Towards a High-Level Machine Configuration System

Paul Anderson – University of Edinburgh

ABSTRACT

This paper presents a machine configuration system which stores all configuration parameters in a central ''database''. The system is *dynamic* in the sense that machines reconfigure themselves to reflect any changes in the database whenever they are rebooted. The use of a central database allows configurations to be validated, and correct configurations to be automatically generated from policy rules and high-level descriptions of the network. A permanent record of every machine configuration is always available and the system is extensible to handle configuration of new subsystems in a modular way. The paper includes a review of previously published work and common techniques for *cloning* and configuring workstations.

Introduction

When a new machine is installed, it will rarely be used with the default configuration supplied by the vendor of the operating system. The partitioning and allocation of space on the disks, the software packages to be carried, and the network name and address are typical *configuration parameters* that will be set differently by different sites and for different machines at the same site. In addition to these basic parameters, most large sites will require a more extensive customisation of the basic system, for example running additional or replacement daemon processes such as time synchronisation.

Most vendors provide some kind of installation procedure which allows the basic configuration parameters to be set. However, in a typical large site, these procedures are nearly always inadequate for one or more of the following reasons:

- The procedures cover only the vendor-supplied software and are not extensible to cover local and third-party software.
- The interface to the procedures is often a GUI and cannot easily be automated for handling large numbers of systems.
- The procedures are not complete, and further manual operations (for example, `crontab`), or additional hand-editing of configuration files (for example, `inetd.conf`), are required to completely configure the machine.
- The configuration information is stored on the machine itself so that it must be re-entered whenever the machine is re-installed, and it is unavailable for inspection when the machine is down.
- The procedures are highly vendor-specific and are not appropriate for use in a heterogeneous environment.

Sites with a small number of machines, or simple configuration requirements, sometimes use only the vendor-supplied procedures, but this means that machine upgrades or installations require considerable manual intervention. Large sites will usually have developed their own procedures to help overcome some of these problems, and the following section surveys some of the techniques that have been used.

The remainder of the paper describes a configuration procedure that has been developed for use in the Computer Science Department at Edinburgh University. This stores complete machine configuration information in a central ''database'', allowing configurations to be validated and automatically generated. The system is also modular so that new subsystems can be added independently to the configuration procedure.

Background

Most vendor-supplied installation and configuration tools suffer from all of the problems listed in the previous section. In many cases, attempts to simplify installation for small sites (for example, graphical user interfaces) have caused further difficulties for large sites. Even where some provision has been made for large-scale automation (such as Sun *auto-install*[1]), the configuration process is still inadequate for the other reasons given above.

The most common technique for dealing with a large number of machines is *cloning*. Cloning procedures are not normally supplied by the vendor, but different systems have evolved at many large sites (for example, Ohio State University[2]), all sharing similar characteristics. A single *template* file-system is hand-crafted with the site-specific configuration information and replicated directly to create a new machine. Clearly, such a pure cloning process is only sufficient if there are no machine-specific configuration parameters, and every machine on the site has an an identical basic file-system (or there are a small number of categories). This approach has been taken in some cases, such as the Athena [3] system, but it usually requires unacceptable modifications to the vendor's base operating system.

Various schemes have been used for applying machine-specific changes to the template after (or during) the cloning operation; for example, the above Ohio scheme, `typecast [4]` and `mkserv[5].` These are adequate for environments where the configurations are largely static and similar. However, they can become unwieldy when there is a wide variation in the required configurations and/or frequent changes. It can often be difficult to determine the configuration that is actually being applied to an individual machine; in some cases, this information might not exist explicitly[1]; in other cases, it might exist in a wide range of different files and formats. The lack of modularity in the configuration process also makes it difficult for different people to maintain the configuration of separate subsystems, and changing the configuration of an existing machine is usually difficult.

Storing the machine-specific configuration information explicitly in some external database (for example, `sad[6])` is a major improvement, since the configuration of a particular machine is always clear and the information is always accessible, even when the machine is down. There is still the option of using procedural rules to generate certain configuration parameters[2] but the rules are evaluated before the machine is actually configured and the results of the evaluation are visible explicitly in the database.

The information from such a database can be used during the cloning process to control the creation of the file-systems when the machine is being built. In this case, the machine-specific characteristics are hard-wired into the file-system and the database information is no longer required for the running of the machine (a *static* configuration). Alternatively, all machines can be created as pure clones and the configuration information can be read *dynamically* from the database as the machines are running (usually at boot time). If the configuration information is used in a static way, it is difficult to change without completely re-cloning the system, but the machine is not dependent on the availability of the central database, and no configuration procedures need to be run at boot time. Dynamic configuration requires special configuration procedures (usually run at boot time) and the machine is dependent on the existence of the central database, but it does allow changes in the database to be reflected immediately in the actual machine configuration.

[1]For example, a particular configuration parameter might be generated ''on the fly'' at installation time by a script which implements some kind of policy rule.

[2]For example, there may be a rule of the form ''machines belonging the research group always carry GNU Emacs''.

A purely dynamic system is normally impractical for several reasons:

- Configuration of hardware-related parameters such as disk partitioning is not possible on a running system where the disks contain live data.
- Configuration of very low level system software (such as basic networking) is difficult because the machine normally needs the network to be available before it can access the configuration database.

However, the ''rotting'' of static systems and the difficulty of identifying the configuration state of a particular machine can lead to many problems which make a dynamic system attractive.

Many vendors are now moving towards dynamic configuration systems based on object-oriented technology. The Tivoli ''Management Environment'', for example, is an object-oriented product which is available on several platforms. This provides a central configuration database and a ''framework'' into which objects can be slotted to control the various subsystems in a uniform way. Hopefully, standards will develop, and become adopted, so that multiple vendors (and users) can construct objects which inter-operate across heterogeneous systems. Although this provides the most promising future direction for system configuration, most vendors do not currently supply such software as part of their standard operating system package, and current implementations may be too expensive and/or inflexible for many sites.

A Simple Dynamic Implementation

The Computer Science Department at Edinburgh University runs a network of 300-400 workstations with about 2000 users. System administration tools from the department are often adopted on a wider scale throughout the University. At present, these machines are mostly Suns (currently being upgraded to Solaris 2) and X terminals, but the ability to integrate systems from different vendors is considered very important and DEC, HP and SGI systems have all previously been integrated into the network. Particularly within research groups, such as the LFCS[3], systems change rapidly and machine configurations are very diverse, so it is important to have a sufficiently flexible infrastructure to support this type of environment.

The *lcfg* (''local configuration'') system [7] now being used in the Computer Science Department is a mainly dynamic system with a small amount of static configuration for the hardware and low-level

[3]Laboratory for Foundations of Computer Science.

parameters. All information that is necessary to distinguish one machine from another is contained in the central database and every machine can be rebuilt or duplicated using just the information from the database together with the generic system software[4]. Only Suns are currently being configured with `lcfg`, but it is intended that the system be portable, presenting a uniform interface to the configuration process across different platforms. The static part of the configuration which interfaces with Sun auto-install is the only part of the system which is expected to be significantly different on different platforms.

The static part of the configuration occurs when a machine is installed. Information is read from the database and used to construct auto-install configuration files determining the type of machine, the layout of the disks, the base software configuration, and other static parameters. When the machine reboots for the first time after an installation, a further script performs any remaining static configuration. This might include addition of clients or loading of additional software across the network. All machines can be installed entirely automatically, complete with all the necessary local customisation, simply be creating the database entries and booting the system from an install server.

Every time the machine boots, a script reads the configuration database to determine the *subsystems* that should be configured on that machine. This executes a script for each subsystem (for example, DNS or `xntp`) which consults the database for relevant parameters and dynamically configures the subsystem accordingly. New subsystems can therefore be incorporated into the configuration process simply by adding their names to the database entry for a specified machine. The dynamic configuration allows machines to be reconfigured very quickly to adapt to changing requirements, or work around failed hardware.

The Configuration Database

The configuration scripts use common routines to consult the database for *resources* of the form

host.*subsystem*.*attribute* = *value*

In theory any database could be used to hold these resources and any mechanism could be used to distribute them to the client machines. A large relational database might be a useful tool for extracting information about machine configurations, and making complicated changes to groups of machines, but it is not strictly necessary and, at present, a simple flat file is used for each machine. The resources are distributed and supplied to the client machines using NIS[8]. NIS is not ideal for this purpose, since it involves propagation of the entire database every time a single change is made, and all system software below the level of NIS must be statically configured. We hope to eventually develop a special protocol that operates at a lower level, but NIS is currently proving adequate as a resilient method of supplying machines with the necessary resources.

The information in the source files is deliberately of a very low level. As described later, the eventual aim is to generate this information automatically from a higher level description of the machine and its relationship to other machines in the network. At present, the files are edited by hand and passed through the C preprocessor which allows some degree of structure to be introduced, and machines with similar configurations to share common blocks of resources. A total of about 400 different resources are available for configuration of various different subsystems, but many of these will nearly always be used in their default values and a typical large server requires about 70-100 resources to fully describe the configuration. Clients usually require about half this number, and the use of the C preprocessor reduces the configuration description even further (some examples are given in the appendix).

Independent processes can very easily extract information from the database and one important application of this is to validate the consistency of the resources. A simple Perl script scans the resources for a specified machine and performs various consistency checks; the script is continually being extended to identify the most common configuration errors and this allows many problems to be detected before the machine installation has started. Since information is available on all machines, inter-machine problems can be located that might not normally be detected until a much later stage. In particular, it is possible to check before removing a machine from the network, that all dependencies on that machine have been removed. Not all of these dependencies are immediately obvious; for example, every ethernet segment must include a host supplying `bootparam` service, and removing the last `bootparam` server from an ethernet segment should cause a warning to be generated. Such checks can be used to identify weak points in the network by answering questions, such as ''what happens if a particular server fails''.

Some of the resources are purely informational and are used for administrative purposes (for example, the owner and location of the machine). One interesting application is an experimental *World Wide Web* service which makes information on all machines available over the World Wide Web by automatically querying the database when the page for a particular machine is accessed[5]. The

[4]Obviously backups of any user data are also required.

[5]`http://www.dcs.ed.ac.uk/cgi-bin/hosts/INDEX`

information in the database allows hyper-text links to be generated between clients and their servers, and between personal workstations and the home pages of their owners.

The Configurable Subsystems

Each configurable subsystem on a machine (for example, a printer) is a member of a particular *class* and the configuration for all subsystems in a class is performed by the same *class script*. All the class scripts share a number of common routines and are written in a stylised manner; this allows new classes with simple configuration requirements to be added very easily. A single subsystem called `boot` starts when the system boots. The resource `boot.services` is consulted to determine all the other subsystems that should be configured at boot time and the appropriate class scripts are executed. Provision is also made to execute these scripts manually, or at regular intervals (from `cron`).

There are currently about 30 different classes implemented, of which the following is selection:

- **auth** configures all the authorisation of access to the machine. This controls, for example, the groups of users that are permitted to log in, and the machines to be included in `hosts.equiv` file.
- **amd** controls the `amd` automounter, specifying the `cluster` that is to be used and hence determining the servers from which the various file-systems will be mounted.
- **dns** controls the type of DNS service to be provided and (where appropriate) specifies the servers to be used.
- **www** controls the World Wide Web server.
- **xdm** controls the `xdm` subsystem specifying which X terminals are to be managed and configuring some of the parameters of the login session. A separate subsystem controls the font server.
- **inet** controls the services that are managed by `inetd`, including the access control which is managed by the `tcpd` wrapper program.

The above subsystems run only when the machine boots, and any change in the database resources is not reflected in the corresponding subsystem until the machine is rebooted (or the subsystem is manually restarted). These are mostly one-off configurations (such as `auth`) or daemons which start once and run continuously (such as `www` or `xdm`). Some subsystems need to be run at regular intervals (for example, backups) and the `boot` subsystem can arrange to schedule these to run from `cron`. In particular, a group of processes runs every night to perform any necessary updates to the local file-systems:

- **updatelf** uses `lfu`[9] to update the local file-systems with any changes that have been made to the master copies of locally maintained software. The configuration of this subsystem determines the software packages that are to be carried by the machine.
- **patch** applies any new systems patches that have been installed which are relevant to the machine.
- **update** makes any necessary modifications to files in the root file-system to track the latest static configuration.

Most class scripts also accept additional arguments to stop and restart the subsystem, and to display logging and status information. A client program called `om`, and its associated daemon `omd`, provide a way to execute these additional *methods* remotely, including an authorisation scheme with access control based on the user, the host, the subsystem, and the method. This allows users to be given permission, for example, to stop and restart certain daemons running on their personal workstation. One possibility is that `om` will be extended to understand netgroups of machines, allowing subsystems to be easily restarted on a whole cluster of machines with a single command.

High Level Configuration

One of the most important aspects of machine configuration is to specify the role of a machine within the network. This includes the relationship between a client and the servers which supply various different services. Typically, these will include file services of various types (home directories, program binaries), name service (DNS), time synchronisation (`xntp`), font service and others. If a client and server are configured independently, then there is no guarantee that the configurations are compatible; for example, a client can quite easily be configured to expect file service from a machine which is not exporting the required files, or even from a machine that does not exist! Even within a single machine, there are similar dependencies offering scope for errors when different subsystems are configured using different methods; for example, if a particular machine is to run a World Wide Web server, then the appropriate software must be available on the machine.

Using a common source of configuration information allows most of these dependencies to be checked automatically. However, the low level nature of the raw configuration resources means that production of configuration files is awkward and error prone. Ideally, we would like to describe the relationship between machines at a much higher level and have the low level configuration information generated automatically. For example:

- Machine A is the name server for the research group.
- Machine B is a member of the research group.
- Machine C is a member of the research group.

From the above specification, it is possible to generate all the necessary low level configuration information to load the name-server software, and start the name-server subsystem, on machine A, and configure the other machines to act as clients of this machine. An error (or at least a warning) would be expected for any machines which did not have a name-server.

The simple example given above can be accomplished quite easily, using features of the C preprocessor, with the existing implementation. Changing machine A to some other machine should cause the software and the daemon to be transferred to the other machine, and clients to change their `resolv.conf` files to point to the new server.

In addition to the essential rules, like the name-server example above, it is also very useful to be able to specify policy rules in a similarly explicit manner. For example:

- Students are not allowed to log in to personal workstations of staff members.
- File-servers which are updating local file-systems during the night should do so at different times to avoid network congestion.

Such policy rules are frequently contravened in practice because they are not critical to the operation of the system and mistakes can easily go unnoticed. Using the rules to actually generate the machine configuration guarantees that they will be enforced.

As the rules and their interactions become more complex, the need for a special-purpose *configuration language* to replace the C preprocessor quickly becomes apparent. Designing such a language [10] is not easy for several reasons; it must be able to express high-level rules in a clear, explicit way, but be capable of generating low level configuration information from these rules. Since the configuration subsystems must be extensible, the language itself must be extensible so that new rules can be added to control new subsystems, or new features of existing subsystems. Possible designs for such a language are currently under investigation.

Conclusions & Further Work

The use of a dynamic configuration system storing parameters in a central database has been a big improvement over the previous static system. In particular:

- The ease with which configurations can be changed, and machines can be completely rebuilt, means that machine configurations do not ''rot'' and are always up-to-date.
- New subsystems can easily be introduced and configured onto existing machines without interfering with other subsystems on the machine.
- The ability to validate and examine explicit machine configurations from the database has reduced the number of errors that are caused, for example, by forgetting some dependency when removing a server.
- Since the machines automatically reflect the configuration in the database, it is possible to have some confidence that policies specified in configuration rules are actually being enforced on the machines. This provides an improvement, for example, in security.

Disadvantages include the longer time required to boot a machine and the difficulty of manually creating correct low-level configuration information.

The ability to specify configurations and policies at a much higher level is a very useful facility. The best way in which to implement and exploit this possibility is an area for further investigation. In the short term, incorporation of further subsystems, porting to other platforms, and improvements to the mechanism for storing and distributing the resources are likely areas of future work.

Availability

Copies of this paper and associated technical reports are available via WWW from `http://www.dcs.ed.ac.uk/staff/paul` or `pub/paul/papers` on `ftp.dcs.ed.ac.uk` (ftp).

Acknowledgements

Thanks to all the systems staff of the Computer Science Department for long discussions on the design of the configuration system and for suffering all the machines with broken configurations during the development and testing.

Author Information

Paul Anderson is a graduate in pure mathematics. He has taught computer science and managed software development before becoming involved in systems administration. He is currently employed as Systems Development Manager with the Laboratory for Foundations of Computer Science, where he is responsible for the research laboratory's network. He is also working with other system managers to develop the computing facilities within the department and the University. Paul can be reached by mail at:
The Laboratory for Foundations of Computer Science
Department of Computer Science
University of Edinburgh
King's Buildings
Edinburgh EH8 3JZ
U.K.
His email address is: paul@dcs.ed.ac.uk.

References

1. Sun Microsystems, ''Automatic installation,'' in *Solaris 2.3 system configuration and installation guide*, 1993.
2. George M Jones and Steven M Romig, ''Cloning Customized Hosts (or Customizing Cloned Hosts),'' *Proceedings of the LISA V Conference*, pp. 233-237, Usenix, 1991.
3. Jennifer G Steiner and Danial E Geer, *Network services in the Athena environment,* Project Athena, Massachusetts Institute of Technology, Cambridge, MA 02139.
4. Elizabeth Zwicky, ''Typecast: beyond cloned hosts,'' *Proceedings of the LISA VI Conference*, pp. 73-78, Usenix, 1992.
5. Mark Rosenstein and Ezra Peisach, ''Mkserv – Workstation customization and privatization,'' *Proceedings of the LISA VI Conference*, pp. 89-95, Usenix, 1992.
6. Rick Dipper, ''Management information and decision support tools for Unix systems administration.,'' *Proceedings of UKUUG/SUG Conference*, pp. 143-153, UKUUG, 1993.
7. Paul Anderson, ''Local system configuration for syssies,'' CS-TN-38, Department of Computer Science, University of Edinburgh, Edinburgh, August 1991. Available by anon ftp as file `pub/paul/papers/tn38.ps` from site `ftp.dcs.ed.ac.uk`.
8. Sun Microsystems, ''The Network Information Service,'' in *System and network administration*, pp. 469-511, Sun Microsystems, 1990.
9. Paul Anderson, ''Managing program binaries in a heterogeneous UNIX network,'' *Proceedings of LISA V Conference*, pp. 1-9, Usenix, 1991.
10. Bent Hagemark and Kenneth Zadeck, ''Site – a Language and System for Configuring Many Computers as One Computing Site,'' *Proceedings of the LISA III Conference*, pp. 1-13, Usenix, 1989.

Appendix 1: Configuration for a Simple Server

```
/***************************************************************************
  Staffa
***************************************************************************/

#include <lfcs.h>

/* Resources for information only */

info.type               server
info.location           the machine halls
info.make               Sun
info.model              10/40
info.owner              LFCS
info.memory             16 16 16
info.sno                411m1238
info.hostid             727099f2
info.disks              internal wren
info.disktype_internal  SUN1.05 cyl 2036 alt 2 hd 14 sec 72
info.disksize_internal  1Gb
info.diskdev_internal   c0t3d0
info.disktype_wren      CDC Wren VII 94601-12G cyl 1929 alt 2 hd 15 sec 68
info.disksize_wren      1Gb
info.diskdev_wren       c1t1d0

/* Statically configured resources */

install.system_type     server
install.arch            sun4m
install.client_arch     sun4c sun4m
install.local           B_INSTALL_CONFIG
install.interfaces      le0 qe0
install.hostname_le0    HOSTNAME
install.hostname_qe0    HOSTNAME-j
install.updatelf        true
install.install_server  true
install.filesystems     root swap var usr export local
install.fs_root         c0t3d0s0 32 /
install.fs_swap         c0t3d0s1 64 swap
install.fs_var          c0t3d0s3 64 /var
install.fs_usr          c0t3d0s4 auto /usr
install.fs_install      c0t3d0s7 350 /export/install
install.fs_export       c0t3d0s5 free /export
install.fs_local        c1t2d0s2 all /disk/local

/* Dynamically configured resources */

auth.rootpwd            LFCS_SERVER_PASSWD
auth.users              LFCS_SERVER_USERS
auth.equiv              LFCS_EQUIV
auth.rhosts             LFCS_RHOSTS
amd.cluster             HOSTNAME.dcs.ed.ac.uk
dns.type                server
yp.type                 slave
yp.servers              HOSTNAME
boot.services           SERVER_SERVICES
boot.run                SERVER_RUN
cron.objects            boot
cron.run_boot           0 0 * * *
updatelf.fs             local
updatelf.fs_local       sun4-51 share
updatelf.netgroups      delete copy
updatelf.action_copy    copy
```

```
updatelf.action_delete  delete
nfs.exports             local
nfs.fs_local            /disk/local
nfs.options_local       -o ro=machines
```

Appendix 2: Configuration for a Simple Diskless Client

```
/*********************************************************************
 * Gasker
 *********************************************************************/

#include <lfcs.h>

/* Resources for information only */

info.type               private
info.owner              paul
info.location           1612
info.make               Sun
info.model              Classic
info.sno                302U4308
info.hostid             8001d534

/* Statically configured resources */

install.system_type     client
install.arch            sun4c
install.interfaces      le0
install.hostname_le0    HOSTNAME
install.root            B_SERVER:/export/root/HOSTNAME
install.swap            B_SERVER:/export/swap/HOSTNAME

/* Dynamically configured resources */

mail.root               paul
auth.rootpwd            LFCS_CLIENT_PASSWD
auth.users              LFCS_CLIENT_USERS
auth.equiv              LFCS_EQUIV
auth.rhosts             LFCS_RHOSTS
amd.cluster             B_SERVER.dcs.ed.ac.uk
dns.servers             B_SERVER
cron.objects            boot
cron.run_boot           0 4 * * *
```