



System Installation and Configuration

Paul Anderson

Department of Computer Science
University of Edinburgh
paul@dcs.ed.ac.uk

Introduction



- Installation and configuration of a new machine involves considerably more than installing the vendor's base operating system
- Some method of automating this process is necessary for all but the smallest sites
- This talk presents our experiences with a configuration system developed in the Department of Computer Science at Edinburgh University

- ✍ Apart from configuring the basic network parameters, most hosts have additional services and software which require configuration - eg. licence managers, backup systems, mail servers, DNS servers, etc, etc.
- ✍ Performing these configurations individually requires an impossible amount of effort for a large site. If there is no automatic way of updating configurations, it is also difficult to be confident that configurations are always correct - this has implications for security, performance and other critical aspects of the network.
- ✍ The **LCFG_[11]** system was presented at LISA VIII and has now been in use for about two years. This talk will present our experiences with the system and identify those aspects which have worked well, and those which have not.

Overview



- Configuration and installation
 - What is involved?
 - Some desirable characteristics
- Common techniques
- The LCFG system
 - The resource database, subsystems
- High level configuration
- Successes and unfulfilled hopes
- References

- ✍ Identifying the basic installation and configuration tasks, together with some desirable properties of any automatic system, provides the criteria for evaluating existing systems and designing new ones.
- ✍ Some common systems are discussed and evaluated against these characteristics. This provides the motivation for the approach taken by the LCFG system, which is then described in more detail.
- ✍ LCFG has the potential to be used in “high level” way for configuring aspects of the network as a whole, rather than concentrating on individual machines.

Configuration Requirements



- **Hardware**
 - Disks, network interfaces, etc.
- **Network**
 - Addresses, remote filesystems, etc.
- **Software**
 - Packages carried, updating procedures
- **Services**
 - NFS, Mail, DNS, time, etc.
- **Access control**
 - Login, NFS, rlogin, etc.

- ✍ All the above aspects need to be configured whenever a new machine is installed, or an existing machine is rebuilt.
- ✍ The configuration also needs to be easily updated when necessary.
- ✍ It is very useful if the configuration information is available explicitly so that it can be easily checked and compared. Eg. how many machines depend on server X for their DNS?
- ✍ An automatic configuration system needs to provide some way of specifying, storing and applying this information.

Desirable Characteristics



- Persistence
- Availability
- Flexibility
- Extensibility
- Automation
- Portability

These are important attributes for any automatic configuration system

- ✍ Configuration information must persist across system rebuilds, allowing machines to be easily rebuilt with an identical configuration - eg. after a crash. The information should also be available when the system is down, so that network-wide summaries can be easily extracted and configurations can be compared. This implies that complete configuration information should be stored apart from the machine itself.
- ✍ The system should be flexible enough to support a wide range of configurations, and easily extensible to handle new subsystems and configuration options. It should also be modular so that different subsystems can be maintained by different people. It is important that the system is "lightweight" so that simple extensions can be made with little effort.
- ✍ It must be possible to rebuild hundreds of machines without individual attention to each machine, even if the configurations of the machines are different.
- ✍ The system must be portable across different platforms.
- ✍ Low cost (ie. free!) is also important.

Common Techniques



- Vendor installation programs
- Individual machines which have been hand-configured can be cloned to produce identical machines
- Some differences are usually required between systems and cloned machines require further customisation
- Configuration information for the customisation may come from a database

- ✍ Vendor installation procedures normally meet very few of the “desirable characteristics” - they are usually only suited to hand-configuring the base operating system for a single machine. Persistence is only provided for a very limited amount of configuration information - eg. NIS maps, and the procedures are not usually portable.
- ✍ Individually configured machines can easily be cloned, simply by copying the local disks. Usually, the clones require some customisation and many schemes have been used for doing this, either during, or after, the cloning process - eg. **Typecast**^[4], **Mkserv**^[5], **Config**^[14], etc, etc.
- ✍ In the simplest case, customisation involves the hand-editing of configuration files which may have copies stored remotely to achieve persistence and availability - eg. **OMNICONF**^[12].
- ✍ Customising a machine by creating configuration files automatically from information in a central database makes the information easier to modify, summarise, and compare - eg. ^[18,19]

Dynamic Configuration



- Dynamically configured systems read configuration information on-demand from a remote source
- This requires less customisation than statically configured systems
- The configuration of the machine is always up-to-date with the external source
- Such machines can not operate stand-alone unless the configuration information is cached

- ✍ In a dynamic system, machines read configuration information as it is required, from some central source. Eg. protocols such as **bootp** and **dhcp** provide small amounts of configuration information dynamically. "As required" should be interpreted rather loosely - it may really mean "every time the information is required", but more commonly, it means "when the machine is rebooted", or when a particular subsystem is restarted.
- ✍ Since purely dynamic systems hold no local configuration information, all machines could be identical clones (in theory). In practice, purely dynamic systems are not reasonable, because some aspects of the configuration cannot easily be changed "on the fly" - eg. disk partition sizes.
- ✍ By guaranteeing that we have a central record of machine configurations which is always correct, we can use this data to prove properties of the network, such as aspects of security.

- The Local ConFiGuration system is a framework for an extensible configuration system
- Used successfully for two years with 300+ machines and extended by different sysadmins
- This system satisfies most of the “desirable characteristics” identified earlier
- A mainly dynamic system with a configuration database suitable for “high-level” configuration

- ✍ A complete automatic configuration system is a very large undertaking. Given the speed with which requirements change, it is unlikely that such a system could have been constructed and maintained with available resources. LCFG provides a framework which includes a configuration information database, and tools for constructing dynamic configuration modules very quickly.
- ✍ Initially, only a small number of subsystems were provided, but this has now been extended by many different people to include virtually all the configuration requirements for our machines (currently 55 subsystems).
- ✍ Some aspects of the installation process are inherently vendor-specific. eg. LCFG integrates with **Suninstall**_[1] to perform automatic installations on Suns. Otherwise, the system is largely portable and portable modules can be used alone on other platforms, without porting the complete system. Most of the other “desirable characteristics” are well satisfied.
- ✍ LCFG includes static and dynamic components which use configuration information from the same database. Dynamic techniques are used wherever possible. Since the database includes complete configuration information for all the machines in the network, this information can be validated, or generated from higher-level specifications.

LCFG Implementation



- Modular dynamic subsystems
- Subsystem classes
- Libraries and templates for creation of new subsystems
- Resource “database” holds configuration information as attribute-value pairs
- Fully automatic static installation process, integrated with Suninstall
- Remote control of subsystems

- ✍ Each “subsystem” which is to be configured using LCFG is implemented by a script which retrieves configuration information from the central database and configures the subsystem. eg. the Apache web server, or the DNS configuration. These scripts are written in a very stylised way which allows new subsystems to be added very quickly, making use of existing templates and library functions. Each script must respond to a small number of standard “methods”, which allow the subsystems to be activated at appropriate times - eg. at boot time, or regularly from cron.
- ✍ Subsystems may be members of a class, so that multiple instances of a single script can control multiple instances of the same subsystem. Eg. multiple backup systems.
- ✍ The resource database holds attribute-value pairs which are keyed by host and subsystem name.
- ✍ A single subsystem runs at installation time to perform any configuration which has to be static, such as disk partitioning. This “install” subsystem takes configuration information from the database and passes it to Suninstall.
- ✍ A remote protocol is available, allowing subsystems to be queried and controlled remotely.

Resource Database



- Attribute value pairs (resources) have the form:
 - host.subsystem.attribute = value
- Simple interface allows different technologies to be used for storing resources
- NIS currently being used (temporarily!)
- Machines can be cloned by copying database entries
- The database also holds other information and is used to generate other tables

- ✓ Wildcard values are permitted in database entries, which allows defaults to be supplied for particular machines and/or particular subsystem classes.
- ✓ NIS was used as a way of implementing the required functionality quickly - it needs to supply resources to remote hosts and support multiple copies of the database for robustness. The intention is still to move this implementation to some other technology.
- ✓ Since machines always reflect the configuration in the database, changing the database effects a change in the network configuration (perhaps after waiting for the next reboot). This allows machines to be cloned by copying database entries. It also allows "include" files to be used for specifying common sets of resources to be shared by groups of machines.
- ✓ Other information is stored in the same database - eg. inventory information such as the machine location. Some other network tables are also generated automatically from this database - eg. bootptab, the NIS bootparams map and (parts of) the netgroup map.
- ✓ Some information is used by Macintoshes and PCs although they do not use full implementations.

Subsystems



- Subsystem scripts
- Methods
 - start/stop
 - run,
 - query/log
 - doc
- Remote control
 - om/omd

- ✍ Each subsystem (or strictly, each subsystem class) is implemented by a script. Currently, most of these are written in Bourne shell, for portability (probably a mistake - Perl would have been better). A “**generic**” script implements defaults for the standard methods and provides a number of standard functions.
- ✍ The **start** and **stop** methods are run when the system is booted and shut down respectively. Normally, subsystems will read configuration information and start daemons when they are “started” and kill daemons when they are “stopped”. They can also be started and stopped on demand.
- ✍ The **run** method is executed by cron, usually once nightly.
- ✍ The **log** and **query** methods allow the log and the current status to be inspected.
- ✍ The **doc** method prints documentation for the subsystem and the supported resources.
- ✍ Each host runs a daemon (**omd**) which listens to requests from a client (**om**) to call methods in specified subsystems. Authority for different users to execute different methods from different subsystems is specified in the resource database. This allows most users to query subsystems and selected users to restart selected subsystems, for example.

Special Subsystems



- **boot**
 - Starts and stops all other subsystems sequentially
- **update**
 - Updates files on the local disk
- **updateif**
 - Updates local binaries
- **patch**
 - Applies system patches
- **install**
 - Performs static installation

- ✓ The **boot** subsystem starts when a system is booted and stops when the system is shut down. It consults the database to see which other subsystems should be active on the machine and starts/stops them sequentially. It also runs periodically from cron to call the run method for other subsystems.
- ✓ **Update** runs regularly to update any files on the local root partition which may have changed. This involves a small number of core configuration files and programs such as the boot script itself. It does not include the majority of programs and data files on the local disk which are updated by a separate process ...
- ✓ **Updateif** calls `lfu[9]` to update the local binaries.
- ✓ **Patch** runs regularly to apply patches to the OS. It consults the database to see which patches should be applied and compares that with a list of applied patches. Patch can automatically schedule a reboot, or mail the administrator if one is required.
- ✓ **Install** is run once to install a new machine. This takes information from the database and uses it to configure Suninstall. This subsystem is largely vendor-specific.

Example Subsystems



- accounting
- amanda
- amd
- apache
- auth
- cap
- dns
- gated
- hfs
- inet
- kerberos
- mail
- mrouted
- news
- nfs
- oracle
- package
- pcnfs
- plp
- quotas
- reflector
- samba
- snmp
- squid
- ssh
- syslog
- system
- volmgt
- xdm
- xntpd

- ✍ Some subsystems typically control daemon processes. These scripts usually configure and start the daemon during the “start” method, and stop the daemon during the “stop” method. eg. the **apache** subsystem control the Apache web server.
- ✍ Some subsystems set up configuration which is used by other parts of the system. eg. the **auth** subsystem creates a number of authorisation files from information in the database. This usually occurs in the “start” method, and the other methods are unused.
- ✍ Some subsystems perform regular operations. These scripts sometimes configure themselves during the “start” method, but perform the bulk of their actions during the “run” method (which gets called regularly from cron). Eg. the **package** subsystem checks the Sun packages installed on the machine and adds or removes packages to match the specification in the database.

High-Level Configuration



- Since complete configuration information for all machines is available in the database, this can easily be used to validate and query properties of the network
- It is also possible to generate individual resources in the database from a higher-level description of the network configuration

Validation:

- ✍ Individual resources can be validated for simple syntax - eg. IP addresses, user names, etc.
- ✍ Consistency of resources with an individual machine can be checked - eg. a machine which is intended to run the Apache subsystem, should be carrying the appropriate software package.
- ✍ Consistency of resources between machines can be checked - this allows the identification of many network problems - eg. A client is importing a filesystem that is not being exported by the server, or a particular ethernet segment has no bootparam server. This can also be used for “what if” analysis - eg. what happens if this server were to crash?

Generation:

- ✍ In theory, it is possible to automatically generate configurations for individual machines from higher level rules. This prevents small errors which occur when creating individual configurations, and allows policies to be enforced automatically.
- ✍ Eg. Machine C is a client of server S implies many individual configuration resources for NFS, DNS, NIS, etc.

Successes



- Configuration database
 - Cloning and customisation are very easy
 - Configurations are visible
- Dynamic configuration
 - More confidence in network configuration
- Modular construction
 - Allows many different people to extend system
- A lightweight framework
 - System is used for all aspects of configuration, because it is the easiest way to do things

- ✓ Previously, a single monolithic script was used to customise cloned machines. Any further customisation involved editing the script (which usually broke something completely unrelated). Crashed machines can now be completely re-installed on new hardware with a single command. It is also easier to provide users with more individual configurations.
- ✓ Dynamic configuration has increased boot times, but the advantages have been worthwhile.
- ✓ Machine configurations are guaranteed to match the database and the information from the database can easily be extracted and validated. We now have much more confidence in certain aspects of configuration across the network - eg, patch levels are correct on all machines, authorisation for various services is consistent and correct.
- ✓ Modular construction has been vital - without this, individuals would not have been able to contribute to the system and it would not have developed. Similarly, the ease with which new subsystems can be created is very important.

Unfulfilled Hopes ...



- High level validation and generation are really interesting possibilities, but this needs a lot of work
- We would still like to replace NIS as the database and provide some caching mechanism for resources on the local machine
- The remote subsystem control has not been fully exploited

... and one mistake

- The framework should have been written in Perl !

- ✍ High-level validation and generation are extremely interesting and would be very useful to have. We simply haven't had the resources to develop this area. There are many interesting problems, including the design of a configuration specification language.
- ✍ NIS was recognised as a "temporary" solution for the resource database, but there has never been effort available to replace it. Fetching individual resources is slow, and queries where the key is not known involve enumeration of the entire map. We would also like to cache resources on the local machine so that machines could be disconnected from the network for a time.
- ✍ We would have liked to extend the remote control system to handle parallel operations on groups of remote systems. We would also like to strengthen the security, perhaps using Kerberos.
- ✍ Bourne shell was originally chosen for the framework in an attempt to be extremely portable. Perl is now widely available and using the power of Perl would have made many things a lot simpler - several subsystems call Perl anyway and we would like to re-implement the framework in Perl if we get time.

A Sample Client



```
#define DISKSIZE          200
#define                  LFCS_CLIENT_ACCESS
#define ARCH              sun4c
#define _OWNER            smk
#define _LOCATION          2602
#include <lfcs_client.h>

info.make                Sun
info.model                4_50=IPX
info.sno                  315t1097
info.hostid               572148dd
```

- Configuration files are passed through the C pre-processor, so that “defines” and “includes” can be used. This means that simple clients require virtually no machine-specific resources - the default values are adequate for many resources, and others are supplied by a header file which defines resources for clients in that “group”.
- The default resources for all clients in the LFCS group are defined in the **lfcs_client.h** file. The configuration file for this particular client simply defines some symbols that are used by the header file, and declares some “info” resources which are just used for informational purposes.

A Sample Server (1)



```
#define _OWNER          root
#define _LOCATION        Machine-Room
#define EXTRA_SERVICES www cap amanda reflector
#define EXTRA_RUN      amanda
#define EXTRA_INET     tftp bootps amanda
#include <lfcs_server.h>

info.make              Sun
info.model             10/40
info.memory            16 16 16
info.sno               411m1238
info.hostid           727099f2

amanda.cluster         lfcs2
amanda.days            Mon Tue Wed Thu Fri Sat Sun

www.server             /usr/local/etc/httpd/httpd -r \
                      /usr/local/etc/httpd/httpd_gc.conf

cron.run_boot         30 0 * * *
cron.objects          boot amanda
cron.run_amanda       35 16 * * 1-5
cron.args_amanda      1
cron.additions        backup
cron.owner_backup     root

install.filesystems   root swap var opt usr install export\
                      scratch tmp spare src4 src2

install.fs_root       c0t3d0s0 64 /
install.fs_swap       c0t3d0s1 64 swap
install.fs_var        c0t3d0s3 64 /var
install.fs_usr        c0t3d0s4 auto /usr
install.fs_export     c0t3d0s5 free /export
install.fs_install    c0t3d0s7 400 /export/install

install.fs_opt        c2t0d0s7 all /opt
install.fs_src4       c2t1d0s6 existing /disk/home/src4
install.fs_src2       c2t1d0s7 existing /disk/home/src2
install.fs_scratch    c1t4d0s3 existing /disk/scratch/dump
install.fs_tmp        c1t4d0s4 existing /disk/scratch/tmp
install.fs_spare      c0t2d0s2 existing /disk/spare

install.install_server true
install.clients       araig baleshare balta boor carna \
                      char copinsay cow duslic fidra \
                      fogla garay gasker ghamna harris \
                      haskeir horrisdale inns iolla
```

A Sample Server (2)



```
nfs.exports          usr opt scratch tmp src2 src4
nfs.fs_usr           /usr
nfs.fs_opt           /opt
nfs.fs_scratch       /disk/scratch/dump
nfs.fs_tmp           /disk/scratch/tmp
nfs.fs_src2          /disk/home/src2
nfs.fs_src4          /disk/home/src4

nfs.options_usr      -o ro=lfcs_export_binaries -d usr
nfs.options_opt      -o ro=lfcs_export_binaries -d opt
nfs.options_scratch  -o rw=lfcs_export_other -d scratch
nfs.options_tmp      -o rw=lfcs_export_other -d tmp
nfs.options_src2     -o rw=machines -d src2
nfs.options_src4     -o rw=machines -d src4

nfs.nfs_threads      64

cap.excluded_wires   129.215.64.0

reflector.confmgr    bluenote.dcs.ed.ac.uk
reflector.refmon     gasker.dcs.ed.ac.uk
```

A Sample Script



```
#!/bin/sh
class=hlfs;
. /etc/obj/generic

#####
Start() {
#####

    Generic_Start;

    CheckFiles $hlfsd
    LoadResources debug cleanlog_freq cleanlog_zap \
        reload_interval hlfsd_binary disabled args

        if [ "$disabled" ] ; then
            WriteResources disabled hlfsd_binary debug \
                pid cleanlog_freq cleanlog_zap
            OK "Not started (disabled)"
            exit 0
        fi

    CheckResources reload_interval hlfsd_binary
    CheckFiles $hlfsd_binary
    [ "$debug" ] && Debug start

    # Run the daemon
    $hlfsd_binary -n -p -i $reload_interval $args >/tmp/$$
    sleep 4
    pid=`cat /tmp/$$`
    rm /tmp/$$

    WriteResources debug pid cleanlog_freq cleanlog_zap \
        reload_interval hlfsd_binary disabled args

    if [ "$debug" ]; then
        ReadResources
        ShowResources DEBUG $reslist
    fi

    OK $start_type
    exit 0
}
```

References (1)



- [1] Sun Microsystems. **Automatic Installation**. In Solaris 2.3 system configuration and installation guide, 1993.

- [2] George M Jones and Steven M Romig. **Cloning customised hosts (or customizing cloned hosts)**. In Proceedings of the 5th Large Installations Systems Administration (LISA) Conference, pages 233-237, Berkeley, CA, 1991. Usenix.

- [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**. In Proceedings of the 6th Large Installations Systems Administration (LISA) Conference, pages 73-78, Berkeley, CA, 1992. Usenix.

- [5] Mark Rosenstein and Ezra Peisach. **Mkserv: workstation customisation and privatization**. In Proceedings of the 6th Large Installations Systems Administration (LISA) Conference, pages 89-95, Berkeley, CA, 1991. Usenix.

- [6] Rick Dipper. **Management information and decision support tools for Unix systems administration**. In Proceedings of UKUUG/SUG Conference, pages 143-153, 1993. UKUUG.

- [7] Paul Anderson. **Local system configuration for sysies**. CS-TN-38, Department of Computer Science, University of Edinburgh. Available from <http://www.dcs.ed.ac.uk/~paul/>.

- [8] Sun Microsystems. **The network information service**. In System and network administration, pages 469-511, 1990. Sun Microsystems

References (2)



- [9] Paul Anderson. **Managing program binaries in a heterogeneous UNIX network.** In Proceedings of the 5th Large Installations Systems Administration (LISA) Conference, pages 1-9, Berkeley, CA, 1991. Usenix.
- [10] Brent Hagemark and Kenneth Zadeck. **Site: a language and system for configuring many computers as one computing site.** In Proceedings of the 3rd Large Installations Systems Administration (LISA) Conference, pages 1-13, Berkeley, CA, 1989. Usenix.
- [11] Paul Anderson. **Towards a high-level machine configuration system.** In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 19-26, Berkeley, CA, 1994. Usenix.
- [12] Imazu Hideyo. **OMNICONF: making OS upgrades and disk crash recovery easier.** In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 27-31, Berkeley, CA, 1994. Usenix.
- [13] Magnus Harlander. **Central systems administration in a heterogeneous Unix environment: GeNUAdmin.** In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 1-8, Berkeley, CA, 1994. Usenix.
- [14] John P Rouillard and Richard B Martin. **Config: a mechanism for installing and tracking machine configurations.** In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 9-17, Berkeley, CA, 1994. Usenix.
- [15] Paul Riddle. **Automated upgrades in a lab environment.** In Proceedings of the 8th Large Installations Systems Administration (LISA) Conference, pages 33-36, Berkeley, CA, 1994. Usenix.

References (3)



- [16] Michael E Shaddock and Michael C Mitchell and Helen E Harrison. **How to upgrade 1500 workstations on Saturday, and still have time to mow the yard on Sunday.** In Proceedings of the 9th Large Installations Systems Administration (LISA) Conference, pages 59-65, Berkeley, CA, 1995. Usenix.
- [17] Christine Hogan. **Decentralizing distributed systems administration.** In Proceedings of the 9th Large Installations Systems Administration (LISA) Conference, pages 139-148, Berkeley, CA, 1995. Usenix.
- [18] Gregory S Thomas and James O Schroeder and Merrilee E Orcutt and Desiree C Johnson and Jeffrey T Simmelink and John P Moore. **Unix host administration in a heterogeneous distributed computing environment.** In Proceedings of the 10th Large Installations Systems Administration (LISA) Conference, pages 43-50, Berkeley, CA, 1996. Usenix.
- [19] Michael Fisk. **Automating the administration of heterogeneous LANs.** In Proceedings of the 10th Large Installations Systems Administration (LISA) Conference, pages 181-186, Berkeley, CA, 1996. Usenix.