 3 shows one of the authors during the test of the complete setup with all VPN endpoints and switches.

A number of site-surveys on the different locations were held well in advance in order to deploy the network on various sites efficiently. Recognizing the local situation, patch layouts and uplink possibilities proved to be very helpful in the deployment. The site survey revealed that using a wireless bridge between two sites was a better solution than adding another VPN site. Also explaining the design, the setup and the work we needed to do on-site to the local tech staff and contractors created a good working relation during the event.

Another important point in building an event-network setup is that labeling and having / keeping inventory lists (and up-to-date DNS) are critical. The necessity of fast, ad-hoc setup for a short term goal lends to the engineers' ability to improvise and adapt in a fluid environment. But without proper labeling (see Figure 2) and documentation, debugging would have been a serious problem and could have had negative repercussions for the IMO event.

6. Operation

The complete infrastructure was set up in the different venues over 4 days. There were quite a few network layout and connectivity problems that were unforeseen, including physical layer Internet feed issues at one location; these problems required some improvisation and quick thinking.

Other issues that we faced were partly on the political level, and being able to cope with the improvisation talents of other parts of the organization, as rooms and network layouts were changing unexpectedly and without advance notice. As we had configured a number of extra ports in all VLANs on the different switches, and have them clearly labeled as such, it was not a big problem to cope with these ad-hoc changes.

Next to good documentation and a solid communication infrastructure (a wiki and cell phones with all numbers of the organization team at large preprogrammed) good interpersonal communication skills and true dedication within the team were absolutely critical, here. Simple tools like network analysis software (such as wireshark), cable testers and tracers have proven to be very useful in the appropriate hands. Also a labeling machine has proven to be very useful, as can be imagined.

7. Backup

Fortunately, all the measures that we took to prevent and overcome network problems were not needed. We did have 3G cellular modems at two important sites that would have been able to serve as a backup link if necessary. As already mentioned, the central site had a standby machine to be used as VPN terminator, multiple Internet feeds, etc. In case of an emergency on any of the remote sites this warm-standby machine could be converted into any of the VPN endpoint machines on the fly by manual switching configuration directories.

For the main logistics and operational system we had a backup in another data center, reachable over the Internet and a live copy of all the operational data on a laptop.

8. Conclusion

A good and small team, with defined skills, clearly expressed responsibilities and a vast amount of improvisational talent will get you an interesting and high performance multi-site network for an event. However, do not underestimate the amount of time needed for preparation and design. We started this project more than 1 year before the actual deadline of the event. Although the requirements for the network were not strictly defined, a common sense engineering approach and a number of definition and engineering meetings resulted in a very workable, not overcomplicated setup.

The pragmatic approach worked here. But, if the event had been larger or longer running, appropriate best-practices, such as the use of a configuration management system and extensive monitoring systems, would have had to be evaluated.

9. Acknowledgements

The authors want to gratefully thank the masters program for System and Network Engineering at the University of Amsterdam, the rest of the IT team and organizers of the IMO and all sponsors for their help, support and good company. Thanks go also to the authors' employers (Competa IT B.V. and the University of Amsterdam) that supported this work by allowing the time to do this project.

Lessons Learned When Building a Greenfield High Performance Computing Ecosystem

Andrew R. Keen
Michigan State University
Dr. William F. Punch
Michigan State University
Greg Mason
Michigan State University

ABSTRACT
Faced with a fragmented research computing environment and growing needs for high performance computing resources, Michigan State University established the High Performance Computing Center in 2005 to serve as a central high performance computing resource for MSU's research community. Like greenfield industrial development, the center was unconstrained by existing infrastructure. The lessons learned are useful when building or maintaining an effective HPC resource and may provide insight for developing other computational services.

TAGS
HPC, infrastructure, customer support, imaging & automating builds, configuration management, security

1. UNDERSTANDING THE ECOSYSTEM

The primary goal of a capacity HPC resource is to meet the resource's users' computational needs, rather than targeting maximum theoretical performance. The whole HPC ecosystem must be considered, including facilities, the supporting computational services, the human resources necessary to deliver these resources, and the policies to ensure optimum system utilization. However, when choosing HPC resources many administrators neglect these requirements in pursuit of maximizing theoretical performance. We learned this lesson early in the center's life.

The center began operations in 2005. The first system implemented was a 64 processor, large memory SMP system. The system performed very well on the original benchmark suite as defined in the RFP, and the fast floating-point performance, large per-processor cache, low-latency inter-processor interconnect, and significant memory bandwidth of the SMP system ensured that it performed well on those benchmarks. However, after the system was put into production, the system's performance proved unacceptable when multiple I/O-intensive jobs were ran. It was determined that the performance of the supplied locally attached disk system was inadequate.

- While the attached disk subsystem performed well enough to support a single workload, the low rotational speed (and low IOPS) of the attached disks degraded performance significantly worse than linearly as multiple requests were sent to the array.
- The job management policy as implemented by the job scheduler was designed to maximize processor utilization instead of maximizing peak efficiency. A more intelligent scheduling policy could have limited the number of disk-intensive jobs while leaving multiple processors unused.
- The benchmark cases, while reflective of the individual workload components of the center, did not reflect the center's day-to-day workload.

After profiling the problematic applications at a system (iostat, libffio [1], Performance CoPilot [2]) and storage level (on-controller operation, bandwidth, and latency statistics) the vendor and the center determined that a faster storage subsystem was required to allow the server to run multiple disk-intensive jobs. Implementing the storage added about twenty percent to the original purchase price of the system but it resolved the I/O bottleneck and reduced time processes spent in iowait. The center was able to reuse the original storage in a role it was better suited for.

2. CLUSTER ADMINISTRATION

When managing a HPC resource, we have found it to be important to define our functional requirements and the tools we use to address them. Issues involving hardware and software management, data storage, environmental constraints, availability, and security are some of the issues we have had to address; cluster and workload management tools have helped us address these challenges.

2.1 CLUSTER MANAGEMENT

A modern HPC resource requires many software components, including: authentication and authorization, data storage, data transfer, network management, hardware management, job management, operating system and configuration management. There are many software packages available to HPC administrators to accomplish each of these tasks; however, many have potential pitfalls in their default configuration that are nonobvious to the inexperienced administrator. First-time HPC service system administrators should strongly consider using integrated open source (e.g., Warewulf [3], ROCKS [4][5]) or commercial (e.g., Bright [6], Platform HPC [7]) cluster management solutions [8] as a way to avoid common mistakes and as a way to familiarize staff with the interactions between software subsystems. We have been through three cluster management environments and have

found that there are a number of things to consider when selecting cluster management software.

- How easily does it integrate with other technologies? One integrated cluster management product we used did not natively support using an external LDAP server for client account information. We had to build a RPM to manually distribute the basic settings that were not supported by the management software.
- Does the product lock you into a specific vendor? An extensible product that uses standards like IPMI and standard software distributions is preferable. We have had issues with both commercial and open source products not supporting newer or exotic hardware.
- Consider firmware management options, particularly when purchasing new clusters. Important features include the ability to automatically update firmware and the ability to set commonly used configuration options. We have used Dell's OpenManage Deployment Toolkit [9] and setupbios from the Dell PowerEdge C System Management tools [10] to manage these environments. In HPC environments, BIOS configuration options ([11],[12]) can have significant performance impacts.
- Do the configuration management options provided match your policies? Does the software reinstall a system to update installed software or configuration? Will that interfere with scheduling or your SLAs' availability requirements? We've found that keeping the software environment on all compute nodes identical prevents problems with incompatible software versions. However, the problem is to find the appropriate balance between the improved availability provided by deploying changes while the system is running the job (rather than making changes offline) versus the possible problems caused by a temporarily non-homogenous configuration? How sensitive are your workloads to the CPU utilization of the configuration management tool? Tightly coupled MPI applications can be particularly sensitive to small disruptions.
- Is your workload certified or well suited for a given cluster management system? Some applications have advanced integration with certain workload management software ([13],[14].)

We originally started out managing a single SMP system by hand. When we purchased our first cluster, the vendor implemented node imaging with SystemImager [15]. However, we struggled with maintaining the proper workflow of making a change to a node, updating the image on the server, and pushing it out. Instead, changed files were frequently distributed using pdcp and commands were used with pdsh [16]. Changes made would be lost when systems were reinstalled; leading to regressions. To address this, we've separated the node installation from the node configuration. We have a very simple Kickstart [17] installation to bootstrap into puppet [18][19], where all of our configuration changes are stored. We also store the puppet configuration manifests in git [20] and have integrated branching with puppet's dynamic environments [21], which has simplified testing and implementing changes.

Your cluster management environment, whether self-maintained or part of a package, should include robust monitoring and management infrastructure. We use Cacti [22] for environmental and hardware-level device monitoring, Nagios [23] for service monitoring and alerting, and Ganglia [24][25] for performance information on the compute nodes. Ganglia provides per-node OS performance metrics, and Cacti is focused on out-of-band metrics and polls less aggressively. We have also developed automated regression testing that runs nightly and after every maintenance event to ensure that the systems are healthy; and implemented lightweight health checks [26] that run periodically and before and after every job. Nodes flagged offline by automated checks currently require administrator attention, but we are working with Moab's event-based triggers [27] to automate a manual testing workflow for hardware and software failures.

An isolated environment for testing infrastructure changes that is as close to the real configuration as possible is desirable. If you have automatic provisioning and configuration software (like we have with xCAT [28] and Puppet) you can install a test environment from your existing configuration, or clone system infrastructure virtual machines.

2.2 Workload Management

A HPC resource's workload can be any mixture of interactive and batch processing work. Understanding both the technical and political requirements and choosing software and policy that best reflects your users' needs is important. Poor choices can cripple the effectiveness of the resource and frustrate users. We have had four major iterations of our scheduling policy as we've learned and adapted to our workload.

2.2.1 Workload Management Software

While UNIX and most of its descendants have supported multi-user time-sharing since 1974 [29], the traditional POSIX tools for managing user workloads break down at multi-node scales and on modern hardware. Therefore, most HPC sites use workload management software to address these problems. The simplest method is to run workload daemons provided by software vendors (like Mathworks' MATLAB [30] and Wolfram Alpha's Mathematica [31]) that integrate directly into the software's normal interface. While convenient, they are generally only

well suited for small clusters or clusters that are dedicated to a single application. Anecdotally, most HPC resources use multi-node batch queuing software for workload management; we've used both PBSPro and TORQUE. We've chosen to use OpenPBS-derived systems to minimize disruption when transitioning between systems, and some of our application software integrates well with TORQUE and Moab.

2.2.2 Job Management versus Job Scheduling

Most resource management packages consist of two major components: a job manager that starts, monitors, and stops the workloads on the nodes in the cluster, and a job scheduler that directs the actions of the job manager. While TORQUE includes a scheduler, its functionality is very limited. We use a Moab Workload Manager [32], a commercial product from Adaptive Computing that allows us to set policies that provide better service to our users.

2.2.3 Job Scheduling Policies

In our experience, determining appropriate scheduling policy can be seen as balancing competing goals of responsiveness and utilization. To generalize, users want their work to complete as soon as possible (often expressed as a desire to have their job start sooner), while management wants high utilization to maximize return on their investment. We've implemented a number of limits (wall-clock, total jobs in use, the total number of CPUs used) to ensure fair and reliable access. These limits impact both user experience and system functionality. Limits should be high enough to meet user needs while maintaining fair access. Longer wall-clock limits limit the effectiveness of fair-share, make it harder to schedule other jobs and maintenance, and can increase the amount of work at risk from power or equipment failures. We currently allow up to a week of wall-time per job, but would like to reduce this. We've encouraged users to reduce their wall-clock time by allowing users who run less than four hours to run on idle nodes that other users have bought priority access to, by implementing trial support for system-level checkpointing with BLCR [33][34], and by educating users about ways to checkpoint and auto-resubmit their jobs [35].

We needed to choose the minimum resource increment that users could request, whether per-core, per-node, or the entire system. Our impression is that larger systems are generally scheduled in per-node or larger increments, and per-core was seen more frequently on small and medium-sized resources. We've chosen to schedule the majority of our cluster on a per-core basis, choosing throughput over peak single job performance. We can combine a 1 CPU, 20 GB job and seven 1 CPU, 256 MB jobs on a single 8 CPU node with 24 GB of RAM, or two jobs with 1 CPU and 1 GPU each, and 6 CPU-only jobs on a node with two GPUs and eight CPU cores. While there can be a performance hit when sharing nodes among workloads, we haven't found it to be problematic for most of our workload, as long as the cores themselves are not shared. Memory or I/O bound jobs, larger jobs, and benchmarking runs can still request whole nodes. Some sites use Moab's generic consumable resources to manage non-CPU resources as reserved resources to prevent contention [36].

We've found that setting short default wall-clock limits (1 minute) encourages users to set an accurate request, which makes scheduling more predictable. We're implementing further TORQUE submit filters [37] to give users immediate feedback about potential problems with their jobs.

We have learned to avoid linking queues to resource selection if at all possible. It increases the barrier for new users learning to use the systems, requiring them to determine what queue is best suited for a particular job. This proved to be frustrating to our users and non-transferable to other sites. We originally had queues and dedicated limits for given job categories (high CPU count jobs, long running jobs.) Utilization suffered, and users were unhappy because there were idle cores that were not available for their workload. In addition, queues are usually assigned at submit-time; if jobs are linked to specific hardware by the queue the system is unable to take advantage when other hardware becomes available. While time and resource specifications are largely similar on most HPC resources, queue names and their policies are unique to each site. We use a combination of node features, job attributes, credentials, node utilization, and reservations to place jobs on appropriate hardware. By standardizing our hardware and environment we've maximized the number of nodes a given job can run on.

2.3 Understanding availability requirements

When building your system, consider the impact of a single component's failure. A single node crashing is not disruptive to the center's mission if your workload can be restarted from a checkpoint file or resubmitted, but if one of the critical services goes down you can lose the entire cluster's workload. Building a robust core infrastructure is important if the total cost of a downtime including lost productivity is considered. When building a HPC resource, you should consider building infrastructure at a higher reliability level than the general compute cluster.

One of our clusters was designed with two network fabrics, a high-speed Infiniband network and a gigabit Ethernet network. However, there was a significant amount of traffic on the gigabit Ethernet network that regularly caused the entry-level gigabit switches it was connected to to crash. By replacing these switches with higher quality switches we were able to prevent these interruptions in service. If we had better understood our user workloads we could have specified better Ethernet switching hardware in the bid instead of incurring the cost of replacing them later.

Initially, a single physical node served both as the user login server and the infrastructure server. This proved to be very extremely problematic, as a single user with a misconfigured application on the login node could disrupt important system-wide resources such as the LDAP server.

By separating user nodes and infrastructure systems, we provided a clear divide between the service environment and user environment. As the center has added additional systems (now numbering in the thousands of general purpose cores as well as specialized hardware) our infrastructure has grown as well. Where appropriate, we added redundancy to services. Later, we migrated from physical machines to virtual machines for all non-storage infrastructure systems, minimizing the impact of hardware failure. Using that framework we are now deploying user-requested VMs from templates for applications with special needs, such as web services.

2.4 Storage

Data management is an important component of a HPC environment. There are at least three categories of storage that you may need to consider in a HPC environment: home directories, temporary storage, and archives.

2.4.1 Temporary Storage

Initially, the only scratch space provided was directly attached to the large memory systems, as described previously. Beginning with our first cluster in 2005 we have deployed three generations of the Lustre [38][39] file system.

There are a number of challenges when dealing with parallel I/O. Our first Lustre installation was too heavily biased towards reliability; the lack of capacity significantly impeded users' ability to do large research. Our second implementation concentrated too heavily on capacity and had no server-level redundancy; a single server failure would cause disruptions in scheduling. That is, jobs with files on the failed server that would normally complete within the time originally requested would be delayed long enough to push them past the scheduled end-date, at which point it would be killed by the job scheduler. We designed our most recent purchase to be well balanced between performance, capacity, and reliability. The Lustre servers are once again highly available, and no single point of failure exists in the storage stack. As scratch is designed to be temporary storage, we automatically purge files older than 45 days from scratch. This too has proven problematic and requires gentle "reminders" as files age on scratch. We are greatly cognizant that permanently removing files is potentially disastrous to the user so we often rotate files to offline storage before deletion, where the length of time they remain there depends largely on the available storage.

2.4.2 Persistent Storage

For permanent home directory storage, the center provides hundreds of terabytes to internal clients via NFS and campus and VPN clients via CIFS. We added ZFS-based [40] storage in 2009 as a replacement for a 15 TB Linux server that used XFS [41] and LVM. The cost, performance, ease of administration, and data integrity features were compelling reasons to choose ZFS.

We use 7,200-RPM SATA or near-line SAS hard drives for bulk storage, and use SSDs for read (L2ARC) and write (ZFS Intent Log) caches. While we have found that a write cache can significantly improve NFS performance (an order of magnitude in improved performance on one user benchmark), we have found that the separate read cache device is unnecessary for our workload (by looking at read and write statistics from the Illumos [42] kernel), so we will not use them in future systems. In our environment, separate read-cache devices would only be helpful with per-filesystem tuning. For us, it is much more efficient to simply add more DRAM to a system to leverage the ZFS ARC and kernel page cache.

With ZFS, creating new snapshots is non-disruptive. We create hourly and daily snapshots locally, which are directly available to the user. Nightly backups are replicated from the snapshots offsite to an external server. We've used the zfs-auto-snapshot SMF service in OpenSolaris and are migrating to a simplified version of this same process on new systems. An internally developed script handles offsite replication.

ZFS also provides robust on-disk data integrity protection. We have not lost a single bit of data to either silent data corruption or hardware failures.

The ability to do thin provisioning is also useful when creating new user accounts. Later updates to ZFS have added block-level deduplication [43] and compression [44]. Compression is quite useful for offsite backups. We've found it quite easy to achieve 2x or greater savings with the gzip compression found in later versions of OpenSolaris and Illumos with our users' datasets.

2.4.3 Archival Storage

We do not provide archival storage, but are working with university departments to meet the needs of the research community, including ways to provide components of future NSF grants' data management plan requirements. Currently, we work with users to migrate data to national data repositories.

2.5 Physical Concerns

HPC system density has increased well beyond the traditional datacenter standard of three to five kilowatts per rack [45]. The center has undergone three major renovations as density has increased. The first systems were low-density that could be cooled with a forced air raised floor. The second set of systems averaged about 12 KW per rack with two CPU sockets per rack U. Presently, we target a density of twenty five kilowatts per compute rack, using blades or blade-like systems, with two systems with two CPU sockets (a total of 4 sockets) per rack U. Our data center is space constrained, so density is important. It would not be possible to cool these dense systems with traditional forced air distributed by the 12" raised floor in our facility. While some new facilities (like NCSA's National Petascale Computing Facility [46]) are able to support systems of similar or greater densities with a 6' raised floor, that is not feasible in our facility. We instead use Liebert XD [47] high-density refrigerant-cooled in-row

spot cooling systems to supplement the existing under floor cooling. We are also considering higher density water-cooled rack systems in the future. We've seen significant improvements by implementing cold aisle containment. We were able to prototype closing the cold aisle by using cardboard and were able to observe a significant reduction in overall room and system inlet, component, and output temperatures. We then chose to implement a permanent installation based on this evidence.

It is important to coordinate new purchases with the people responsible for your facilities to understand what your constraints are and choose the cluster configuration that best meets your needs within those constraints. We cannot add any more power in our current facility. This has informed our hardware retirement policy; we retire hardware sooner to make room for new, denser, more efficient hardware.

2.6 Security

HPC resources can be more difficult to secure than standard Unix systems, given that most environments allow users compiler access and the ability to upload arbitrary binaries and scripts. Most HPC systems do not use common security features like firewalls and virus scanners on individual nodes due to the associated performance penalty. Choosing the right balance between usability and security is a decision that each site must address based on the risk of attack and the institutional, regulatory, and legislative requirements for data security. The full range of permissions has been fielded, from only allowing administrator-installed executables and validated input files to allowing users root access to systems.

Administrators can limit the utility of stolen credentials by limiting the sources that those credentials are valid from and by using implementing two-factor authentication. We have mitigated the potential risks by separating user-accessible systems from administrative systems, by isolating system control traffic from the user accessible research networks and by isolating most cluster systems from the public Internet. We are also implementing SELinux [48] on infrastructure systems where appropriate and use Fail2Ban [49] to limit brute force attacks.

It is useful to think about the trust relationships between systems. Some cluster management software's default configuration allows any root user on a system that it manages to access any other system in the cluster as root, including administrative systems. This can effectively compromise the entire cluster if a single machine is compromised. In general, systems that are trusted should minimize the number of systems that they trust. Remote root access, when needed, should be restricted to connections from hardened hosts and should rely on agent-based public key authentication from external systems rather than password-based authentication.

3. User Experience

3.1 Provide a Stable Environment

It is important to choose an appropriate operating system whose lifecycle matches your needs and resources. The advantage of enterprise distributions (like Red Hat Enterprise Linux [50] and derivatives like Scientific Linux[51]) is that the long support window reduces the number of disruptions users from major upgrades. However, the adaptation of new hardware support, system libraries, and kernel features often lag community or development distributions like Fedora [52]. Many commercial software packages only support commercial enterprise distributions or their derivatives.

On our first system, user access was constrained to queue submissions to ensure that processors were not oversubscribed. Since the login system's architecture (x86_64) was incompatible with the SMP system (ia64), users were forced to submit a job or request an interactive job to compile or test their code. Depending on utilization, users could face significant delays for development. Rather than dedicating expensive processor slots on the large system to interactive or debugging queues, we instead purchased a small development system where users could do development work and simple testing without wait and at a much lower cost per processor. This has worked so well that we have replicated this model for our cluster systems. We've also added specialized development nodes that don't have a corresponding cluster as a way to allow users to develop, test and evaluate their applications on new systems. Some have shown significant adoption by users and lead to larger cluster purchases (GPUs) and some have not (Cell processors.)

An environment modules system allows administrators to deploy multiple versions of software such that the user can choose which versions and combinations of software are appropriate for their work. Furthermore, bundles of software modules can be created for common workflows. We have switched to lmod [53] [54] in March 2012, as it addressed a longstanding bug with Environment Modules [55]. When loading modules within modules (important for some tool chains and in particular bioinformatics, where the outputs and inputs of several independently-developed tools must be combined), we would see intermittent memory corruption [56].

3.2 Naming

As the center added systems and users, the naming conventions became more of an issue. The center's first system and its partner node were named after the university's colors. After we (predictably) ran out of school colors, we expanded the naming system to use notable former faculty, but they had additional, unintended meanings. In addition, we used the same name for the host name of the development node for that cluster and as an alias for cluster as a whole. Users often misunderstood the relationship between the cluster name and the login node. To address this we designed a functional and consistent

naming convention that reflected the underlying hardware architecture of the cluster at an appropriate level of detail, providing useful but not over-specific information about the cluster. We chose to identify clusters based on the year acquired and the processor or accelerator family. However, naming critical infrastructure after the platform can be problematic when hardware is added or replaced. Our first generation ZFS file servers were named 'thor-XX', Sun's product codename for the Sun Fire X4540s. When we added new file servers, we had to name them 'thor-XX', despite not being X4540s, because of assumptions made when implementing the initial systems. We've since stopped extracting configuration information from hostnames, and instead use metadata from Facter [57].

When choosing naming standards, it is useful to keep the user in mind, as their insight can be most valuable. The default batch queue was originally named 'dque', but the abbreviation was unclear to the users; we renamed it to 'main' based on user suggestions.

Upgrading and changing software components should be done incrementally when not visible to the user, as it simplifies diagnostics if problems are detected. We've upgraded the components of the workload management and infrastructure software incrementally, but have tied major changes in user experience together (compute node operating system upgrades and changes to module structure) to minimize the overall number of changes users have to make to their workflow.

4. Communication

Ultimately, a HPC resource is a service organization and it is important that users feel informed about the resources they use. In particular, we have found that transparency is the key to this connection. Users should be able to see monitoring and utilization statistics and predictions as to when their jobs will run. It is important to avoid making changes to the system without communication. In the end, the administrators will have much more latitude in their ability to effectively administer a system if users are informed. Even what may be minor changes from a system administrator's perspective can be important to users.

Effective user communication is an ongoing challenge. It has been our experience that direct, personally addressed emails are only read slightly more often than announcements on mailing lists. Thus we have explored other avenues of communication and utilize many regularly. We send out weekly newsletters with upcoming events and funding opportunities. We post blog updates on our documentation wiki, which is syndicated via RSS. This information is republished via two Twitter accounts; one focused on system availability and maintenance information [58], and one for general center information [59].

Another important point of feedback is provided when the user can track the status of requests for help. We use RequestTracker [60] as both a help desk and a request tracking system to ensure that user requests are responded to in a reasonable timeframe and to gather metrics on staff responsiveness for reporting. While not quantified, informal feedback from users indicates *perceived* responsiveness improves when we use a ticketing system. Even basic monitoring, like the number of new tickets, open tickets, and resolved tickets can provide visibility to managers as to the current effectiveness of the staff.

Collaboration tools like wikis can be very effective, with some caveats. If you allow users to modify wiki content, staff should monitor changes. Don't anticipate that users will reduce the administrators' documentation workload; the vast majority of edits on our wiki are by staff.

If you frequently bring new users online, we have found that holding user training on standard issues (login, module system, parallel tools, etc.) in addition to one-on-one help. Such sessions can provide excellent feedback on how to improve the system or documentation. We have begun the process of moving those training sessions to video. We will provide them online, allowing users the opportunity to pick when they want to learn (or review) a topic.

User data is very, very important. You should communicate clearly with users the guarantees of the data they should expect and be sure to update them aggressively when those policies change. A failure to communicate with users early in the center's life about deletion from another temporary directory resulted in a researcher losing a significant amount of work. When implementing a new automatic deletion policy on our scratch system, after previously relying on the honor system, we archived data after the first few passes and were able to recover data for users who had not noticed the announcements we had published.

Reporting is important for internal and external use. Hard metrics like CPU-time consumed, utilization, and average wait time are useful, but soft metrics like papers published, grants received, resources discovered or products designed can be more useful for the majority of resources that are not leadership-class facilities [61].

Inspired by a conversation with staff from the University of Michigan's CAEN Advanced Computing Group, we've also instrumented our environment module infrastructure with syslog to track software usage. Harvard University's FAS Research Computing has also implemented a similar approach [62].

Well-developed internal technical documentation aids consistency in implementation and between administrators and reduces the amount of time administrators spend re-implementing fixes. External technical documentation (ideally, with domain examples) reduces the learning curve for new users significantly. If policies for users are not visible, they don't exist. Users have perceived the enforcement of poorly documented procedures as capricious targeting by administrators. When users can see the potential benefit of enforcement they usually accept the need for such measures.

5. Other Resources

We have used external resources when we found them to be particularly suitable. The center was able to use our campus Kerberos authentication and to gather information about users from the campus LDAP. Some of our Ethernet network switches and our external firewall are managed by campus IT, and HPC staff manages HPC-focused resources (storage, high speed networking.) We try to avoid duplication of efforts and try to focus our resources on the goals of our center.

External resources include people as well as technology. We've worked with IT staff from other departments on campus that use our systems to add domain-specific support for users in their departments. By utilizing their knowledge, we were able to provide a better service for the resource's users.

Domain experts are valuable resources for users and for administrators as a bridge between dedicated systems administrators and users. They are as important of a component of the ecosystem as the hardware, software, and facilities.

The Information Technology Infrastructure Library [63] has been useful when addressing some of the challenges in this paper. While the author was initially skeptical, there have been a number of times it has been helpful when understanding the ways our procedures could improve. The library's sheer size and scope make it difficult implement whole cloth. We've had the best luck when using components of that fit well within the environment and as a reference when implementing support workflows.

Attending technical conferences, joining mailing lists, listening to technical podcasts and reading personal and professional blogs has been very helpful; networking with peers can provide some valuable insight into HPC.

6. CONCLUSION

We have encountered many challenges when deploying a HPC resource; understanding the service sponsor's goals, the requirements of the resource's users, the technologies and policies best suited to meet them, and the risks associated with providing that resource. We have found that published best practices and advice from colleagues can help staff understand HPC requirements more easily.

Have a clearly defined service sponsor and understand what their goals are for the service. Ideally, the service sponsor would be the person responsible for deciding whether to continue the project. A single person who can arbitrate between different stakeholders ensures that there is clear direction for the staff.

Make sure that you understand your stakeholders' needs and bring them into the decision making process; try to ensure that you have a representative sampling on any advisory boards. Your users should be invested in the success of the resource. Your harshest critics can provide useful information about your areas for improvement.

7. Acknowledgements

Michigan State University, for its recognition and support of HPC services on campus. Dr. Wolfgang Bauer, the current Institute director. Dr. Leo Kempel, who helped found the Center. Dr. Dirk Colbry and Dr. Ben Ong, research specialists, current and previous colleagues at the Center, including Kelly Osborn, Ed Symanzik, Ed Kryda, Eric McDonald, Jim Leikert, Greg Mason and John Johnston. The Practice of System and Network Administration by Thomas A. Limoncelli, Christina J. Hogan, and Strata R. Chalup, great reading for any system administrator. Doug Hughes for his invaluable help with this paper, and the LISA09 workshop on HPC.

[1] "SGI TPL (Linux: Developer/LX_AppTune - Chapter 7. Flexible File I/O)." [Online]. Available: http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=bks&fname=/SGI_Developer/LX_AppTune/ch07.html. [Accessed: 12-Sep-2012].

[2] "Performance Co-Pilot." [Online]. Available: http://oss.sgi.com/projects/pcp/. [Accessed: 12-Sep-2012].

[3] "Warewulf." [Online]. Available: http://warewulf.lbl.gov/trac. [Accessed: 12-Sep-2012].

[4] "www.rocksclusters.org | Rocks Website." [Online]. Available: http://www.rocksclusters.org/wordpress/. [Accessed: 12-Sep-2012].

[5] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "NPACI Rocks: tools and techniques for easily deploying manageable Linux clusters," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 7–8, pp. 707–725, Jun. 2003.

[6] "Bright Cluster Manager - Advanced Linux Cluster Management Software." [Online]. Available: http://www.brightcomputing.com/Bright-Cluster-Manager.php. [Accessed: 12-Sep-2012].

[7] "IBM Platform HPC." [Online]. Available: http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/hpc/index.html. [Accessed: 12-Sep-2012].

[8] D. Cable and M. Diakun, *A Review of Commodity Cluster Managment Tools*. Science and Technology Facilities Council, 2011.

[9] "Dell OpenManage Deployment Toolkit - Systems Management - Wiki - Systems Management - Dell Community." [Online]. Available: http://en.community.dell.com/techcenter/systems-management/w/wiki/1772.dell-openmanage-deployment-toolkit.aspx. [Accessed: 17-Sep-2012].

[10] "Dell PowerEdge C Server | System Management." [Online]. Available: http://poweredgec.com/. [Accessed: 12-Sep-2012].

[11] "Optimal BIOS settings for HPC workloads - High Performance Computing - Blog - High Performance Computing - Dell Community." [Online]. Available:

http://en.community.dell.com/techcenter/high-performance-computing/b/weblog/archive/2012/08/17/optimal-bios-settings-for-hpc-workloads.aspx. [Accessed: 17-Sep-2012].

[12] "How to Enable 'HPC-mode' to Achieve up to 6% Improvement in HPL Efficiency - General HPC - High Performance Computing - Dell Community."[Online]. Available: http://en.community.dell.com/techcenter/high-performance-computing/b/general_hpc/archive/2012/07/05/how-to-enable-quot-hpc-mode-quot-to-achieve-up-to-6-improvement-in-hpl-efficiency.aspx. [Accessed: 17-Sep-2012].

[13] "MATLAB Distributed Computing Server - Supported Schedulers." [Online]. Available: http://www.mathworks.com/products/distriben/supported/index.html. [Accessed: 17-Sep-2012].

[14] "ANSYS High Performance Computing Features." [Online]. Available: http://www.ansys.com/Products/Workflow+Technology/High-Performance+Computing/Features. [Accessed: 17-Sep-2012].

[15] "SystemImager." [Online]. Available: http://systemimager.sourceforge.net/. [Accessed: 17-Sep-2012].

[16] "pdsh - Parallel Distributed Shell - Google Project Hosting." [Online]. Available: https://code.google.com/p/pdsh/. [Accessed: 17-Sep-2012].

[17] "Anaconda/Kickstart - FedoraProject." [Online]. Available: http://fedoraproject.org/wiki/Anaconda/Kickstart. [Accessed: 17-Sep-2012].

[18] "What is Puppet? | Puppet Labs." [Online]. Available: http://puppetlabs.com/puppet/what-is-puppet/. [Accessed: 17-Sep-2012].

[19] L. Kanies, "Puppet: Next-generation configuration management," *The USENIX Magazine. v31 i1*, pp. 19–25, 2006.

[20] "Git - About." [Online]. Available: http://git-scm.com/about. [Accessed: 17-Sep-2012].

[21] "Git Workflow and Puppet Environments | Puppet Labs." [Online]. Available: http://puppetlabs.com/blog/git-workflow-and-puppet-environments/. [Accessed: 17-Sep-2012].

[22] "Cacti® - The Complete RRDTool-based Graphing Solution." [Online]. Available: http://www.cacti.net/. [Accessed: 17-Sep-2012].

[23] "Nagios - The Industry Standard in IT Infrastructure Monitoring." [Online]. Available: http://www.nagios.org/. [Accessed: 17-Sep-2012].

[24] "Ganglia Monitoring System." [Online]. Available: http://ganglia.sourceforge.net/. [Accessed: 17-Sep-2012].

[25] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, "Wide area cluster monitoring with ganglia," in *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, 2003, pp. 289–298.

[26] "10.2 Compute Node Health Check." [Online]. Available: http://www.clusterresources.com/torquedocs/10.2healthcheck.shtml. [Accessed: 17-Sep-2012].

[27] "Moab Workload Manager - Object Triggers." [Online]. Available: http://www.adaptivecomputing.com/resources/docs/mwm/19.0triggers.php. [Accessed: 17-Sep-2012].

[28] "xCAT - Extreme Cloud Administration Toolkit." [Online]. Available: http://xcat.sourceforge.net/. [Accessed: 17-Sep-2012].

[29] D. M. Ritchie and K. Thompson, "The UNIX time-sharing system," *Commun. ACM*, vol. 17, no. 7, pp. 365–375, 1974.

[30] "MATLAB Distributed Computing Server - Level of Support for Schedulers." [Online]. Available: http://www.mathworks.com/products/distriben/supported/level-of-support.html#MWjobmanager. [Accessed: 17-Sep-2012].

[31] "Introduction to the Wolfram Lightweight Grid System - Wolfram Mathematica 8 Documentation." [Online]. Available: http://reference.wolfram.com/mathematica/LightweightGridClient/tutorial/Introduction.html. [Accessed: 17-Sep-2012].

[32] "Moab HPC Suite Basic Edition." [Online]. Available: http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/. [Accessed: 17-Sep-2012].

[33] "BLCR » FTG." [Online]. Available: https://ftg.lbl.gov/projects/CheckpointRestart/. [Accessed: 17-Sep-2012].

[34] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (blcr) for linux clusters," in *Journal of Physics: Conference Series*, 2006, vol. 46, p. 494.

[35] "New Powertool to help checkpoint jobs - Dirk Joel-Luchini Colbry - HPCC Wiki." [Online]. Available: https://wiki.hpcc.msu.edu/display/~colbrydi@msu.edu/2011/10/06/New+Powertool+to+help+checkpoint+jobs. [Accessed: 17-Sep-2012].

[36] "Moab Workload Manager - Managing Consumable Generic Resources." [Online]. Available: http://www.adaptivecomputing.com/resources/docs/mwm/12.6consumablegres.php. [Accessed: 17-Sep-2012].

[37] "Appendix J: Job Submission Filter (aka 'qsub Wrapper')*."[Online]. Available: http://www.clusterresources.com/torquedocs21/a.jqsubwrapper.shtml. [Accessed: 18-Sep-2012].

[38] "Wiki Front Page - Whamcloud Community Space - Whamcloud Community." [Online]. Available:

http://wiki.whamcloud.com/display/PUB/Wiki+Front+Page. [Accessed: 17-Sep-2012].

[39] P. Schwan, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux Symposium*, 2003, vol. 2003.

[40] J. Bonwick and B. Moore, "Zfs: The last word in file systems," *online][retrieved on Jan. 22, 2008] Retrieved from the Internet*, 2007.

[41] "SGI - Developer Central Open Source | XFS." [Online]. Available: http://oss.sgi.com/projects/xfs/. [Accessed: 17-Sep-2012].

[42] "About illumos - illumos - illumos wiki." [Online]. Available: http://wiki.illumos.org/display/illumos/About+illumos. [Accessed: 17-Sep-2012].

[43] "ZFS Dedup FAQ (Community Group zfs.dedup) - XWiki." [Online]. Available: http://hub.opensolaris.org/bin/view/Community+Group+zfs/dedup. [Accessed: 17-Sep-2012].

[44] "The Blog of Ben Rockwood." [Online]. Available: http://cuddletech.com/blog/pivot/entry.php?id=983. [Accessed: 17-Sep-2012].

[45] N. Rasmussen, "Air distribution architecture options for mission critical facilities," *ELEKTRON JOURNAL-SOUTH AFRICAN INSTITUTE OF ELECTRICAL ENGINEERS*, vol. 22, no. 10, p. 68, 2005.

[46] "National Petascale Computing Facility." [Online]. Available: http://www.ncsa.illinois.edu/AboutUs/Facilities/npcf.html. [Accessed: 17-Sep-2012].

[47] "Precision Cooling - High Density Modular Cooling." [Online]. Available: http://www.emersonnetworkpower.com/en-US/Products/PrecisionCooling/HighDensityModularCooling/Refrigerant-Based/Pages/Default.aspx.

[48] "Main Page - SELinux Wiki." [Online]. Available: http://selinuxproject.org/page/Main_Page. [Accessed: 17-Sep-2012].

[49] "Fail2ban." [Online]. Available: http://www.fail2ban.org/wiki/index.php/Main_Page. [Accessed: 17-Sep-2012].

[50] "Red Hat | Red Hat Enterprise Linux for Scientific Computing – HPC." [Online]. Available: http://www.redhat.com/products/enterprise-linux/scientific-computing/. [Accessed: 17-Sep-2012].

[51] "Scientific Linux - Welcome to Scientific Linux (SL)." [Online]. Available: https://www.scientificlinux.org/. [Accessed: 17-Sep-2012].

[52] "Fedora Project - What is Fedora and what makes it different?" [Online]. Available: http://fedoraproject.org/en/about-fedora. [Accessed: 17-Sep-2012].

[53] "Lmod | Free Development software downloads at SourceForge.net." [Online]. Available: http://sourceforge.net/projects/lmod/. [Accessed: 17-Sep-2012].

[54] R. McLay, K. W. Schulz, W. L. Barth, and T. Minyard, "Best practices for the deployment and management of production HPC clusters," in *State of the Practice Reports*, 2011, p. 9.

[55] "Modules -- Software Environment Management." [Online]. Available: http://modules.sourceforge.net/. [Accessed: 17-Sep-2012].

[56] "Modules that load modules segfault when unloading due to invalid memory accesses." [Online]. Available: http://sourceforge.net/mailarchive/forum.php?thread_name=201204201559.01764.twhitehead%40gmail.com&forum_name=modules-interest. [Accessed: 17-Sep-2012].

[57] "The Facter Program Quickly Gathers Basic Node Information | Puppet Labs." [Online]. Available: http://puppetlabs.com/puppet/related-projects/facter/. [Accessed: 17-Sep-2012].

[58] "HPCC@MSU (hpccmsu) on Twitter." [Online]. Available: https://twitter.com/hpccmsu. [Accessed: 17-Sep-2012].

[59] "iCER (icermsu) on Twitter." [Online]. Available: https://twitter.com/icermsu. [Accessed: 17-Sep-2012].

[60] "RT: Request Tracker - Best Practical." [Online]. Available: http://bestpractical.com/rt/. [Accessed: 17-Sep-2012].

[61] S. C. Ahalt, A. Apon, A. H. P. C. C. Director, D. Lifka, and H. Neeman, "Sustainable Funding and Business Models for Academic Cyberinfrastructure Facilities: Workshop Report and Recommendations," in *Workshop Report and Recommendations*, 2010.

[62] "fasrc/module-usage · GitHub." [Online]. Available: https://github.com/fasrc/module-usage. [Accessed: 18-Sep-2012].

[63] "ITIL® Home." [Online]. Available: http://www.itil-officialsite.com/. [Accessed: 18-Sep-2012].

Building a Wireless Network for a High Density of Users

David Lang - Intuit

Abstract:

Why do conference and school wireless networks always work so poorly? As IT professionals we are used to the network 'just working' and fixing things by changing configuration files. This mind-set, combined with obvious-but-wrong choices in laying out a wireless network frequently lead to a network that seems to work when it's tested, but that then becomes unusable when placed under load. This is at its worst at technical conferences where there are so many people, each carrying several devices, all trying to use the network at the same time, and in schools where you pack students close together and then try to have them all use their computers at the same time.

Is this a fundamental limitation of wireless? While it is true that there are some issues that cannot be solved, there are a lot of things that the network administrator can do to make the network work better. The key issue is the obvious, but under-appreciated fact that wireless networking is radio communications first. If your radio link doesn't work well, you have no chance of fixing it with your configuration and software. This paper is intended to give you an appreciation of what the issues are, and enough information to know what sorts of things to look out for when planning a high density wireless network.

Introduction and Overview

Wireless networks for conferences and schools tend to work very well when tested, and then collapse completely when all the users show up to use them. This pattern is repeated time and time again to the point where people tend to think that it's a fundamental limitation of Wi-Fi technology. There are real limitations that you have to deal with, but if you keep them in mind it is very possible to build a wireless network for thousands of people and have it be rock solid and reliable.

I have been running the Wireless network for the Southern California Linux Expo (SCaLE) since 2010 and this paper is based on the results of the attempts to provide wireless service from 2007 through 2012 at SCaLE. In 2012 we had 1965 attendees with 1935 unique MAC addresses on the network and 875 devices connected at peak.

The key thing to recognize when building a wireless network is that the network is primarily radios, and only secondarily digital. This doesn't mean that getting the radio side of things right will guarantee that your network will work, but it does mean that getting it wrong will guarantee that your network will not work.

Background

I

Prior to 2007, SCaLE only offered network capability to the booths on the trade show floor. To deal with rogue DHCP servers on the network, each booth is put in its own VLAN and network.

In 2007 SCaLE decided to start offering wireless network access. SCaLE purchased 10 3Com dual band access points and distributed them around the hotel. They discovered that these APs only allowed 32 devices per AP and as a result, they quickly became saturated and the wireless network was completely unusable.

In 2008, SCaLE borrowed 11 Linksys access points and the network administrator found a firmware hack that allowed him to boost the power. He put each access point on a different channel. This was also the year that the One Laptop Per Child laptops became available to the public and I volunteered at the OLPC booth demoing the laptops. The wireless network was unusable, in large part due to the interference between adjacent APs. After the event I spoke with the organizers and cringed when I heard how the wireless had been setup. I volunteered to help in the future.

In 2009, Xirrus donated wireless services to SCaLE. I don't have any details of what they setup, but the result was unusable.

In 2010, I received a call one month before the event, telling me that the commercial vendor that they had lined up to provide the wireless services had backed out, and since I had offered my expertise in 2008, was I interested in trying to run wireless services. I said yes and analyzed the problems.

I focused first on the radio side of the environment. If the radio side doesn't work, you have no chance of making the digital side work well, and you frequently won't even be able to tell what's wrong by looking at the network information.

Defining the Problem.

Radio Issues:

The 2.4GHz band (b/g) has 11 channels assigned in the US, but they overlap and as a result, you can only use 3 of the channels at once without problems.[1]

1 If you can use channels 13 and 14 in your county you can squeeze in a forth channel.

The 2.4HGz band is also used extensively by other equipment, including cordless phones, cordless microphones, bluetooth, and even microwave ovens. While the 802.11 protocol is designed to be resistant to interference from these things, these things can cause packets to be corrupted and result in retries.

The 5GHz band does not have these problems. It's channels do not overlap, there are far more of them (at least 10 available, sometimes more), and there are far fewer sources of other interference. Unfortunately, most equipment doesn't support the 5Ghz band, and even when equipment does support it, many systems default to the 2.4GHz band.

Standard Wifi, like many mobile radio services, suffer from the Hidden Transmitters problem. A simplified description of this is where you have three stations along a line. The station in the middle can hear stations on each side, but the stations on the outside cannot hear each other. This prevents the stations on each side from avoiding transmitting when the one on the other side is already transmitting. When both sides transmit at the same time, the receiving station in the middle gets confused and can't make out either signal, causing both to have to retransmit the packet.

Excessive power levels can add to the Hidden Transmitter problem. It is common to think that if you can't get through, turn up the power, but if only one side turns up the power it seldom improves communications. This is because wireless networks are two-way conversations and if only one side gets louder it doesn't increase the range that the conversation can take place, but the stronger signal does go further and interferes with other stations.

The Wi-Fi protocols have evolved over time, with new modes being created that squeeze more data into a given amount of airtime. In most cases the newer, higher speed modes are more sensitive to interference, so the protocol includes fall-backs to slower modes when the data is not getting through. If the problem is outside interference, weak signal and similar problems, this works very well, but if the problem is an overload of the available airtime, the result is that each station transmitting takes longer to send its signal, which makes it more likely that a hidden transmitter or other interference will corrupt the packet resulting retries.

802.11 has a fair amount of housekeeping traffic to let all stations in the area know that they exist and to maintain the connection to the Access Point. This traffic eats away at the time available and is frequently required by the spec to be transmitted at the lowest supported speed.[2]

802.11n can be a benefit or a problem. The fact that it can transmit more data in a given amount of airtime can reduce congestion, but if you enable the high bandwidth (dual channel) mode it will require

2 Any broadcast traffic (such as SSID broadcasts, connection requests, etc) must be transmitted at the lowest speed supported so that devices that only support that speed and not higher ones will still be able to decode the message.

that two adjacent channels be allocated to it. Also, if the equipment is configured to operate in pure 802.11n mode, the 802.11b/g equipment will not recognize that there is a station transmitting and so will go ahead and attempt to transmit their packet.

Inappropriate use of high-gain antennas can be a problem as well. Unlike turning up the transmitter power, improved antennas help both transmitting and receiving the signal. But if they are used incorrectly they will cause the station using them to cover a larger area and so interfere with and be interfered by more stations.

Inappropriate access point/antenna positioning can have very similar effects to using the wrong antennas. It's very tempting to try and get the best possible coverage from each access point, but when you are trying to get the most number of users in a small area, this can actually hurt you. It's sometimes as simple as changing the height of the access point to limit how far it's signals will travel.

Mesh Networks require that the packets be transmitted over the radio more times, and as a result are almost always the wrong thing to use in a high-density environment.[3]

Retries can also be caused by problems on the digital side of things.

The Bufferbloat phenomenon[4] where the delays in getting packets to their destination can result in the packets timing out before they arrive can also result in packets being retransmitted.

The typical collapse of wireless networks results from the combination of:

- Retries (frequently due to hidden transmitters or other interference)
- Fall-back to slow speeds
- Wasted packets (due to bufferbloat and other problems)

3 In this case I am referring to wireless links between the Access Points, In this case the traffic from many users is combined onto the uplink channel, the APs are using high power on the uplink channel and therefor the uplink channel is even more congested than the normal channels, resulting in them collapsing before the normal traffic does. Full 802.11s/OLPC style mesh networks collapse even faster as the packets are retransmitted over the normal channels.

4 In an attempt to prevent packet loss, and with memory becoming vastly cheaper over time, buffers on network devices have become very large. If there is significant congestion and the buffers stay full for an extended time, packets can sit in the buffers long enough that by the time they arrive at their destination they have already timed out and a replacement is in flight.

Solutions:

I needed to find out what I was up against. I did a site survey to find out what the situation was.

- Where are the network and power jacks (I've had cases where they were >8 feet apart)
- What other Wi-Fi signals are in the area, what channels are they on?

  Some places in the Hotel were covered by 5 different existing networks, including networks from other nearby hotels.

  Good tools to use are Wi-Fi analyzer on Android or Kismet on a laptop
- What interference is there in the area (usually not as critical as looking for Wi-Fi signals)

  My-Spy spectrum analyzer can see all signal, not just WiFi signals.
- What effect do the walls have on your signal (movable partitions tend to block the signal more than traditional walls due to the metal mesh in the partitions)

  I took an AP to plug in and then walked around nearby rooms and hallways to find out where I could hear it.

Once I knew what the environment is like, I worked to get as many access points in the area as you can get without them interfering with each other and without creating additional hidden transmitter situations.

The fundamental approach to making a lot of people able to use wireless in a small area is to use many low-power access points instead of a small number of high power access point. Cell phone service has a similar problem and solution, they refer to this as setting up microcells.

I encouraged the use of 5GHz channels. There are far more of them so you can have more radios covering a given amount of floorspace without interference, resulting in significantly more bandwidth per user. In addition to this making things better for the people who move to 5GHz, it also reduces the load on the 2.4GHz band, helping the people who don't move.

I turned power down on 2.4Ghz to allow for more access points without overlapping footprints.

I positioned the APs to take advantage of things that block the signal for me.

- The human body is mostly water, water absorbs 2.4GHz signals. By putting access points low in the room, the crowds will prevent their signal from going as far as they normally would.
- In the site survey I found out which walls block the signal. I was able to position some access

points closer to each other than I normally would have allowed.

I used advanced antennas carefully.

- I used directional antennas to cover a long theater room from the back of the room where I didn't have the ability to position access points more centrally.
- I also used mild directional antennas to direct the signal away from areas that were covered by other access points.

Digital Issues:

For SSID selection, I opted to use one SSID for each band (scale24 for 2.4GHz and scale5 for 5GHz), and re-use the same SSID on every access point. While putting a different SSID on each access point gives the user more control, the fact that using the same SSID everywhere allows the client to roam between access points as they move, without the user needing to do anything.

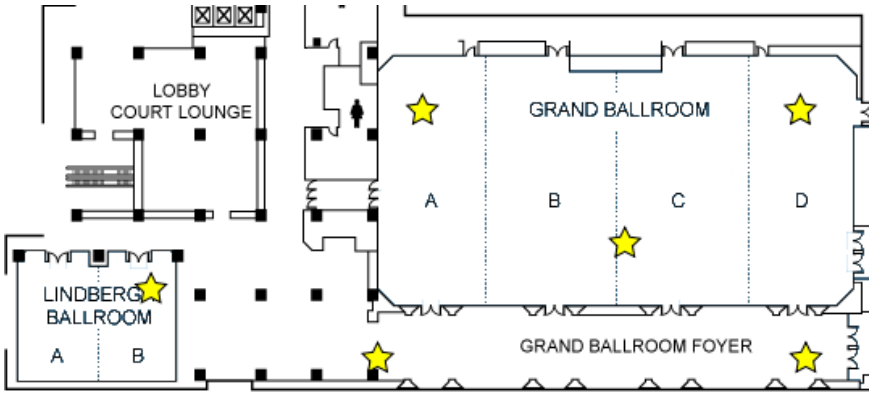


Illustration 1: LAX Westin 1st floor 2010

To make this work I configured the access points to act as bridges instead of routers, and run DHCP on a central server instead of on each access point. This makes it so that the IP address that a device gets continues to be valid as they move around the building.

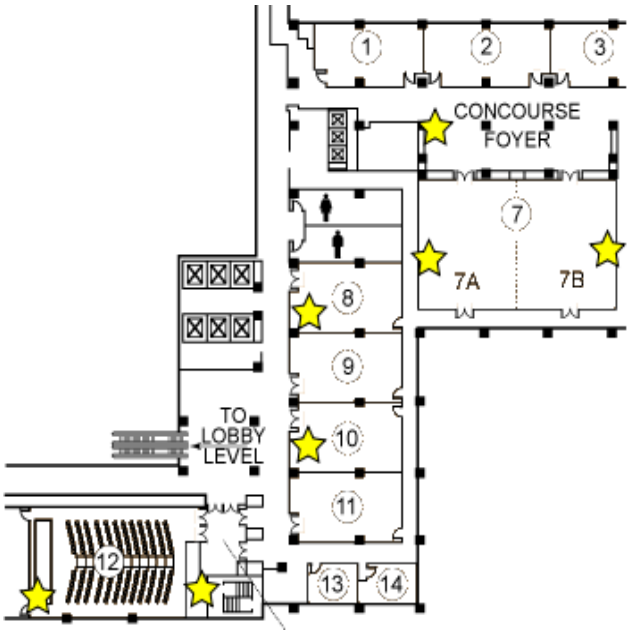


Illustration 2: LAX Westin 2nd floor 2010

Implementation.

2010

We purchased 16 5GHz NetGear APs ($50 each) and 12 2.4GHz Fry's Electronics APs ($30 each), along with three Cisco 160 APs that were purchased early on before testing the Fry's APs.

We implemented normal site bandwidth saving tools:

- HTTP caching proxy (squid)
- Block streaming sources (DNS redirects to a placeholder page)
- QOS traffic shaping to allocate bandwidth between users ('users' being wireless users vs registration vs keynote streaming video, etc.)

Due to the extremely short timeframe, I turned down the power to 'low' with the stock firmware and crossed my fingers. I used directional antennas on the Cisco APs to direct their coverage area to minimize overlap with other APs.

Wireless worked well enough to crush the available Internet bandwidth (4.5Mb) for the first time. Overload caused the wireless network to be unusable.

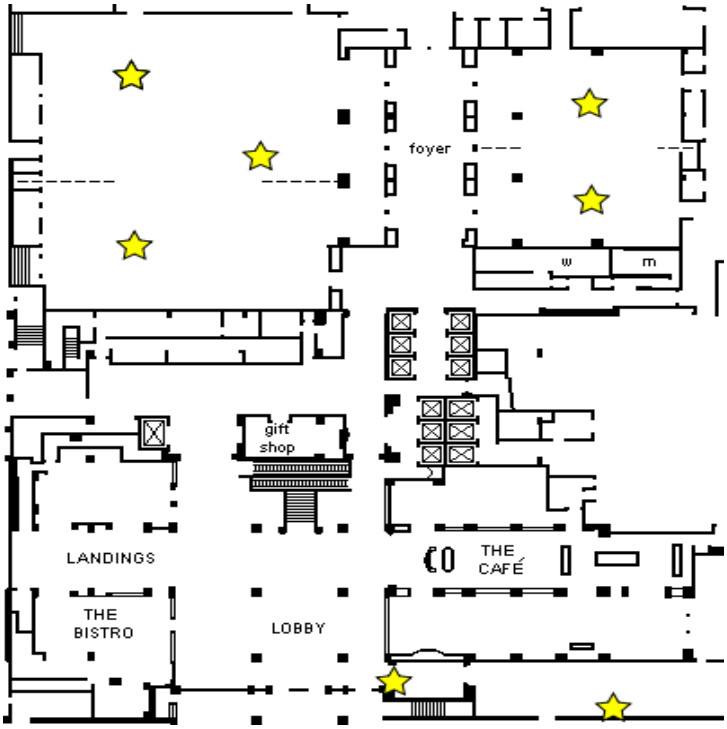


Illustration 3: LAX Hilton 1st floor 2011

2011

SCaLE moved to new Hotel (LAX Hilton) with approximately 50% larger area to cover.

This hotel has 45Mb Internet connection. Prior to this SCALE conference they had never had it turned up above 20Mb. We had them enable full bandwidth, and kept it as close to saturated as our QoS settings would manage for most of the show.

I attempted to use just the equipment purchased in 2010, but with DD-wrt on the 2.4GHz APs. I was very unhappy with DD-WRT. It's more flexible and powerful than the stock firmware, but it does most of its configuration with special variables stored in NVRAM rather than with traditional *nix config files.

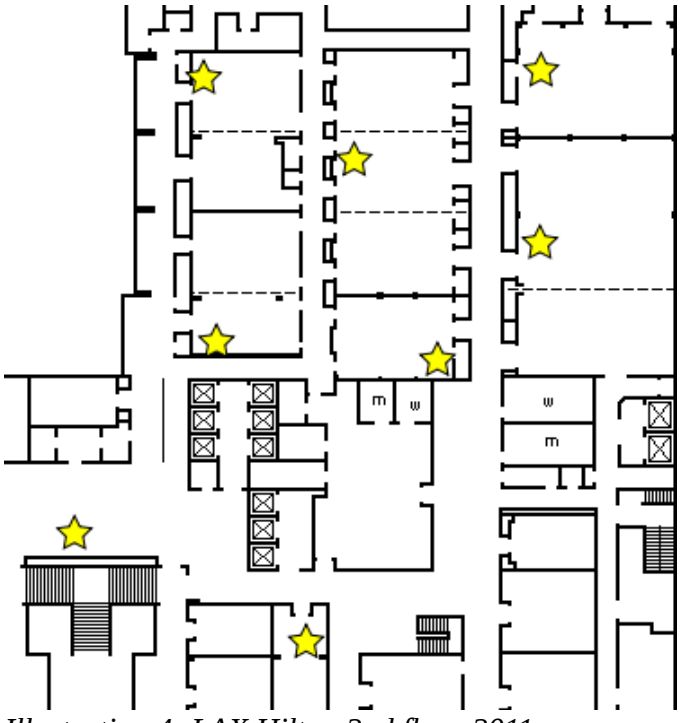


Illustration 4: LAX Hilton 2nd floor 2011

We had many problems with the 5GHz NetGear access points freezing, and with all the access points getting turned off, and with the network cables getting disconnected. The cables getting disconnected turned out to be the most disruptive problem, because the access points continue to advertise the SSID, but any clients connecting to them were dead in the water.

We found that we had insufficient access points to provide good coverage. When you were in a good spot things would work, but far too many areas and rooms did not work.

2012

SCaLE purchased 30 dual-band WNDR3700 APs ($130 each). We ran them all on OpenWRT, with a custom compile. I kept the same locations for the access points on the first fllor, and greatly increased the number deployed on the second floor.

I enabled Wireless Isolation. This prevents the wireless devices from talking directly to each other. This will break some use cases, but for the normal case where devices are talking to servers on the wired network it can both add some protection for the clients, and greatly reduce the wireless bandwidth needed. IP level broadcasts result in ***many*** retransmissions on wireless networks.

I lengthened the Beacon interval, it reduces the amount of housekeeping traffic, at the cost of it taking longer for devices to learn that the network is there or notice new APs as the users move around the building. With lots of access points there is enough overlap to minimize problems, and people are usually not moving that fast when heavily using the network.

I disabled connection tracking in the custom kernel. Connection tracking can be a very significant overhead on the CPU and RAM of the AP. Connection tracking is needed to implement Stateful Packet Filtering, but if you are not using any stateful firewall rules, it can

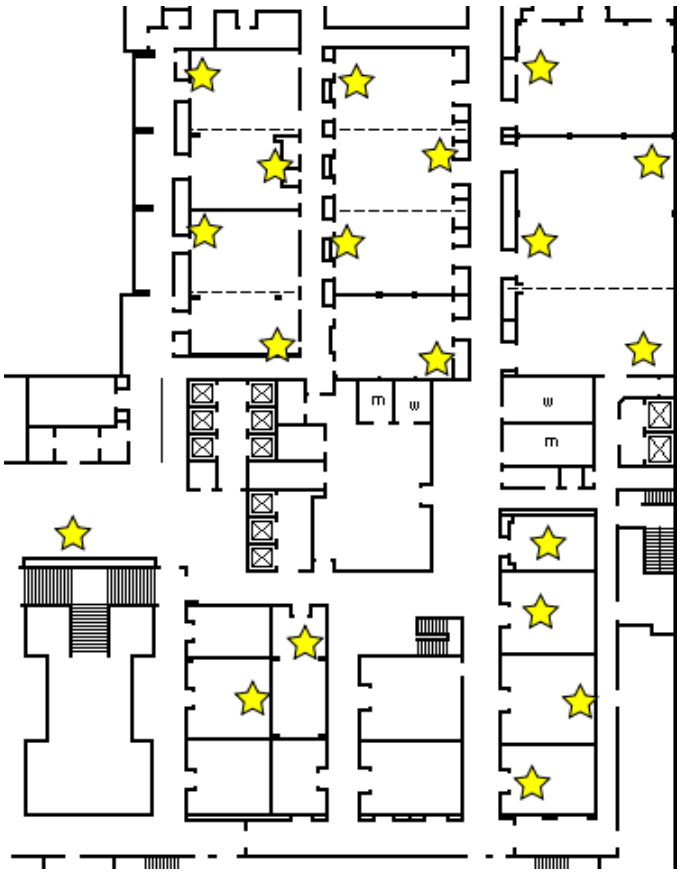


Illustration 5: LAX Hilton 2nd floor 2012

result is a significant amount of memory and CPU overhead for no benefit.

I set shorter than normal inactivity timers so that the APs don't spend resources trying to track devices that have moved or been turned off.

I adjusted kernel network buffers to be much smaller than normal to fight bufferbloat problems (50 instead of 1000). The Linux wireless stack includes quite a bit of buffering inside it, so setting the kernel buffers for the wireless interfaces very low helps minimize the possibility of excessive latency. There is some recent work in this area, but it does not yet deal with the buffers inside the wireless stack.

Running a web proxy also significantly helps fight the bufferbloat problem because by splitting the connections, you avoid having TCP connections with both high latency (long ping times of International Internet connections) and widely varying effective bandwidth of wireless. connections. The widely variable bandwidth doesn't hurt much if it's only connecting to a local proxy server, and that proxy server has stable bandwidth out to the Internet. This is undermined to some extent by the “https everywhere” movement because https connections cannot be proxied.

As a result of the problems we had in 2011 with devices getting unplugged, we setup Nagios, cacti, and some custom rrdtool scripts to track what was happening on the devices.

We moved from having separate switch ports for each wireless band to having one connection per access point and using VLANs to isolate the administration, 2.4GHz, and 5GHz networks from each other.

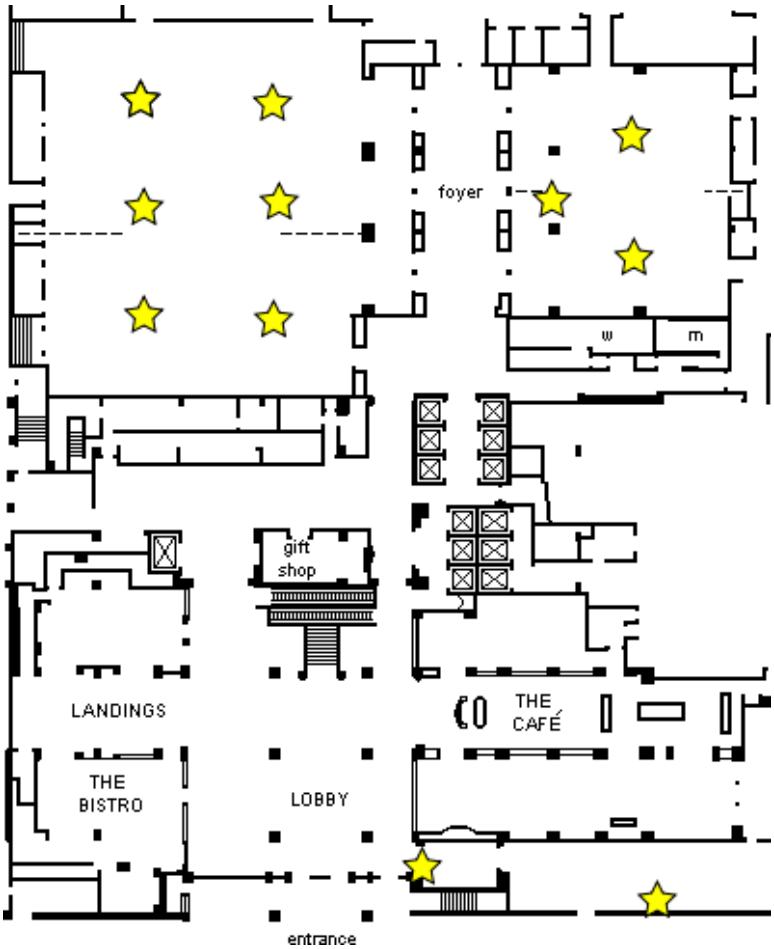


Illustration 6: LAX Hilton 1st floor 2013 (planned)

Results:

The wireless network was rock solid. Approximately 20% of the devices used the 5GHz band. Except for on the show floor, we had fewer than 70 devices connected to any one access point at one time. We had 1965 attendees over the three days with 1935 unique MAC addresses on the network and 875 devices connected at peak.

Illustration 7: LAX Hilton 2nd floor 2013 (planned)

Future Plans

In 2013 we are expanding again. We will be taking over the rest of the hotel, expanding into more of the smaller rooms, using them for additional talks, BoF sesstions, and streaming video from the main rooms to handle overflow. This will require an additional 20 access points, and the access points are going to be serving as the main switches in some of the rooms, connecting the A/V gear to the network for the streaming video. We have not yet done the site survey for these additional rooms as of the time of writing, so we may end up turning off some of the radios on the access points in these smaller rooms.

I will be looking to disable slow speeds. If you can disable the 802.11b speeds entirely you avoid having systems falling back to the extremely slow speeds and using more air time to transmit the same data, making the congestion problem worse. There are very few devices today that don't support at least 802.11g.

About the Author.

David Lang is a Staff IT Engineer at Intuit, where he has spent over a decade working in Security Department for the Banking division. He was introduced to Linux in 1993 and has been making his living with Linux (and using it as his desktop) since 1996. He is a Extra Class Amateur Radio Operator and served on the Civil Air Patrol California Wing Communications staff, where his duties included managing the state-wide digital wireless network. He has been running the wireless network at the Southern California Linux Expo since 2010. He is also active on various Open Source mailing lists. He can be contacted via e-mail at david@lang.hm, by phone at +1 818 292 7015

This paper and related materials are available at http://talks.lang.hm/events/LISA_2012/wireless