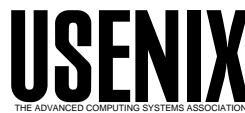USENIX Association

Proceedings of the 17th Large Installation Systems Administration Conference

San Diego, CA, USA
October 26–31, 2003

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SmartFrog meets LCFG: Autonomous Reconfiguration with Central Policy Control

Paul Anderson – University of Edinburgh
Patrick Goldsack – HP Research Laboratories
Jim Paterson – University of Edinburgh

ABSTRACT

Typical large infrastructures are currently configured from the information in a central configuration repository. As infrastructures get larger and more complex, some degree of autonomous reconfiguration is essential, so that certain configuration changes can be made without the overhead of feeding the changes back via the central repository. However, it must be possible to dictate a central policy for these autonomous changes, and to use different mechanisms for different aspects of the configuration.

This paper describes a framework which can be used to configure different aspects of a system using different methods, including explicit configuration, service location, and various other autonomous techniques. The proven LCFG tool is used for explicit configuration and to provide a wide range of configuration components. The dynamic elements are provided by the SmartFrog framework.

Introduction

Typical large infrastructures (see [7] for some case studies) are currently configured from the information in a central configuration repository, such as sets of hand-crafted cfengine [11] scripts, or LCFG [9] source files. Changes to individual nodes are made by editing these central descriptions and running the appropriate tool to reconfigure the node.

As infrastructures get larger and more complex, some degree of autonomous reconfiguration is essential, so that individual nodes (and clusters) can make small adjustments to their configuration in response to their environment, without the overhead of feeding changes back via the central repository. For example, the members of a cluster might elect a replacement for a failed server amongst themselves, without requiring a change to the central configuration server; see the example in Figure 1.

Peer-to-peer style autonomous reconfiguration is already present in several tools, such as ZeroConf [17] (as used by Apple's Rendezvous) and other systems using Service Location Protocols (for example, [19]).

However, a completely autonomous approach is not suitable for large installations; there needs to be some central control over the policy under which the autonomous choices are made; for example, exactly which nodes are eligible to be elected as a replacement server? There will also be a good deal of configuration information that does need to be specified explicitly, and there may need to be several different simultaneous techniques for making autonomous decisions.

LCFG [9] is a proven, practical tool for centralized configuration management of large, diverse infrastructures. SmartFrog [16] is a flexible, object-oriented framework for deployment and configuration of remote Java objects. This paper describes the architecture of a combined LCFG/SmartFrog framework which uses LCFG to install a complete system (including SmartFrog) from scratch. SmartFrog components on the resulting system are then able to take control of arbitrary LCFG components and configure them autonomously, according to policies defined in the central LCFG configuration database.

This approach allows the configuration of various aspects of the system to be shifted easily between explicit

Explicit Specification:
- Node X must run a print server
- Node Y must run a print server

Policy Specifications with autonomous configuration:
- Any Linux server may run a print server
- Nodes A,B,C,V,X,Y,Z are servers
- Nodes P,Q,R,X,Y,Z are Linux machines
- There must be exactly two print servers

The nodes can decide ''among themselves'' which of the eligible nodes actually run as print servers. If one of them fails, they can re-elect a replacement without a change to the central policy.

Figure 1: Explicit specification vs policy specification – an example.

central specification, and autonomous control, using one of several different procedures for making the autonomous decisions, and performing the related peer-to-peer communication. No change is required to the software components that actually implement the configuration.

The combined framework makes an ideal test bed for experimenting with different models of autonomous configuration in a real environment. This paper describes the testbed, together with some demonstrator applications, including a complete Grid-enabled (OGSA [15]) printing service, with dynamic server re-allocation and failure recovery.

The next section describes the background in more detail, including an overview of the LCFG and SmartFrog tools, and the motivation towards more dynamic reconfiguration. Subsequent sections describe the combined LCFG/SmartFrog framework, present some simple example applications, and explicate the OGSA print service demonstrator.

Background

LCFG

LCFG is an established configuration framework for managing large numbers of Unix workstations. Originally developed under Solaris (see [4]) using NIS to transport configuration parameters, the current version (see [9]) runs under Linux and uses XML/HTTP for parameter transport. LCFG acts as an evolving testbed for configuration research, as well as a production system for the infrastructure in the School of Informatics at Edinburgh University. An older version is also in use on the testbeds for the European Data-Grid Project [1]. LCFG includes around 70 modules for managing a wide range of different subsystems,

ranging from PCMCIA configuration on laptops, to OGSA web services for Grid farms (see [5]).

Figure 2 shows the overall architecture of the LCFG system:

- The configuration of the entire site is described in a set of declarative *LCFG sources files*, held on a master server. These source files are managed by many different people and describe various different *aspects* of the overall configuration, such as "a web server" or a "laptop", or a "student machine".
- The *LCFG compiler* monitors changes to the sources files and recompiles the source for any nodes whose aspects have changed.
- The result of the compilation is one XML *profile* for each node. The profile defines explicitly all the configuration parameters (called *resources*) for a particular node. This includes the list of software packages, as well as the values to be used in all the various configuration files. Given a repository of software packages, the information in the profile is sufficient to completely reconstruct the node from scratch; LCFG can install a new machine from the bare metal, or clone existing ones, using just a profile, and an RPM repository.
- Client nodes receive a simple notification when their profile has changed,[1] and they collect their new profile using HTTP(S) from a standard web server.
- The client includes a set of *component* scripts, each responsible for a self-contained subsystem, such as inetd, or kdm. Components are

---

[1]Clients also poll the server in case the notification is lost.
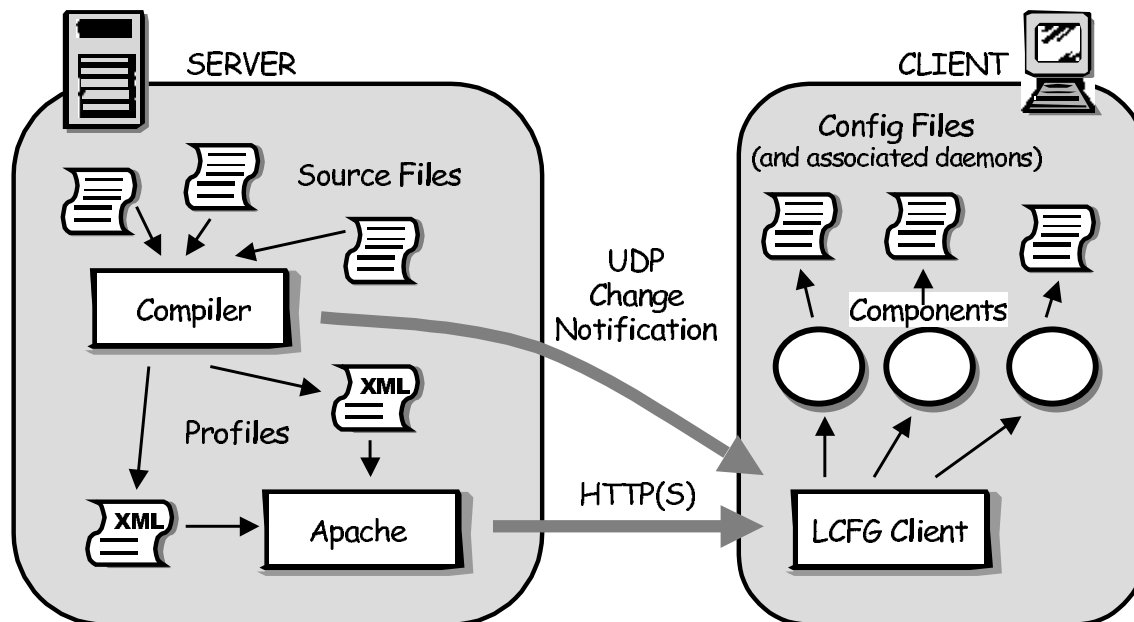

Figure 2: The LCFG architecture.

called whenever any of their resources are changed, and they are responsible for recreating the configuration files that they manage, and taking any other necessary action, such as restarting daemons. Note that each configuration file is managed only by a single component, avoiding any problems with conflicting changes and ordering of updates.

- The LCFG implementation also contains a simple monitoring mechanism (not shown in the diagram) that returns component status information to the server for display on a status page.

Note that there is no one-to-one correspondence between the source files and the profiles, nor between the profiles and the components; source files represent logical aspects of the site configuration,[2] profiles represent individual node configurations, and components manage particular subsystems of a host.

The architecture of LCFG has many well-recognized advantages, which are described more fully in the references, and the basic principles have been adopted for other systems such as [12]. In particular, the single source of configuration information allows a complete site to be reconstructed from scratch. This means that the complete configuration information is always available, and configurations can be validated before deployment by checking the source files.

However, certain configuration information may only be available on the client node; for example, a roaming laptop might need to define a new network configuration while disconnected from the main server; or some information may be obtained from a dynamic source such as DHCP, or DNS SRV records. LCFG includes a mechanism known as *contexts* for handling simple inclusion of some configuration parameters from other sources. This is sufficient to support the above examples, but inadequate for more extensive dynamic reconfiguration.

SmartFrog

SmartFrog is a distributed service and resource configuration engine designed to install and manage complex services spread over a number of computing nodes and other resources. It has been designed to handle the need for distributed sequencing and synchronization of configuration actions, as well as coping with the complexities introduced by the dynamism inherent in such large distributed environments, such as those introduced by partial system failure and communication problems.

The SmartFrog system consists of a number of aspects:

- A declarative description notation for defining desired configuration states, service life-cycles and dependencies between the various service components, including the work-flows to carry out the required changes.

The notation provides a number of features that make it specifically useful for its purpose. It allows the definition of complex structured data and dependencies, with validation predicates to provide rich correctness criteria for their use and modification. The data is defined through the use of templates, with flexible operators to combine and modify the templates to create the final desired form for instantiation.

This process may be linked to database queries or to active run-time discovery services to provide templates with late-bound and dynamic configuration models.

- A component model defining how configurator components – those which carry out the various configuration and other management tasks on the resources and services – are to be implemented. These may then be deployed and managed by the SmartFrog systems as part of the configuration work-flows.

A number of useful pre-defined components are provided as part of the management framework, to support aspects such as resource and service discovery, service failure detection, and script and command execution.

- A distributed deployment and management runtime environment which uses the descriptions and component definitions to orchestrate the work-flows to achieve and maintain the desired state.

Unlike many configuration systems, the environment does not merely support a run-once installation model of configuration – it supports the use and description of persistent components that monitor service state and take appropriate corrective action to achieve continual service availability or other closed-loop control aspects such as ensuring service-level guarantees.

In addition, there is no central point of control, no point to act as a bottle neck in the system. Work-flows are fully distributed and not driven by a central point. Hooks are available to track the progress of these work-flows, and to locate the various management components that are started as a consequence. Thus a rich set of tools may be developed that help in tracing, monitoring and managing the SmartFrog run-time system.

The SmartFrog system provides a security framework to ensure that all configuration actions are valid and authorized by an appropriate authority. The model supports a number of separate security domains, thus ensuring partitioning of responsibility and limiting accidental interaction between these domains. All configuration descriptions and configuration component code must be signed, and these signatures are checked at all points of the configuration process to ensure the integrity of the service configuration process.

---

[2]Although individual nodes also have node-specific source files

The SmartFrog system lacks a number of features that are necessary in a complete configuration system, and which are largely supplied by the integration with LCFG.

The first of these is that SmartFrog assumes that the underlying resources are already running, complete with their OS image. It provides no help in taking a node from bare metal to running system. Smart-Frog starts from the assumption that a node is booted from one of a small set of minimal images at which point a SmartFrog system could configure the various services. LCFG provides the capability to carry out this bootstrap phase.

The second is that SmartFrog is not currently a complete solution; it is a framework for building such solutions. For example, it does not contain a repository for configuration descriptions, nor does it enforce any specific way in which configuration descriptions are to be triggered – these may be triggered by external entities (such as the LCFG system) or by configuration components executing within the SmartFrog framework.

Finally, the SmartFrog framework has yet to be provided with a large collection of service-specific configuration components – such as ones for configuring DNS, DHCP, printers and print queues, and so on. LCFG, however, has been developed over many years to provide precisely this collection of components. A good integration that allows SmartFrog components to wrap and use those of LCFG would provide the best of both worlds.

Dynamic Reconfiguration

There has recently been a growing recognition that computer systems need to support more *autonomic* reconfiguration (for example, [14]) if we are to build ever-larger and more complex systems with an acceptable level of reliability. This requires automatic reconfiguration, *not just of individual nodes*, but of the higher-level roles and interconnections between the nodes. For example, if a server goes down, it should be possible to reconfigure another node to take over this function, and to redirect all the clients to this new server. This is possible with the current LCFG; some automatic tool could simply make the appropriate changes to the LCFG source files, and the whole network could be restructured. However, this involves a large feedback loop via the central configuration server. This provides a single point of failure and a centralized architecture which is inherently unsuitable for very large-scale dynamic systems. We would like to see a much more distributed system where the central server could define a high level policy, and small clusters of nodes could agree, and change, the details of their configuration autonomously within the limits defined by the central policy.

In addition to the fault-tolerance example mentioned above, load-balancing is another case where we would like to make transient, autonomous configuration

changes that do not really represent fundamental changes to the static configuration of the fabric; for example, we might want to stop and start additional web servers on a number of nodes to match the demand. The central configuration should define the set of eligible nodes, but we probably do not want to change the central configuration specification every time the load changes.

There is also one less obvious advantage in devolving the detailed configuration decisions to some autonomic agent; at present, users (sysadmins) are forced to specify explicit configuration parameters, when very often, they only need to specify a more general constraint; for example it might be necessary to specify "Node X runs a DHCP server," when all that is really required is "There should be one DHCP server somewhere on this network segment." This unnecessary explicitness means that the compiler is often unable to resolve conflicts between different aspects, and manual intervention is required; for example, when somebody else removes "Node X."

Previous Work

Reference [6] is a report from the GridWeaver project that includes a thorough survey of existing system configuration tools, together with an attempt to classify common features and different approaches. This report includes a comprehensive list of references to other system configuration tools which are not reproduced here. Very few of these tools even support a clear declarative description of the desired configuration state, and none provide the ability to specify high-level policy about the configuration of a fabric together with a mechanism to enforce it.

Most people will however, be familiar with a number of specific tools that do provide dynamic reconfiguration according to central policy; for example, DHCP dynamically configures IP network addresses within the range specified by the policy embedded in the server configuration. There is currently much interest in more dynamic configuration of network parameters, for example the IETF ZeroConf [17] working group aims to:

- Allocate addresses without a DHCP server.
- Translate between names and IP addresses without a DNS server.
- Find services, like printers, without a directory server.
- Allocate IP Multicast addresses without a MAD-CAP server.

However, large sites will almost certainly want to define the policy within which these autonomous tools operate.

At the opposite end of the scale, there has been some work on dynamic configuration of specific services, particularly web services. Poyner's paper [19] is a good example that describes the use of a Service Location Protocol (SLP) and a centrally defined policy to dynamically configure web services for load-balancing and fault-tolerance.

All the above examples are application-specific implementations, and we are not aware of any attempt to integrate a generic facility for autonomous reconfiguration into a general-purpose system configuration framework. The following sections describes how LCFG and SmartFrog have been combined to construct an experimental framework that does provide this ability to apply different policies and autonomous techniques to arbitrary aspects of a system configuration.

Combining SmartFrog and LCFG

The integration of LCFG and SmartFrog has been achieved in the following way (this is shown diagrammatically in Figure 3):

- The SmartFrog framework has been wrapped as an RPM that can be installed on any node using the existing LCFG software installation components.
- An LCFG component has been developed to manage the SmartFrog Daemon running on a node. This component has two main purposes:
  1. To control the life cycle of the SmartFrog Daemon (typically it will be started at boot time).
  2. To control the list of deployed SmartFrog components, and their configuration information. This allows the central LCFG repository to control the *general policy* under which the SmartFrog framework runs.
- Once deployed, a SmartFrog component can take responsibility for the management of any part of a fabric. However, SmartFrog does not have the extensive range of components that LCFG provides for managing the various aspects of system configuration (apache, fstab entries, etc.). A generic SmartFrog → LCFG

adaptor component has been developed that allows the SmartFrog framework to interact and control any LCFG component. This adaptor allows a SmartFrog component to act as a proxy for an underlying LCFG component. This gives the SmartFrog framework access to all the configuration capabilities of LCFG.

The SmartFrog framework makes it easy for SmartFrog components to perform peer-to-peer interactions with each other. With the combined LCFG and SmartFrog framework these peer-to-peer interactions can lead to re-configurations of the base fabric set-up by the central LCFG server. There are various peer-to-peer mechanism built into SmartFrog:

- Service Location Protocols. These allow SmartFrog components to automatically discover each other without the need for any explicit configuration to link them.
- Partition Membership Protocols. These allow a number of SmartFrog components to declare themselves as part of a group and have the framework automatically elect a leader within the group. If the leader fails, a new leader will be elected in its place. This is typically used to provide fail-over for critical services such as DHCP (The DHCP example is discussed in more detail later).
- Java Remote Method Invocation (RMI). The SmartFrog framework provides look-up services that allow any component to locate any other component on the network by name. Java RMI then allows these components to interact, exchange information and make decisions about required fabric re-configurations.

See the two next sections for a number of examples illustrating the practical use of these concepts.
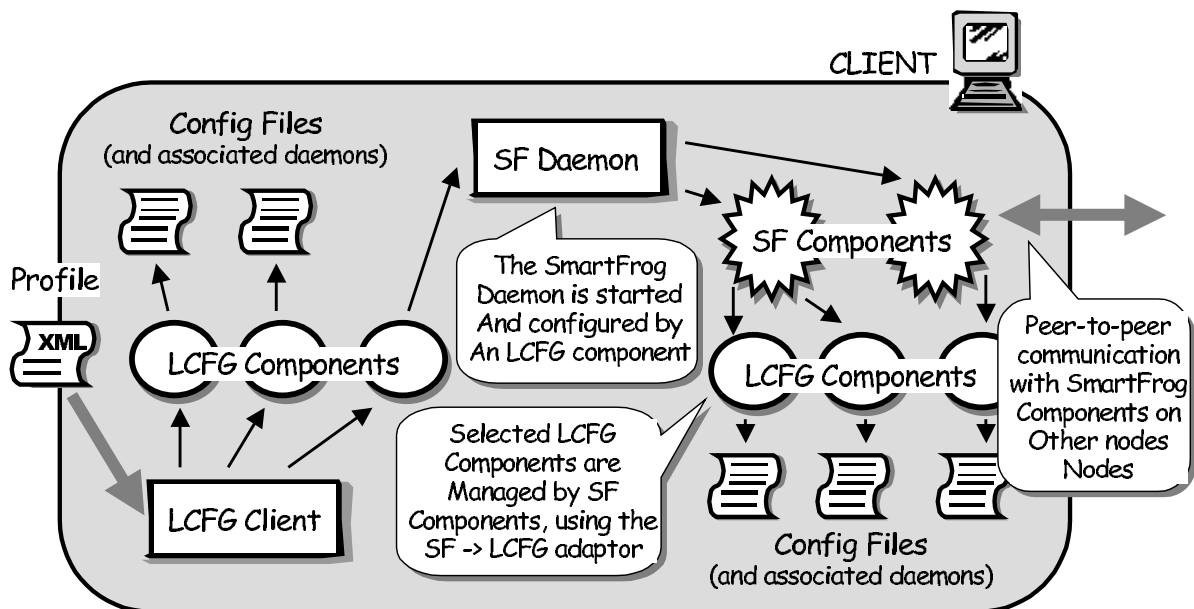

Figure 3: The SmartFrog/LCFG integration architecture.

Example Applications

The importance of the move from single node configuration to dynamic multi-node coordinated configuration can be illustrated by the use of a couple of examples. The two examples presented here consider different aspects of dynamic configuration: the first uses the underlying framework mechanisms to provide service reliability and failure recovery, the second examines the use of discovery for automatically and dynamically adjusting to service location changes.

Service Reliability

A scenario that frequently occurs is that of a service requiring a minimum number of daemons to exist on a collection of servers thereby ensuring a specific service reliability level. So, for example, it may be desirable for there to be at least two instances of a DHCP service on a specified collection of servers. This is relatively easily described: a configuration description would state which servers should hold an instance of the DHCP daemon. Descriptions of this kind would be validated to ensure that two are defined.

However server failures do occur, and it is necessary that the failure of a server containing such a daemon results in the automated re-deployment of the "spare" daemon onto another server thus maintaining the guaranteed service level.

The configuration problem can be described as follows: it would be best if the configuration description could be provided as a set of constraints regarding service replication over a collection of independent nodes, rather than a fixed static mapping of daemons to servers. These constraints should be maintained without needing to define a new static association.

Consider the following base configuration. Each server of a set of servers is configured with two components: a group membership component and a configuration policy engine.

A group membership component is one that uses a network protocol to decide which of a possible collection of such components (each representing their server) are healthy and able to run one or more of the daemons. This protocol must ensure that all servers that are part of this collection agree on its members. From this information a leader may easily be elected.

Such protocols are known as group membership and leadership election protocols, and the SmartFrog framework contains components that implement such a protocol. Note that the important difference between such protocols and simple discovery protocols is the guarantees of consistency of the information at all servers.

The policy component is only activated on the elected leader and, when given the policy (i.e., constraints referring to the number of daemon replicas), allocates daemons to servers in the group so as to balance load whilst maintaining the constraints defined in the policy.

If a node should fail, this is discovered by the group membership protocol and notified to the policy component, which in turn reallocates the service daemons as required to the surviving servers. If the leader should fail, this will be noticed by the whole group membership and a new leader will be elected. This component will ensure that the service daemons are validly distributed to satisfy the given policy.
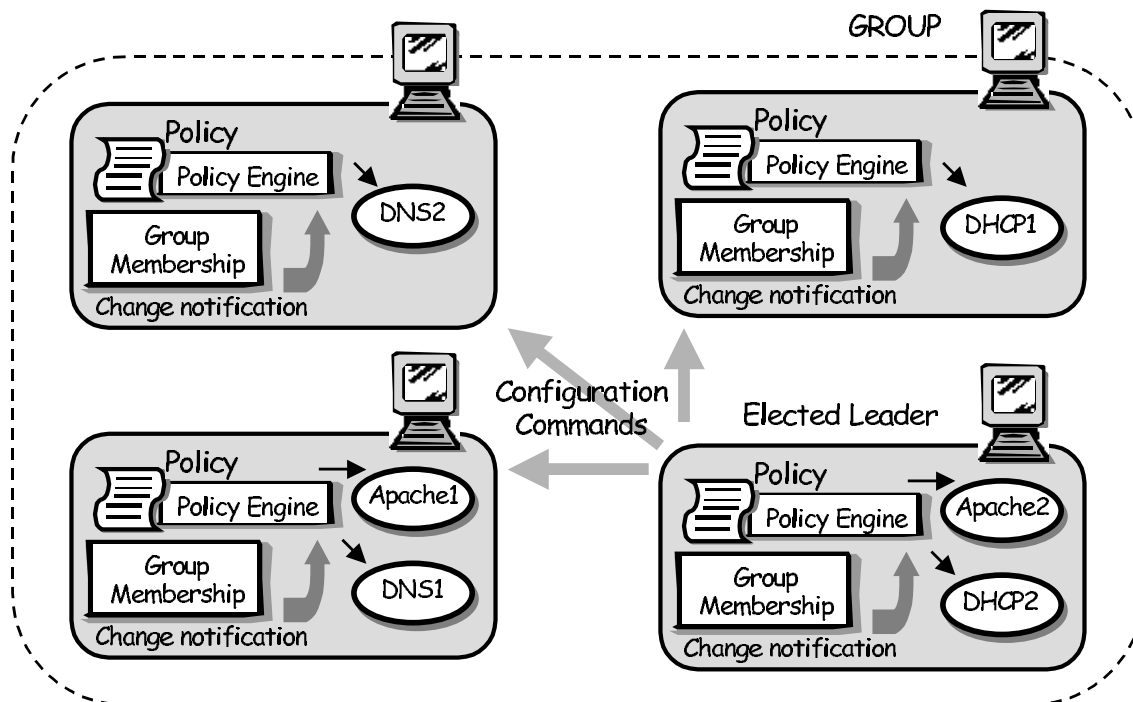

Figure 4: Service reliability.

SmartFrog provides the ability to dynamically describe and manage the configuration requests for the daemons, as well as providing the core components to handle the group communication. LCFG provides the ability to configure the initial collection of servers with the SmartFrog infrastructure, the initial components and, if necessary, the policy description. It also provides the low level components to configure services such as DHCP, DNS, printing, e-mail, and so on that will be triggered by SmartFrog as services are moved around between the servers. Figure 4 illustrates these concepts.

Service Location

The second scenario consists of reconfiguring a system as services move, using a simple service location protocol such as the IETF protocol [18] described in RFC 2165.

A set of Linux file servers offer a set of possibly replicated read-only file systems to a very large collection of Linux client machines via NFS. Each client may require its own unique set of mounts selected from this possible set. Furthermore each file system may be offered by a number of different file servers, with the servers allocated so as to satisfy requirements for reliability, load-balancing of read-requests and storage capacity.

The configuration problem is as follows: although the overall configuration specification for the system may contain the mapping between file-system and server, plus each client's requirements for mounting the various file-systems, changes to the allocation of file-systems to servers may result in many thousands of updates to client machines. These updates would be to modify the automounter settings to mount the correct file servers.

Unfortunately, this is not best handled by pushing these changes to the client machines from some central point as this provides limited scalability and dynamism.[3] A better approach might be to configure a service location component in every client, and a service advertising component into every file server and allowing the distributed service location protocols to resolve the binding between them.

Thus a server would be configured to advertise its own file systems. A client would be told which file systems to locate and prepare an automounter entry for this. Any change of association between server and file-system is then only made on the server and the clients "discover" the new bindings through the location protocols. If more that one server offers access to a specific file system, a number of options exist. A server could advertise itself as the preferred server, in which case the client would select this one in preference. If all servers are equal, a random choice could be made by the client thus spreading load amongst the various servers.

Finally, if a server disappears or communication problems exist between the client and the server (this could be monitored by the client configuration components, for example by scanning log files) a binding to an alternative server could be made. Thus a local decision can be made to resolve a locally identified, localized problem.

Within the combined LCFG/SmartFrog environment, this would be carried out in the following way. LCFG contains the basic components to handle NFS

---

[3]Using LDAP or NIS for the maps would create a single point of failure in the master database server, and would not solve the problem of updating the maps when a server fails or appears.
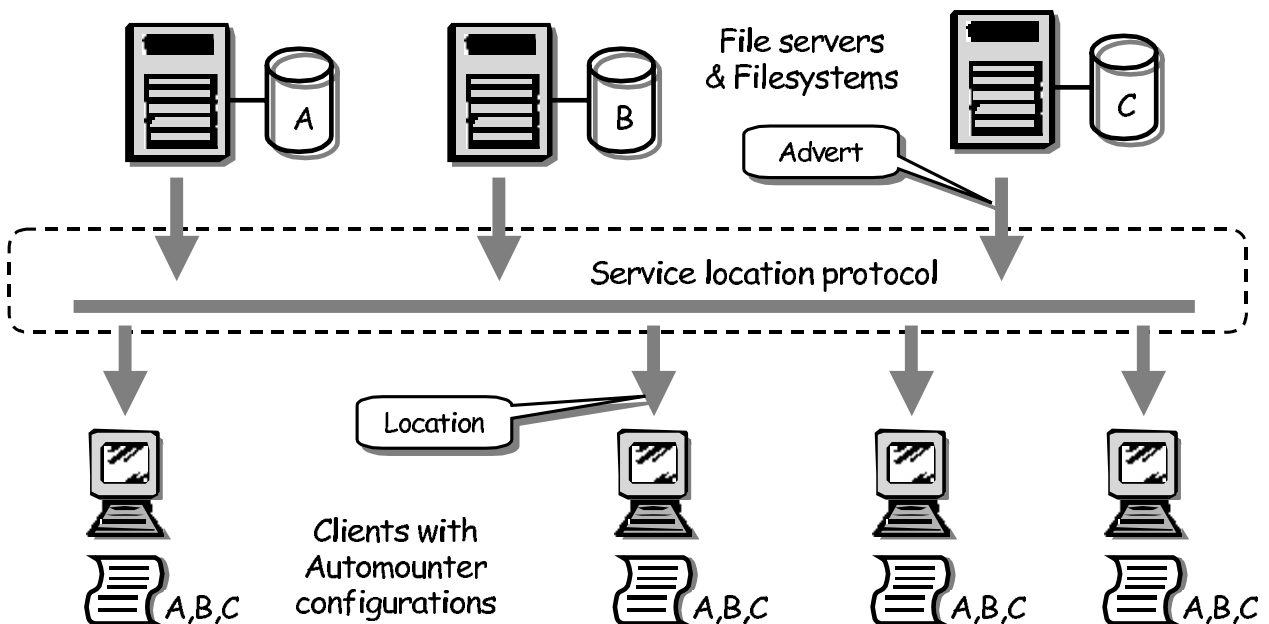

Figure 5: Service location.

servers, configuring the automounter, and so on. SmartFrog provides the appropriate service location and advertising components, using a encapsulated implementation of the SLP protocol. Configuration descriptions would be created that define, for each server and client, the set of file systems they hold, or require. The SmartFrog system would then dynamically instantiate the completed service, using the LCFG components for the specific node configurations. Figure 6 illustrates these concepts.

The GPrint Demonstrator

The GPrint demonstrator is a complete, self-contained cluster that provides a robust printing service via an OGSA interface. This has been developed as part of the GridWeaver project and is described more fully in [10]. It illustrates how the combined LCFG and SmartFrog frameworks can be used to provide a robust printing system that can automatically adjust for print server or printer failures. A short video is available [20] on the web, which demonstrates the GPrint system in action.

The underlying printing system is based on LPRng [3]. The key goals of the system are:

- **Fault tolerance**: no single point of failure combined with autonomic reconfiguration when any part of the system fails.

- **Minimal management**: e.g., to deploy a new print server it should simply be a matter of plugging the node into the network and deploying a print server configuration description to the node.
- Illustrate how the configuration system can integrate with the Globus Toolkit and OGSA standards [15] to provide a robust grid enabled application.

The design of the system is illustrated in Figure 5 and explained further below:
- LCFG provides components to manage the LPRng printing daemon. These components have been used though they are managed by Smart-Frog using the SmartFrog → LCFG adaptor.
- A SmartFrog component has been developed to represent a print server. Using LCFG, this component can be deployed on any node. Once deployed, the component advertises the print server using SLP. The rest of the printing system listens for these advertisements and deploys the printing services across the available resources.
- A SmartFrog component has been developed that can scan a range of network addresses for printers. Once a printer is found, a proxy SmartFrog printer component is deployed to represent the printer to the printing framework.
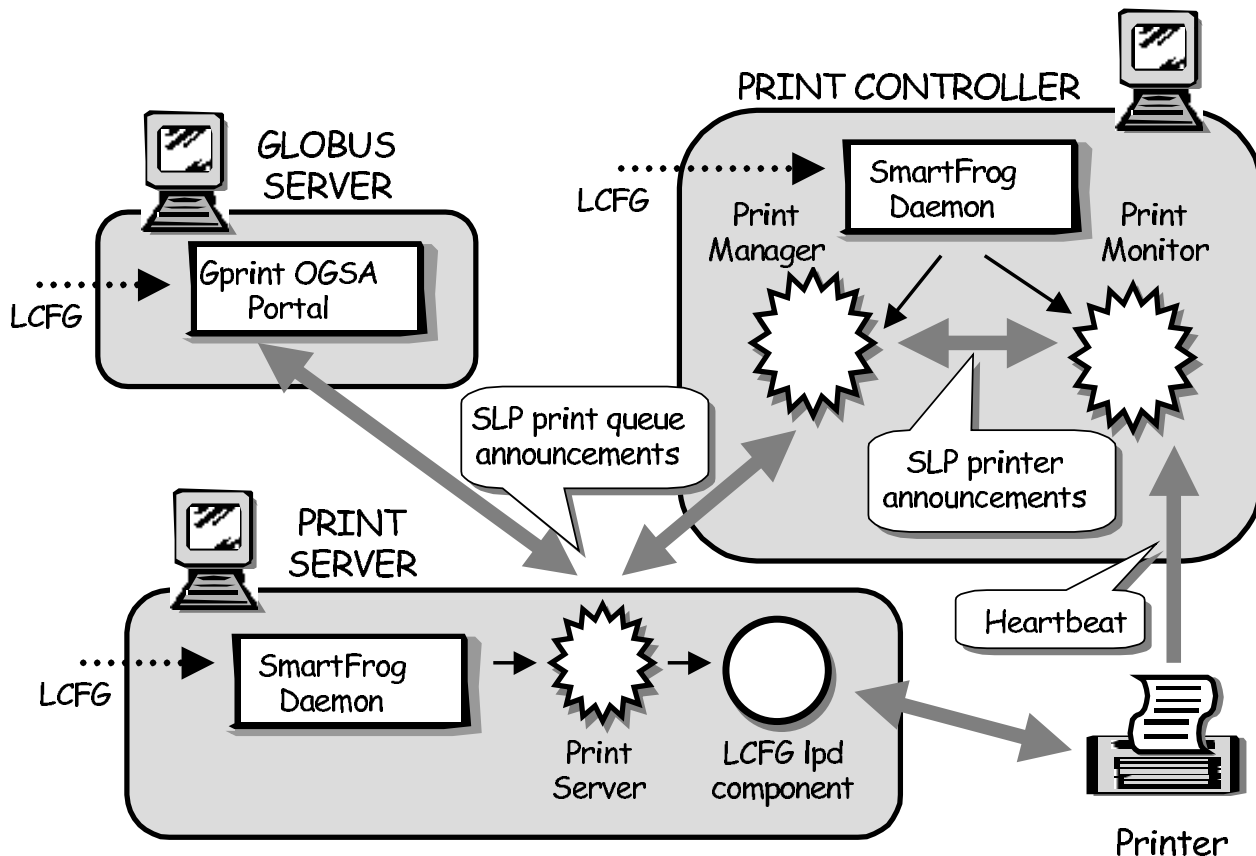

Figure 6: The GPrint demonstrator.

This component performs heartbeat monitoring of the printer and advertises the printers existence using SLP.

- The brains of the system is a SmartFrog print manager component that listens for the print server and printer announcements. This component is configured with a list of the print queues that are to be made available to the end users of the printing system. It deploys these queues based on the currently available resources and will reallocate the queues if any part of the system fails.

  The SmartFrog leadership election protocols can be used to deploy multiple print server managers on different nodes. Only the currently elected leader will perform any system configurations. The other copies are there to take over if the leader node fails. This prevents the manager component from becoming a single point of failure in the system.

- When the print server manager deploys a print queue, this involves deploying a SmartFrog Queue component onto the appropriate print server node. As well as configuring the queue, so that it can print, this component advertises the queue's existence using the SLP protocol. This allows any interested network components to be notified of the current printing configuration. In the full GPrint system, an OGSA portal has been developed which allows print jobs to be submitted through a Grid style interface. The GPrint portal listens for the print queue announcements and advertises the available print queues through its OGSA interface.

Note that the entire GPrint cluster can be rebuilt from ''bare metal'' machines, using just the configuration specifications and an RPM repository. Likewise, new nodes of any type can easily be installed and incorporated into the cluster.

Conclusions

Future configuration systems will need to incorporate a high degree of autonomy to support the anticipated need to scale and demands for robustness. In practice, this is likely to require several different configuration paradigms, such as explicit specification, or discovery by service location protocol.

We have shown that it is possible to build a configuration framework that allows different paradigms to be incorporated easily, without changes to the components that actually deploy the configuration. The prototype implementation of this framework provides a testbed for experimenting with different configuration paradigms using real, production configurations.

We have also demonstrated that the framework can be used to construct a typical real service with autonomic fault-recovery. The service can easily be restructured to support different modes of configuration.

Availability

LCFG software is available under the GPL from www.lcfg.org, although work is currently underway to improve the packaging, and anyone interested in downloading a current version of the software is invited to contact Paul Anderson at dcspaul@inf.ed. ac.uk .

HP is currently working on a public release of the core SmartFrog framework, including source code. Anyone interesting in obtaining this may contact Peter Toft at peter_toft@hp.com .

The GPrint demonstrator is not intended for production use, but code is freely available by contacting the authors.

Author Information

Paul Anderson (dcspaul@inf.ed.ac.uk) is a principal Computing Officer in the School of Informatics at Edinburgh University. He is currently involved in the development of the School's computing infrastructure as well as leading several research projects in large-scale system configuration. He is the original architect of the LCFG configuration framework.

Patrick Goldsack (patrick.goldsack@hp.com) has been with HP Laboratories in Bristol, England since 1987, where he has worked on a variety of research projects in areas ranging from formal methods, through network monitoring and management techniques, to Grid and utility computing. His research interests are in formal languages and large-scale distributed systems.

Jim Paterson (jpaters2@inf.ed.ac.uk) left the University of Glasgow in 1989 with a Ph.D. in Theoretical Physics. Since then he has worked as a software developer for a number of consultancy firms before taking up his current position as a Research Associate at Edinburgh University. His research has focused on problems in large scale system configuration.

References

[1] *The DataGrid Project*, http://www.datagrid.cnr.it/ .
[2] *LCFG*, http://www.lcfg.org/ .
[3] *LPRng Printing Framework,* http://www.lprng.org .

[4] Anderson, Paul, "Towards a high-level machine configuration system," *Proceedings of the 8th Large Installations Systems Administration (LISA) Conference*, pp. 19-26, Berkeley, CA, Usenix, http://www.lcfg.org/doc/LISA8_Paper.pdf, 1994.

[5] Anderson, Paul, *The Complete Guide to LCFG*, http://www.lcfg.org/doc/guide.pdf, 2003.

[6] Anderson, Paul, George Beckett, Kostas Kavous-sanakis, Guillaume Mecheneau, and Peter Toft, "Technologies for Large-scale Configuration Management," *Technical report, The GridWeaver Project*, http://www.gridweaver.org/WP1/report1.pdf, December, 2002.

[7] Anderson, Paul, George Beckett, Kostas Kavous-sanakis, Guillaume Mecheneau, Peter Toft, and Jim Paterson, "Experiences and Challenges of Large-scale System Configuration," *Technical report, The GridWeaver Project*, http://www.gridweaver.org/WP2/report2.pdf, March, 2003.

[8] Anderson, Paul and Alastair Scobie, "Large Scale Linux Configuration with LCFG," *Proceedings of the Atlanta Linux Showcase*, pp. 363-372, Usenix, Berkeley, CA, http://www.lcfg.org/doc/ALS2000.pdf, 2000.

[9] Anderson, Paul and Alastair Scobie, "LCFG – The Next Generation," *UKUUG Winter Conference*, UKUUG, http://www.lcfg.org/doc/ukuug2002.pdf, 2002.

[10] Beckett, George, Guillaume Mecheneau, and Jim Paterson, "The gprint Demonstrator," *Technical report, The GridWeaver Project* http://www.grid-weaver.org/WP4/report4_1.pdf, December, 2002.

[11] Burgess, Mark, "Cfengine: A Site Configuration Engine," *USENIX Computing Systems*, Vol. 8, Num. 3, http://www.iu.hioslo.no/~mark/research/cfarticle/cfarticle.html, 1995.

[12] Cons Lionel and Piotr Poznaski, "Pan: A High Level Configuration Language," *Proceedings of the 16th Large Installations Systems Administration (LISA) Conference*, Berkeley, CA, Usenix, http://www.usenix.org/events/lisa02/tech/full_papers/cons/cons.pdf, 2002.

[13] Edinburgh School of Informatics, EPCC and HP Labs, *The GridWeaver Project*, http://www.gridweaver.org.

[14] Jeff Kephart, et al., "Technology challenges of autonomic computing," *Technical Report, IBM Academy of Technology Study*, November, 2002.

[15] The Globus Project, *OGSA*, http://www.globus.org/ogsa/.

[16] Goldsack, Patrick, "SmartFrog: Configuration, Ignition and Management of Distributed Applications," *Technical report, HP Research Labs*, http://www-uk.hpl.hp.com/smartfrog.

[17] IETF ZeroConf Working Group, *ZeroConf*, http://www.zeroconf.org/.

[18] IETF, *Service Location Protocol (svrloc)*, http://www.ietf.org/html.charters/svrloc-charter.html.

[19] Poyner, Todd, "Automating Infrastructure Composition for Internet Services," *Proceedings of the 2001 Large Installations Systems Administration (LISA) Conference*, Usenix, Berkeley, CA, http://www.usenix.org/events/lisa2001/tech/full_papers/poynor/poynor.pdf, 2001.

[20] Toft, Peter, *GridWeaver, The Movie*, http://boombox.ucs.ed.ac.uk/ramgen/informatics/gridweaver.rm and http://boombox.ucs.ed.ac.uk/ramgen/informatics/gridweaver-v8.rm, 2003.