Tracking Hardware Configurations in a Heterogeneous Network with syslogd

Rex Walters
IBM Microelectronics Division

Tracking Hardware Configurations in a Heterogeneous Network with syslogd

Rex Walters – IBM Microelectronics Division

ABSTRACT

Keeping track of the RAM, disk drives, and various SCSI, network, and display adaptors currently installed in dozens (or even hundreds) of workstations can be a nearly impossible task, particularly in environments where hardware is frequently "borrowed" from one workstation and moved to another, or where machines frequently come and go. This paper describes a novel method of using *syslog* (3) to allow the workstations themselves to log their current hardware configuration with a central host every time they boot. The application reuses existing tools wherever possible, and thus provides a good degree of platform independence. The programs work with nearly any UNIX system, and should be extendable to other non-UNIX (but TCP/IP enabled) systems with only modest effort.

Motivation

A question commonly asked of system administrators, but surprisingly difficult to answer, goes something like, "How many workstations do we have available that can ...?". Variations of the question such as "Which workstations have 'extra' (memory|disks|adaptors) that we can use elsewhere?" or "Where are the workstations with the most resources?" are also common.

The difficulty arises from two facts of life:

1. Any database with this information, no matter how diligently maintained, is almost certainly out of date. Such databases are often based on purchasing records, which in many cases are out of date almost as soon as the hardware is unpacked.
2. Any attempt to determine the necessary information on-the-fly is also likely to fail due to one or more machines being unavailable (they might, for example, have been temporarily located elsewhere, scavenged to the point of inoperability, sent out for repair, or simply turned off).

The conflicting goals of avoiding manual labor (maintaining a database) yet keeping current with our workstation hardware configurations led directly to the development of the software described in this paper.

Design Issues

As is so frequently the case, the software evolved from a simple cobbled together script into something altogether more complex. A number of difficult design issues arose as the software evolved, and, perhaps surprisingly, the decisions made frequently led to somewhat more elegant code.

One overriding goal was to reuse existing tools and utilities as much as possible, drawing upon the collective experience of "the net" wherever practical. The final implementation was written in *perl* (1), uses *sysinfo* (2) to extract information in a platform independent manner, and uses standard UNIX *syslog* (3) to provide, in effect, a simple client-server database capability. The use of these three tools simplified the coding task tremendously (*logconfig* comprises only a few hundred lines of *perl* code). Using applications that had already been ported to numerous platforms also made the software inherently platform independent.

After expending a great deal of effort on some rather tortuous code, we gave up trying to track each hardware configuration change, and decided to only keep the *current* configuration of each machine (and ignore previous configurations). This simplified the code significantly.[1]

Another early decision was to allow the system to track information that could only be entered interactively, not just the information that could be determined automatically. In addition to the current hardware installed, we wanted to keep track of each workstation's primary user, owner, location, primary use, and serial number – none of these could reliably be determined programmatically. Simply adding an option to have the program prompt an interactive user for the desired information proved sufficient.

Despite the temptation to add more and more information to the database, however, we studiously avoided logging anything that already had a final arbiter (we didn't, for example, log hostname aliases or IP addresses since the canonical reference for this data is DNS). We also avoided logging anything that changes too frequently or that an unprivileged user could change in the normal course of using the

[1]The final implementation has the serendipitous side effect that the local configuration file on each machine keeps a running history of the hardware changes for just that machine.

system. We decided to log the amount of completely unallocated disk space, for example, but not the amount of "free" disk space in each filesystem as returned by *df* (unallocated disk space is common under AIX since it allows filesystems to be extended dynamically).

Since our network (not surprisingly) is comprised almost entirely of IBM RS/6000 workstations, the initial script was very AIX specific. This was an incomplete solution even in our environment, however, so we strove to make the software as platform-independent as practical (but still without preventing us from logging platform-specific data). The *sysinfo*(1) program from Michael Cooper of USC provided a platform-independent mechanism for gathering hardware configuration data, but we still took pains to structure the program such that platform-specific data gathering routines could be added easily. The current implementation only supports UNIX platforms, but it's anticipated that PCs and Macintoshes may also be supported in the future (possibly through some sort of proxy mechanism).

Our network comprises only one hundred or so individual workstations, but we specified that the final implementation "must handle at least 1000 workstations without becoming unduly stressed". This was done to ensure that the software would outlive expected growth at our site as well as to ensure that it would be applicable elsewhere.

Although much of the information generated from this software may eventually end up in a relational database of some sort, the administrative complexity of setting up and maintaining a full relational database outweighed the benefits of using one. We decided to store the configuration data in an easily parsed format that would could be easily transferred to a true relational database. A simple flat file "database" maintained by syslog, with one "record" per line and "fields" separated by white-space proved quite sufficient.

We eventually came up with the somewhat novel approach of using *syslog*(3) to implement a simple client-server database. The decision to use *syslog*(3) solved a number of thorny issues such as file locking and concurrency, and provided a simple communication mechanism for logging information from a remote host. The technique is more generally applicable for any data gathered on a host-by-host basis. A previous paper by Shipley and Wang describes a system for monitoring system usage at the Jet Propulsion Laboratories that uses *syslog* in a very similar manner [4].

Implementation

The final system was written in *perl*(1) (version 5.001), and consists of three separate programs: *logconfig*, *pruneconfig*, and *prconfig*, described below.

One machine in the network is designated the *loghost*, and maintains a configuration file with entries for every machine in the network (the "master config file"). Every other workstation in the network maintains a local configuration file containing only the information about that host (the "local config file"). The master config file is essentially the logical concatenation of all the local config files in the network. Each workstation uses *syslog*(3) to log configuration information in the local config file, as well as to relay the information to *syslogd* on the loghost for inclusion in the master config file. The loghost itself maintains both a local config file containing only its own data as well as a master config file containing data for the entire network. The difficulty of distinguishing between the two types of log data seen by the loghost (locally generated and remotely received) was overcome by logging local data at a higher priority (see below).

The core of the system is *logconfig* which uses the freely available program *sysinfo*(1) and, optionally, other platform-specific utilities to determine information about a system; it also uses *syslog* to actually log the information. By default, *logconfig* only logs hardware changes since the last time it was invoked (although this behavior can be overridden with a flag to "force" the logging of all configuration information obtained during the current invocation). Every machine in the network has its appropriate boot script modified to call *logconfig* each time it is booted. *logconfig* may also be invoked by any user at any time.

```
ahab             7012/350       MEM:  192/416 MB  DISK:    992MB (20)
AIX-3.2.5 <>3250 IDs: 0943b438 000025183800  USER: Mary Buck
   7012-26-60834 LOC: C108/060          FUNCTION: client
                 "IBM 0663L12" 1.0 GB SCSI Disk Drive; "IBM97N" Color Graphics
                 Display Adapter; Diskette Drive; Standard Ethernet Adapter;
                 Standard I/O Diskette Adapter; Standard I/O Parallel Port
                 Adapter; Standard I/O Serial Port 1; Standard I/O Serial
                 Port 2; Standard SCSI I/O Controller; Token Ring Network
                 Interface; Token-Ring High-Performance Adapter;
```

Figure 1: Example output from *prconfig*

pruneconfig is run periodically from cron on the loghost to prune redundant information from the master config file (caused primarily by modifying or removing hardware from workstations, or by using the "force" flag with logconfig).

prconfig is a simple utility to present the information contained in a config file in a more human-readable format. By default, *prconfig* only reports on information in the local config file, but if the master config file is available (at our site, the master config file is available everywhere via NFS) it can optionally generate reports for the entire network. The original intent was to provide a variety of report formats for *prconfig*, but in practice the single format shown in Figure 1 proved to be sufficient for the majority of uses. For the occasional case where the report format isn't appropriate, or a more complex query of the database is desired, it is a simple matter to cobble together a "one-off" script by modifying a copy of *prconfig* (even a simple *perl* or shell one-liner is often sufficient to query the database).

The *syslog* configuration file on every client machine, `/etc/syslog.conf`, must have two lines appended:

```
local2.info        /var/CONFIG
local2.info        @loghost
```

This reflects our configuration where the *syslog* facility used by *logconfig* is "local2", data is logged at the priority "info", the local config file is `/var/CONFIG`, and the hostname (or alias) for the loghost is "loghost".

The syslog configuration file on the loghost contains:

```
local2.notice      /var/CONFIG
local2.info        /var/ALLCONFIG
```

Since only data generated by the loghost itself is logged at priority "notice", the local config file, `/var/CONFIG`, only contains information about the loghost itself. The master config file, `/var/ALLCONFIG`, contains all data logged at priority "info" *or higher*, and thus contains both the locally generated data as well as data from all the other machines on the network.

A config file, as generated by *syslog*, can be viewed as a simple flat-file database, with newlines delimiting records and whitespace delimiting fields within a record. As an example, some of the lines from our network's master config file are shown in Figure 2.

Each line is prefixed by *syslogd* with the date, the name of the host that generated the line, and the string "CONFIG:". Each line also contains fields for a "tag", an "index", and arbitrary text. After running *pruneconfig* on the config file, the triple (*hostname*, *tag*, *index*) is guaranteed to be unique for each line (record) in the file, and thus may be used as the key/index for a table in a relational database. Figure 3 shows the major portion of the code in *pruneconfig*, and illustrates the pruning algorithm (simply put, later entries supersede earlier ones).

Currently the following tags are defined:

HOSTID The identifier returned by the *hostid*(1) command on a UNIX system (the *hostid* is frequently used by license managers, but is of dubious value on an RS/6000 because there is no guarantee of uniqueness).

MODEL The model of workstation being logged. Unfortunately, *sysinfo* cannot reliably determine the model of any arbitrary system since new models are, of course, introduced all the time. This implies that platform specific routines must correctly determine and consistently label the model (this is a frequent search field for database queries like "tell me all of the RS/6000 model 380's or better with more than 128 MB of RAM").

RAM The amount of physical memory in the system.

VM The amount of virtual memory or paging/swap space in the system.

```
Dec 13 16:17:42 ryobi.raleigh.ibm.com CONFIG: FUNCTION: * client, printserver
Jan 23 10:33:36 craftsman.raleigh.ibm.com CONFIG: SERIAL: * 7012-26-77365
Apr 17 20:50:00 skil.raleigh.ibm.com CONFIG: OSLEVEL: * <>3250
Apr 21 13:42:11 decker.raleigh.ibm.com CONFIG: FREEDISK: hdisk0 800 MB
Apr 21 13:43:00 dewalt.raleigh.ibm.com CONFIG: DEVICE: sys0-sysplanar0-bbl0
"GXT150 Graphics Adapter"
Apr 23 16:40:34 speedway.raleigh.ibm.com CONFIG: HOSTID: * 1a000396
Apr 23 16:41:13 pmatic.raleigh.ibm.com CONFIG: DEVICE: inet0-fi0 FDDI Network
Interface
Apr 23 16:44:29 elu.raleigh.ibm.com CONFIG: MODEL: * 250/PowerPC
Apr 30 11:17:18 pmatic.raleigh.ibm.com CONFIG: DEVICE: inet0-fi0 *REMOVED*
```

Figure 2: Sample lines from a master config file (lines wrapped for clarity).

OSNAME The name of the operating system (e.g., "AIX" or "SunOS").

OSVERSION The revision level of the operating system (e.g., "3.2.5").

MACHINEID The "machine identification", that is, what is typically returned by `uname -m` on a UNIX system. On RS/6000s this returns a unique hardware id for that machine (most license managers for AIX use this value for nodelocked software). On Suns, this returns the machine hardware architecture (e.g., "sun4m") which is decidedly not unique.

OSLEVEL This tag is currently only meaningful on AIX workstations, and refers to the level of patches and systems software installed (e.g., on an AIX system with an OSVERSION of 3.2.5, the OSLEVEL might be ">3.2.5" if any patches to 3.2.5 had been applied, "<3.2.5" if any systems software not at the 3.2.5 enhancement level is currently installed, or even "<>3.2.5" if both cases were true). The value for this tag is determined by running the AIX *oslevel* (1) command.

TOTDISK The total amount of hard disk space available in the system (in megabytes).

FREEDISK The total amount of completely unallocated disk space on a system (probably only meaningful with operating systems like AIX that allow filesystems to grow dynamically).

DEVICE This is currently the only tag which is normally logged multiple times for a single machine (once for each device in the system). This information is determined solely from *sysinfo*. A unique index is constructed for each device described by *sysinfo* by concatenating the names of each hierarchical "parent" device. For example, in Figure 2, the graphics adaptor for "dewalt" is the first "bbl" device on system planar 0 of system 0, and is thus indexed with `sys0-sysplanar0-bbl0`.

LOCATION The physical location of the machine. This information can only be determined interactively, and thus can't be logged automatically at boot time (a user must invoke `logconfig -i` explicitly). All of the systems administration staff at our site habitually run `logconfig -i` whenever a system is moved or ownership changes (and we send out occasional messages to all users asking them to run *logconfig* interactively on the machine they are currently using).

OWNER The person or organization that owns the machine.

USER The primary user for the machine (this can be a person or a generic category of users like "admin staff").

FUNCTION The primary function or functions for this machine, typically "client", "file server", or "compute server".

```
while (<FILE>) {
    # Skip the line unless it's from logconfig
    next unless /(... \d+ \d+:\d+:\d+) (\S+) CONFIG:\s+(\w+):\s+(\S+)/;
    ($host, $tag, $index) = ($2, $3, $4);
    # Remember it (possibly replacing previous entry)
    $line{$host}{$tag}{$index} = $_;
}
foreach $host (sort keys %line) {
    foreach $tag (sort keys %{$line{$host}} ) {
        foreach $index (sort keys %{$line{$host}{$tag}} ) {
            # If last entry was a *REMOVED*, skip it
            next if ($line{$host}{$tag}{$index} =~ /\*REMOVED\*/);
            print $line{$host}{$tag}{$index};
        }
    }
}
```

Figure 3: Algorithm used by *pruneconfig*

SERIAL The serial number of the system as a whole. Sadly, this can *not* be reliably determined automatically (*logconfig* ignores any serial number returned by *sysinfo*).

An index of "*" refers to the system as a whole; any other unique string may be used to index tags with more than one iterated value (each device in a system, for example, will have a log entry with a tag of "DEVICE" but with its own unique index).

When *logconfig* is invoked interactively, it prompts the user for any information it can't determine automatically (such as its current location, owner, and serial number). It will use any previous log entries in the local config file as default values for these queries (so the user isn't obligated to re-enter information that isn't easily obtained — serial numbers, for example, are invariably placed on the most dimly lit and inaccessible surface on a machine).

Figure 3 shows the simplified flow for logconfig.

Note especially that any hardware that was previously logged in the local config file but wasn't detected during this invocation has a special log entry generated to indicate it was removed (the entry will use the same hostname, tag, and index, but will have a text value of "*REMOVED*"). Such an entry is shown in the last line of figure 2.

Without such entries, each machine would show a current configuration that was the superset of all hardware ever present in that machine. The pruneconfig script deletes "*REMOVED*" log entries as well as the entries they refer to from config files.

Since, by default, *logconfig* only logs changes to the configuration, and *pruneconfig* removes any history of changes, it's usually advantageous to only run *pruneconfig* on the master config file, and let the local config files grow without bound. In this manner, the local config files provide a history of hardware configuration changes for each machine.

Practical Experiences

logconfig has proven to be a valuable tool at our site for managing a surprisingly difficult and time-consuming task. Previous attempts to maintain the same information manually invariably led to out of date and incorrect information. Having accurate, up-to-date data at our fingertips helped us uncover a number of resources that were better allocated elsewhere.

The author has taken to carrying a printout from *prconfig* around in his day planner, replacing it with a new printout every couple of weeks.[2] It is extremely gratifying to be able to answer questions about resources immediately (and correctly!), without having to leave a meeting.

Other pleasant benefits included being able to determine a hostname given the name of a user (users are notorious for leaving hostnames out of all problem reports — previously, this was almost always the first question out of a system administrator's mouth), and having serial numbers at our fingertips when placing calls for service or upgrades (crawling behind a machine to find a serial number is a good task to have *n* people do once, rather than one person do *n* times).

Limitations and Future Plans

One design goal that wasn't completely satisfied was to handle host name changes gracefully. It is a fact of life that host names change on occasion, and since the host names are used to index our database we worried that a host name change might invalidate the data. The current system is able to detect if a serial number had previously been logged under a different host name, and complains loudly if it does, but the only mechanism for removing any data logged under an old, invalid name is to manually edit the database. Host name changes have not been a significant problem to date.

The current implementation tacitly assumes that the DNS, NIS, or /etc/hosts databases used for hostname resolution are correct and complete at all times. We assume that no two machines have the same hostname and that the loghost will log the config entry with the correct, canonical hostname (and with the *same* hostname every time). This is, of course, hysterically optimistic, but the rationale is that hostname problems usually surface pretty quickly and occur infrequently in well maintained networks, and, arguably, any errors in the database

[2]The *psbook* (1) utility available with the *psutils* package from Angus Duggan is particularly useful for printing these reports.

```
&init;
&read_local_config_file;
&log_sysinfo;                     # Log platform-independent info
&log_arch_specific('uname');      # Log platform-specific info
&log_interactive;                 # Log interactively gathered info
&log_as_removed(@in_local_config_file_but_not_logged_this_time);
```

Figure 4: Simplified flow for *logconfig*

should be relatively straightforward to untangle as the data being tracked corresponds to actual tangible hardware that can be physically inventoried. Nonetheless, this is probably the biggest weakness with this system.

Using the hostname logged by *syslog* as the primary index into the database also introduces a slight complication in designing a proxy mechanism. It's hoped that the next version of *logconfig* will allow a user to enter information for machines that can't run *logconfig* directly (PCs or workstations without network connectivity, or machines on order). Since *logconfig* will log the hostname of the proxy machine and not the hostname (or other identifier) of the desired machine, it is necessary to specially mark these proxy entries and distinguish them from normal *logconfig* entries. Probably the simplest mechanism would be to insert two additional fields for proxy lines: a special tag "PROXY" to identify it as such, and another field containing the identifier of the machine being described.

Availability

The perl scripts described in this paper are not yet available on an anonymous ftp server; the author hopes to rectify this problem soon. In the interim, please contact the author via email at "rrw@vnet.ibm.com" if you would like to receive the *perl* scripts described here.

Author Information

Rex is the senior system administrator for IBM PowerPC Embedded Processors in Research Triangle Park, NC, where he is responsible for all aspects of administration and planning for a network of approximately 100 engineering workstations. Prior to this, Rex was the systems administration manager for Gateway Conversion Technologies in Morrisville, NC. Rex spent the first eight years of his career with Mitsubishi Semiconductor America in Durham, NC, where at various times he was an ASIC test engineer, design engineer, and CAD engineer. Rex received his bachelor of science degree in electrical engineering from Virginia Tech in 1985. He may be reached electronically at rrw@vnet.ibm.com, or via surface mail at IBM Corporation, 3039 Cornwallis Road, M/S B52/060, RTP, NC 27709.

References

[Author's note: There have been several papers presented regarding the installation and tracking of *software* configurations, including [7] and [8] from last year's LISA conference.]

[1] Larry Wall and Randal L. Schwartz, "Programming Perl", O'Reilly & Associates, 1990. [Perl itself is available from a variety of sites, the official distribution location is ftp://ftp .netlabs.com/pub/outgoing]

[2] Michael A. Cooper, "Sysinfo 3.0.1", University of Southern California, source code and documentation available from ftp://usc.edu/pub/sysinfo.

[3] *syslog* (3), *syslogd* (8), *syslog.conf* (5) man pages available with nearly all versions of UNIX.

[4] Carl Shipley and Chingyow Wang, "Monitoring Activity on a Large UNIX Network with perl and Syslogd", LISA V Proceedings, 1991

[5] lscfg(1) man page available with AIX ("list configuration").

[6] *psutils* written by Angus Duggan. Excellent PostScript manipulation utilities available at any `comp.sources.misc` archive site (including gatekeeper.dec.com).

[7] John P. Rouillard and Richard B. Martin, "Config: A Mechanism for Installing and Tracking System Configurations", LISA VIII Proceedings, 1994.

[8] Paul Anderson, "Towards a High-Level Machine Configuration System", LISA VIII Proceedings, 1994.