The following paper was originally published in the
Proceedings of the Twelfth Systems Administration Conference (LISA '98)
Boston, Massachusetts, December 6-11, 1998

The Evolution of the CMD Computing Environment: A Case Study in Rapid Growth

Lloyd Cha, Chris Motta, Syed Babar, and Mukul Agarwa, Advanced Micro Devices, Inc.
Jack Ma and Waseem Shaikh, Taos Mountain, Inc.
Istvan Marko, Volt Services Group

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

The Evolution of the CMD Computing Environment: A Case Study in Rapid Growth

Lloyd Cha, Chris Motta, Syed Babar, and Mukul Agarwal – Advanced Micro Devices, Inc.
Jack Ma and Waseem Shaikh – Taos Mountain, Inc.
Istvan Marko – Volt Services Group

ABSTRACT

Rapid growth of a computing environment presents a recurring theme of running out of resources. Meeting the challenges of building and maintaining such a system requires adapting to the ever changing needs brought on by rampant expansion. This paper discusses the evolution of our computer network from its origins in the startup company NexGen, Inc. to the current AMD California Microprocessor Division (CMD) network that we support today. We provide highlights of some of the problems we have encountered along the way, some of which were solved efficiently and others that provided lessons to be learned.

The reengineering of computer networks and system environments have been the subject of numerous papers including [Harrison92, Evard94b, Limoncelli97]. Like the others, we discuss topics related to modernization of our systems and the implementation of new technologies. However, our focus here is on the problems caused by rapid growth. With increasing requirements for more compute power and the availability of less expensive and more powerful computers, we believe that other environments are poised for rapid growth such as ours. We hope that lessons learned from our experience will better prepare other system administrators in similar situations.

Introduction

The California Microprocessor Division of AMD was formed from the merger of the Battery Powered Processors group of AMD and the newly acquired NexGen, Inc. at the end of 1995. The NexGen computing environment circa early 1995 was based primarily on Solbourne workstations and Thicknet (10base5) cabling. Over the past three years we have grown and modernized our network into our current network of nearly 1000 nodes. We have added over three terabytes of disk space and added over 500 UNIX compute servers. Our network has been through two major revolutions, incorporating technologies such as 10base5, 10baseT, ATM, CDDI, FDDI, 100baseT, and Fast EtherChannel at various points along the way. Our passwd file has grown from 433 entries at the beginning of 1996 to over 700 entries today. That number may be small by standards set by today's internet service providers (ISPs); however, the resource requirements of a typical user in our environment are far greater.

The account that follows is roughly in chronological order. We provide the reader with extended discussions on topics related to the growing pains of our compute environment.

Out of Money

Startup companies frequently have severe spending limitations, and NexGen prior to the merger with AMD was no exception. Spending authorization was handled by upper management, which had little interest or experience in dealing with large scale UNIX system environments. The early systems administration team had little contact with upper management, and was hard pressed to get approval to spend money for changes to the environment or downtime to make much needed adjustments to the network or systems.

Rectifying this situation required gathering copious amounts of information on systems issues, coming up with solutions for the problems discovered, and quantifying the loss and potential loss of revenue or schedule delays to the business as a whole. To do this analysis properly, we recruited finance and engineering management into the project, and accompanied middle managers to budget meetings to answer questions and to reinforce the business case for system improvements. The increased contact between the systems administration team and upper management led to a perception of the team as business aware, and developed a trust relationship that eased future project approval.

However, the available capital was still in short supply. Verification of x86-based microprocessors involves many many cycles of simulation to ensure

complete functionality and compatibility. The NexGen engineers were hungry for as many CPU cycles as we could provide. Our solution was to build up a farm of machines based on NexGen's own Nx586 CPU running Linux. The CPUs were obtainable with very low overhead and the peripheral hardware required was inexpensive. Demonstrating a commercial large-scale Linux installation also provided public relations benefits.

The Linux solution turned out to be short-lived. With the introduction of the UltraSparc processor, Sun was able to tilt the price-performance ratio back in their favor. In addition, the Sparc based processors were able to run almost all of our preferred CAD applications, while the Linux based machines were capable of running only a limited number of simulation programs. Fortunately, the merger of AMD and NexGen brought an immediate influx of cash which was used to purchase new machines rapidly in large quantities.

Recent developments in the microprocessor market, including the contributions of our own division, are swinging the price/performance pendulum back in the other direction. Our sister division in Austin, TX currently runs a compute farm based on AMD-K6 CPUs running Solaris x86 [AMD97]. Whether our compute farm of x86 computers was an idea ahead of its time is a subject still open to debate and is beyond the scope of this paper.

Out of Control

The early NexGen computing environment suffered from inexperienced system administration and a lack of centralized control. There were a dozen different NIS domains, partially because of an intention to separate groups of users, but also as a result of a misunderstanding of how NIS works. Since most users ended up having to have access to almost all of the domains, a conglomeration of scripts was used to manually synchronize the various NIS domains from one master domain. One of our first tasks was to merge all the domains into one.

During this period, NexGen also lacked a department responsible for deployment of CAD applications software. Such tasks were left to the whims of the individual design groups. Design engineers were forced to sacrifice valuable time installing vendor software to varying degrees of success. Applications were often installed in user home directories. Poorly written .cshrc files circulated among different design groups, the use of incorrect versions of tools was epidemic, and the presence of multiple copies of identical software was fairly common.

The implementation of standard ''setup scripts'' for CAD applications was key to improving this situation. We devised a scheme in which a single setup script would be used for each vendor's tool suite. The setup script would handle any path and environment variable configuration needed. While crude, these scripts achieved our immediate goal of having centralized control over application use.

The users' .cshrc files source the setup scripts for the desired tools. An example of such a script is given in Figure1a. A snippet of a user's .cshrc using this script is given in Figure1b. /usr/local/bin/get_arch is a short script which returns the operating system being used – sunos, solaris, hp9, hp10, or linux in our example. The optional amdpostpath and amdprepath variables are used to eliminate excessive path rehashes when multiple setup files are used. If these variables are set, the user's .cshrc is expected to use them to build the appropriate path variable after all the desired setup scripts have been sourced. If amdpostpath and amdprepath are not set, the setup scripts will append or prepend directly to the path variable.

Some general guidelines governed the use and creation of our setup scripts:

- path and environment variables that are not vendor specific will be appended to and never overwritten.
- no assumptions should be made with regard to the order in which the setup scripts are used by users.
- the sourcing of multiple setup scripts belonging to one vendor (e.g., setup_cadvendor_1.0 and setup_cadvendor_1.1) is not supported. Some checks are put in to prevent such misuse.
- the setup scripts are named by vendor and version number (e.g., setup_cadvendor_1.0). A symbolic link will be provided to the default version (setup_cadvendor → setup_cadvendor_1.0).

We could now ensure that obsolete and duplicate installations could be eliminated without leaving users in limbo. More importantly, it facilitated moves and changes to the application trees without requiring users to alter their own login files. The setup scripts allowed centralized setups for vendor tools while still allowing users freedom to customize their individual login environments. Once users had been converted to the setup script paradigm, we eliminated redundant installation of tools and performed proper reinstallation of haphazardly installed software.

There are several known limitations of our method. We currently only support csh based shells. There was very little demand in our user community for anything other than csh or tcsh, so we haven't been motivated to spend much effort in supporting other shells. Changes to setup scripts or tool versions are only reflected when the user's .cshrc file is executed. The login process is also relatively slow even with amdpostpath and amdprepath variables set. We have received a few complaints about this, but none demanding enough to make improvements to it a priority. In most cases, our users were so relieved to have a simple and relatively foolproof tool setup environ-

```
#  $Id: k6system.figures,v 1.4 1998/09/28 21:02:00 lccha Exp $
#  setup_cadvendor_1.0  -
#     A setup file for cadvendor

# Check for conflicts:
if ($?SETUP_CADVENDOR) then
 if ($?prompt) then
   echo "WARNING: setup_cadvendor_1.0 conflicts with"
   echo "setup_cadvendor_$SETUP_CADVENDOR already sourced in this shell"
 endif
endif

# Find architecture of platform
if (-x /usr/local/bin/get_arch) then
   set ARCH_FOR_SETUP = `/usr/local/bin/get_arch`
else
   set ARCH_FOR_SETUP = "unknown"
endif

switch ($ARCH_FOR_SETUP)
   case 'sunos':
      if ( $?amdpostpath ) then
         set amdpostpath = ($amdpostpath /tools/cadvendor/1.0/bin)
      else
         set path = ($path /tools/cadvendor/1.0/bin)
      endif
      breaksw

   case 'solaris':
      if ( $?amdpostpath ) then
         set amdpostpath = ($amdpostpath /tools/cadvendor/1.0/bin)
      else
         set path = ($path /tools/cadvendor/1.0/bin)
      endif
      breaksw

   case 'hpux9':
   case 'hpux10':
      if ( $?amdpostpath ) then
         set amdpostpath = ($amdpostpath /tools/cadvendor/1.0/bin)
      else
         set path = ($path /tools/cadvendor/1.0/bin)
      endif
      breaksw

   case 'linux':
      exit
      breaksw

   default:
      exit
      breaksw
endsw

# Setup version variable
setenv SETUP_CADVENDOR 1.0

# Setup license files
if ( $?LM_LICENSE_FILE ) then
   if ( "$LM_LICENSE_FILE" !~ *"1700@key"* ) then
      setenv LM_LICENSE_FILE \
         ${LM_LICENSE_FILE}:1700@keya,1700@keyb,1700@keyc
   endif
else
   setenv LM_LICENSE_FILE 1700@keya,1700@keyb,1700@keyc
endif
```

Figure 1a: Sample setup file for fictitious cadvendor (software version 1.0).

ment that they were willing to overlook these shortcomings.

We had also considered using wrapper scripts written in C-shell or Perl. In practice, we found that they take more effort to maintain when new versions of software are installed. Many of our CAD packages consist of numerous binaries which would all require individual wrappers or a link to a common wrapper. The composition of executables in the package can change frequently from version to version, forcing the system administrator installing the software to carefully inspect each release to ensure that all necessary wrappers are in place.

Some alternative methods have been presented in [Rath94], [Evard94a], and [Furlani91]. We may look at implementing some of the ideas presented in those papers in the future. However, feedback from our users has indicated that our method is currently adequate and therefore improvements have not been a high priority.

Out of Disk Space

Prior to 1995, NexGen's data was distributed on a variety of Sun, Solbourne, and HP700 servers which were all cross mounted via NFS hard mounts to each other. This type of design results in a network with multiple points of failure, each of which can affect performance and availability of the entire network. In addition, backup of many small servers tends to be cumbersome, requiring backups running over the network or the installation of numerous local tape drives.

Early 1995 marked the arrival of NexGen's first large centralized fileserver, an Auspex NS5000. Bristling with a dozen network interfaces, it was able to provide reliable file service to all machines on the network without requiring any network router hops. Data from the various ''servers'' was migrated to the central fileserver and network reliability improved accordingly.

We made some decisions in the implementation of this first Auspex that we would later regret. We opted to use automount ''direct'' maps rather than indirect maps based on field reports from other Auspex customers that a large quantity of indirect map entries could cause overloads on the Auspex host processor. This denied us the flexibility and scalability that indirect maps would have given us. We limited the partition size to 5GB per partition due to limitations of the backup technology we were using at the time, Exabyte 8500 8mm tape drives driven by shell scripts using dump. Each partition contained a mixture of project data, user home directories, applications, and temporary data.

The mixture of different types of directories on shared partitions combined with the relatively small size of the partitions was a nightmare to administer. Large amounts of scratch data often filled up partitions causing critical design data to be lost or corrupted. Large vendor application installations had to be awkwardly distributed among several disks.

The merger with AMD in 1996 brought a second Auspex server to our network. This gave us an immediate opportunity to revise our disk usage strategy based on our previous experience. We opted to dedicate individual partitions base on their use. We designated four general categories of disk use:

- Applications
- User home directories
- Project directories
- Critical project directories.

Partitions were individually sized based on needs. Critical project directories were allocated a sufficient amount of free buffer space to minimize disk full situations. At the opposite extreme, application directories were permitted to operate with very little free space in order to minimize waste. Access to these directories was provided by indirect maps. As of this writing we have scaled this plan to eight fileservers, each serving roughly 500 gigabytes of disk.

To monitor disk usage and to give advance warnings of partitions getting full, we wrote a disk monitoring script in Perl to be run hourly by cron. We started by using a simple script that would parse the output of the df command and generate e-mail when any disk

```
set amdpostpath
set amdprepath
foreach vendor (cadvendor mentor cadence avanti)
   if (-e /usr/local/setup/setup_$vendor) then
      source /usr/local/setup/setup_$vendor
   else
      if ($?prompt) then
         echo "WARNING: setup_$vendor does not exist ... skipping"
      endif
   endif
end
set path=($amdprepath $path $amdpostpath)
```

Figure 1b: Snippet of code from user's .cshrc .

fell below a given threshold. This script did not keep any historical data and therefore was not able to detect the difference between a partition that was rapidly nearing capacity from a partition that had inched across the forbidden threshold. As a result, this script generated too many nuisance e-mails which rendered the important warnings useless.

To solve these problems, we wrote highly configurable Perl script with the following features:

- Rules based notification – rules are expressed in Perl syntax and are based on a comprehensive set of conditions tracked by the script.
- Comprehensive set of conditions – rules for notification can be based on any combination of the following conditions:
  - amount of free disk available
  - percentage of free disk available
  - time of day
  - time of last notification
  - various calendar data (month, day, year, day of week)
  - change in free disk available since last notification
  - change in free disk available since last run of script
- History database – disk usage information and a record of previous notification sent is kept. This allows the frequency of warning messages to be tuned to how quickly the disk is filling up.

Out of CPU

As product schedules became tighter and tighter, our need for faster turnaround times on our simulation and verification runs became even more critical. The brute force solution of purchasing more computers was a key part of our solution to this problem. But this alone would not allow us to reach our goal. We needed a way of using the available CPUs more efficiently. We elected to use Platform Computing's LSF (Load Sharing Facility) product to help us reach our goal of having every available CPU in use at all times.

Our model was based on having the most powerful servers located in the computer room. We deployed less powerful (i.e., less expensive) workstations or x-terminals on user desktops and encouraged users to submit all jobs, including interactive ones, to the server farm. Keeping the powerful machines off people's desktops helped prevent large jobs from causing performance problems for interactive users and reduced problems caused by ''possessive'' users that would complain about any background jobs running on their machines.

Our simulation jobs were typically CPU bound with minimal disk and memory requirements. Achieving maximum CPU utilization was therefore the key objective. We deployed LSF on nearly every available machine in the division as well as many machines ''borrowed'' from outside our division in order to maximize the number of CPUs available. Every additional CPU we were able to utilize contributed to an improved time to market for the products being developed by our division.

LSF allows job scheduling based on several factors, including available memory, load average, and the number of LSF jobs already running on the machine. Server room machines were configured to run one job per CPU at all times. Desktop machines are configured to run batch jobs when the load average and the idle time fall within specific parameters. The actual thresholds used were tuned based on user feedback. This allowed us to get maximal use out of idle desktop CPUs while keeping the console users happy.

Further details of our configuration beyond the scope of the paper can be found in [Platform97].

Out of Space

Computers require space. Lots of computers require lots of space. Setting up workstations lined up on tables and ordinary utility shelves will work for smaller installations, but for large installations there is no substitute for well-constructed racks. For flexibility, we opted for an ''open-shelf'' type of rack. We used these racks to stack servers up to limits allowed by local fire codes.

Our primary goals were high density, easy accessibility, and reasonable cost. Since appearance was only a minor consideration, and access to our server room is well-controlled, we were uninterested in enclosed cabinet style racks with doors and locks. Instead, we opted for basic 23'' wide racks with 11'' deep ventilated shelves bolted to them front and back for an effective depth of 22''. This arrangement proved to be highly versatile, accommodating PCs, Sun Sparc machines from the Sparc20s through the UltraServer2 machines, and HP 735s and J-class servers. With most of the ''pizza-box'' style chassis, we were able to stack up to 14 machines on 7 foot high racks. Ladder racks across the top and bolts in the floor provided stability in the event of earthquakes.

Out of Power

In August 1996, during a critical part of CMD's product development schedule, we began to notice a high number of memory failures from our compute server ranch. This led us to suspect some sort of environmental problem. We first suspected that the machines were not being adequately cooled and ventilated. After determining that this was not the cause of our maladies, we then focused on possible fluctuations in the power supply. Bingo! Our facilities department discovered that the transformer outside of our building was operating at about double the rated capacity.

The news from our utility company was not good. We could either take eight hours of downtime to get the transformer replaced immediately at the utility's expense, or risk blowing the transformer and

taking several days downtime to have it replaced at our expense. We opted to plan an orderly Saturday downtime to get the transformer replaced. In the meantime, to lessen the chance of a catastrophic transformer failure, we shutdown any equipment that was not absolutely necessary, including unused monitors, obsolete yet functional computer systems, and hallway lights. After about a week of working in a very dark building, we spent an entire weekend powering machines down and back up.

Moral of this story: check your power requirements carefully with your facilities department before it's too late. We had to pay the price with a hastily planned shutdown at a critical point in the project, but at least we were lucky enough to have not blown the transformer unexpectedly. A related requirement is to ensure that the cooling capacity of your air-conditioning system is sufficient to maintain server room temperature even on the hottest summer days.

Out of Network Bandwidth

Reaching capacity limits of our network was a persistent problem throughout our growth experience. Fortunately, as our network grew, better and faster network technology also became available. Our early network of bridges, hubs, and routers with shared segments and multiport collision domains gave way to a completely switched network composed primarily of Xylan Omniswitches by early 1996. In subsequent years, we were able to migrate our backbone from FDDI to ATM OC-3 technology, and our end stations from 10baseT to CDDI and 100baseT interfaces.

The early network topology (circa 1995) is shown in Figure 2a. Our first major upgrade was to eliminate the hubs of shared ethernet segments and replace them with ethernet switches, creating dual-port collision domains (i.e., one machine per shared ethernet). The resulting switch based network of Figure 2b served us well for approximately three years. It suffered somewhat from the irregular growth that characterized that period of our expansion. We were several times required to add hundreds of machines to our network with no downtime permitted, which left little leeway for major topological changes. As a result, the loads across the various subnets were poorly balanced.

As the network evolved the main bottlenecks that limited our scalability were the numerous routers and the fileserver interfaces. Our new design attempted to eliminate as many of the router bottlenecks as possible. In order to do this, we attempted to flatten the network as much as possible. We had considered completely flattening the network into a single subnet, but

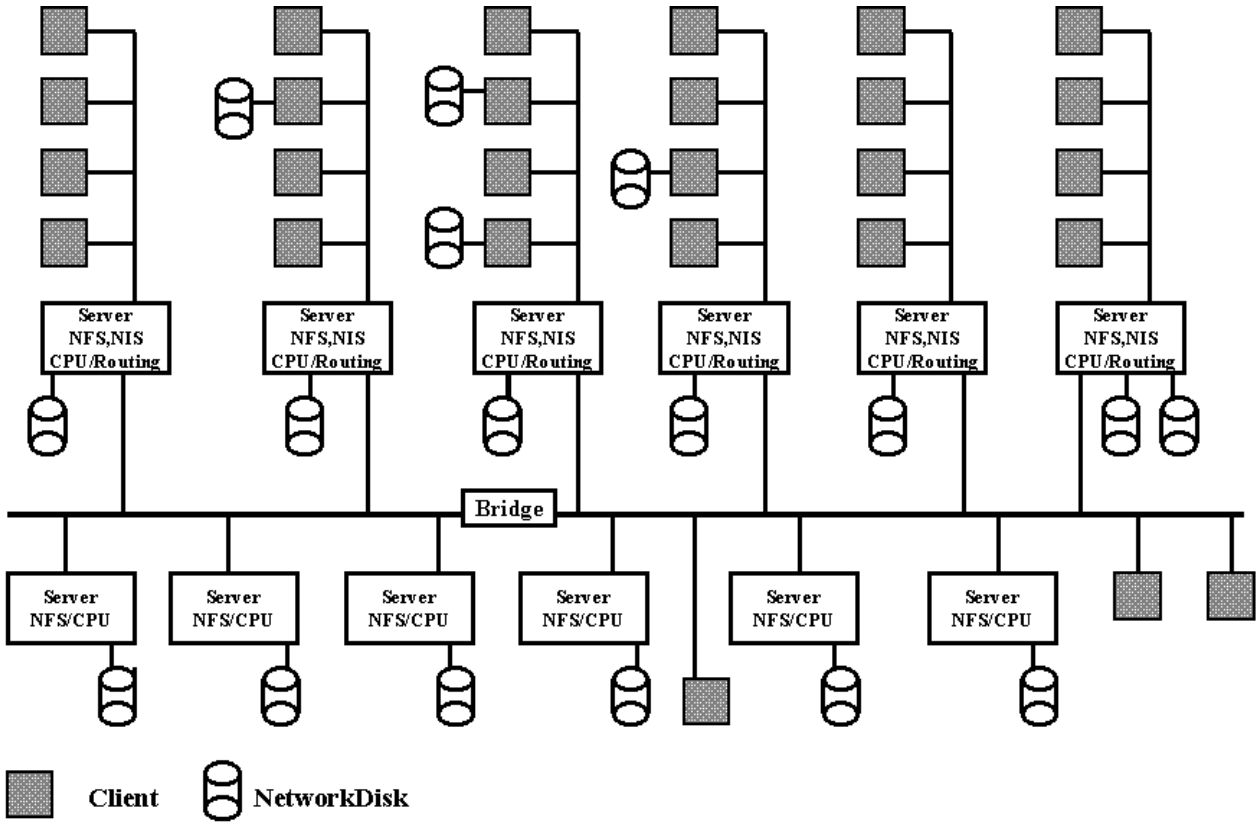

Figure 2a: Early network topology.

we were not able to come up with a suitable topology that had sufficient bandwidth at the core of the network without using unproven technology.

Providing enough throughput to the fileservers would also prove to be a limiting factor in a completely flat network. We opted to use Cisco's Fast EtherChannel technology [Cisco97, Cisco98, Auspex98] to combine several full-duplex 100bT links into a single logical pipe or "channel." Currently a maximum of four links can be combined into one channel, which limits each logical interface to the fileserver to 800Mbps. In order to provide the desired throughput to each fileserver, we calculated a minimum requirement of three 800Mbps channels per fileserver. This implied a minimum of three subnets to avoid having the added complexity of managing a server having multiple interfaces on any one subnet.

The final design as implemented is shown in Figure 2c. The access (bottom) layer of switches provides connections to all of the client compute and desktop machines. The distribution (middle) layer consists of four Cisco Catalyst 5500 switches, each with a route-switch module to provide fast routing between subnets. These switches are responsible for traffic between the various access layer switches and all the routing between the subnets of our local network. Two Catalyst 5500 switches with router cards make up the core (top) layer, which tie together the various distribution layer switches. In addition, the core layer provides access to the routers that handle our external network traffic. All interswitch links use Fast EtherChannel, and every switch is connected to at least two devices in the layer above it for redundancy.

Our analysis indicated that roughly 85 percent of our network traffic was NFS related. NFS traffic is also particularly sensitive to latency, so special accommodations had to be made in the topology for the NFS fileservers. In our design, the fileservers were connected via 800Mbps Fast EtherChannel to the various distribution layer switches to minimize the number of switch hops required by the end stations. Each fileserver had an interface on each of the three subnets, ensuring that every NFS client workstation had access to a fileserver interface without needing a router hop.

Out of Time, Part I

Size does matter [Godzilla98]. Perhaps the most valuable resource in any computing environment is the system administrator's time. Large scale computing environments highlight the need for automation. As we added more and more machines to our environment, it became obvious that many of the methods currently in place were no longer acceptable. Manual procedures had to be scripted or automated in some way. Semi-automated procedures needed to be fully-automated.

The sporadic growth of our network and the constant flurry of moves, adds and changes within our network resulted in a chaotic state of network wiring. The task of tracing a machine to its network port often

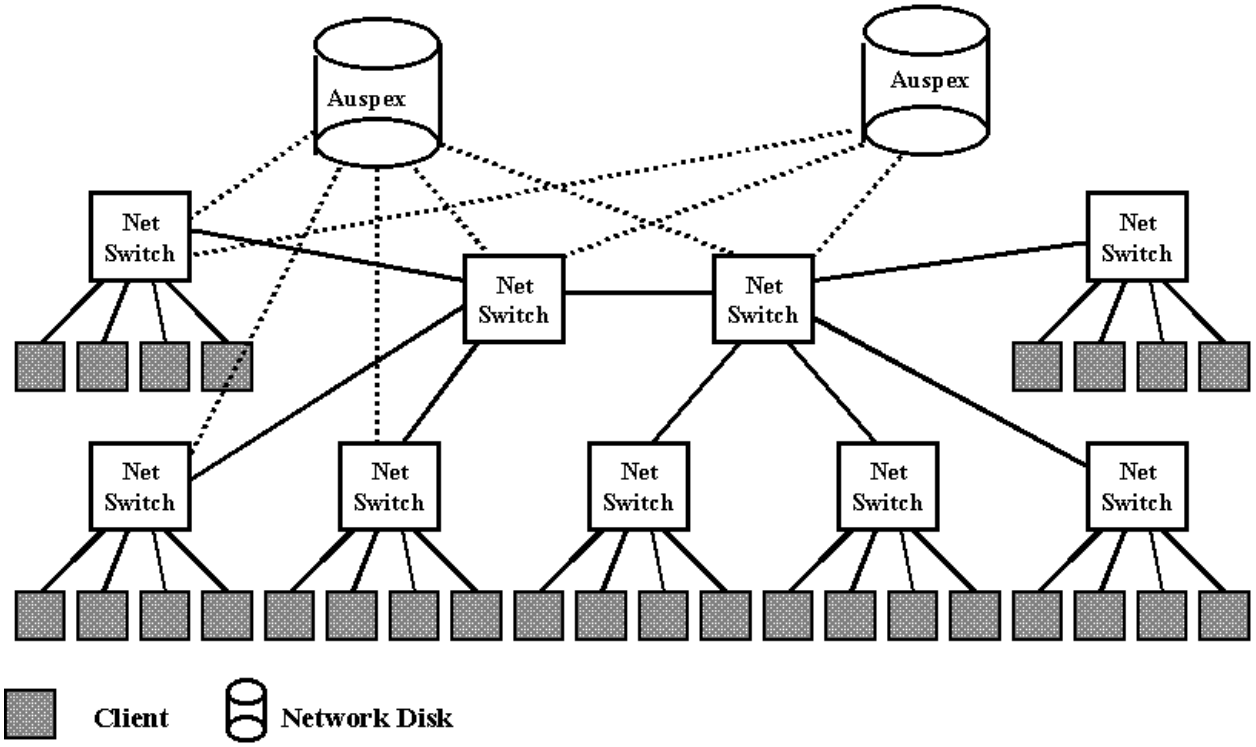

Figure 2b: Switch based network (1996).

required two people to trace through the rat's nest of cables. A week's worth of cleaning up and untangling cables was often undone in less than an hour by an emergency machine relocation or cubicle swap. To solve this problem we developed a package of scripts that collected MAC address information from the arp tables of our network switches and correlated those addresses with the information from our ethers table to accurately report where machines were connected in the network.

We realized early into our bulk purchases of workstations that we would benefit greatly by keeping machine configurations as consistent as possible. The idea was to reduce the "entropy" as described in [Evard97] thereby minimizing debug time. We had loaded our first twenty workstations by cloning hard drives using a simple script that performed a dd from one disk to another. Once the disk copy completed, a post-install script was run to take care of the machine specific configuration.

While this technique is one of the fastest methods of loading the operating systems software, it was also expensive in administrator time. Pulling the hardware off the rack, swapping in the master disk drive, starting the cloning process, and then reinstalling the system would take a minimum of 15 minutes even in best case scenarios. Updating all the machines with this method would theoretically require several man-days and would probably require several weeks in practice.

We now employ a variety of network loading methods, Jumpstart [Sun98] for Solaris products, netdist [HP94] and Ignite-UX [HP98] for HP-UX 9.X and 10.X respectively, and Kickstart for Linux. The basic theory in each of these is similar:

1. Perform a diskless boot using bootpd or similar protocol. A miniature version of the operating system is loaded via tftp into the swap partition
2. Load the operating system onto the local disk
3. Run a customization script to load patches and handle local configuration information.

With these methods, administrator time is now reduced to assuming control of the machine and rebooting with the designated command to force a boot from the install server rather than the local disk. The machine then takes care of the rest. Typical time to load the operating system has increased to a few hours, mostly dependent on the amount of operating system patches involved. The penalty in load time is far outweighed by the benefit of savings in administrator time.

Out of Time, Part II

Our original server room configuration had keyboards attached to every machine and a terminal that was wheeled around on a cart to attach to the console

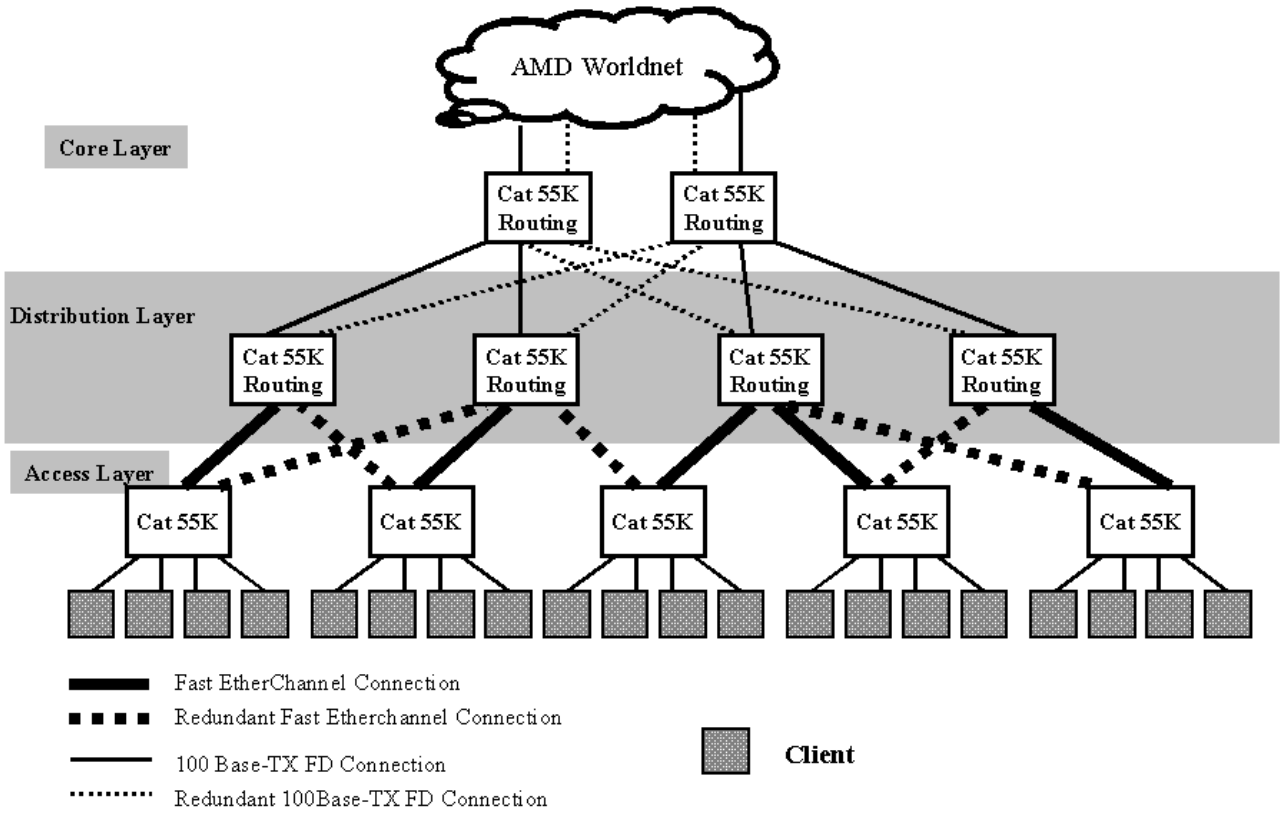

Figure 2c: Current network.

port of any machine that required attention. This was both messy and inefficient.

We were able to solve this problem by using terminal servers with ‘‘reverse telnet’’ capability. This feature allows one to telnet from a remote host to any of the terminal server’s serial ports on the terminal server. Connecting the terminal server’s serial ports to the serial console ports of the compute servers enables one to telnet directly to the console of the machine.

The default configuration of terminal servers provides access to the network from the server’s serial ports. In our case, we used the terminal servers in the opposite direction, to provide access to the serial ports from the network. Consequently, many changes to the default settings of the terminal servers were required to achieve the desired functionality. We list below the most significant changes we made to our Xyplex terminal servers. Other manufacturers probably have similar options:

- **port access mode:** We set the port access mode to ‘‘remote’’ to allow connections to be initiated from the network side. This setting also instructs the terminal server not to output any data to the serial port other than the data being passed from the network port.
- **default session mode/telnet binary session mode:** These should both be set as passall, which directs the terminal server to pass the data from the network port to the serial port without attempting to process the data.
- **telnet echo mode:** This mode should be set to ‘‘character’’ to prevent the terminal server from buffering the data flow.

Our current setup uses twenty 40-port terminal servers. In order to manage the terminal servers and ensure consistent configuration we opted to boot the terminal servers from a single network boot server. This allows us to perform updates to the terminal server configurations by simply loading a new image file on the boot server and rebooting each of the terminal servers. In addition, each terminal has a local flash card that can be used for booting in the event that the boot server is unavailable.

We developed a package of shell and Expect scripts to map connections between the compute server console ports and the terminal server serial ports. Without these scripts, we would have been forced to rely on manually generated documentation. Given the frequency of server moves and changes in our environment, this documentation would have soon become outdated and useless without automatic updates.

Our Sun servers automatically use the serial port if a keyboard is not detected upon bootup. If a Sun machine receives a ‘‘break’’ signal on its serial console port, it halts execution and drops into ROM monitor mode. Unfortunately, power cycling of a terminal server attached to a Sun console port is often perceived by the host as a ‘‘break’’ signal. [Cisco98b] suggests working around this problem by preventing the Sun computers from halting when receiving a ‘‘break.’’

However, this would also prevent us from halting the unresponsive machine remotely by intentionally sending a ‘‘break’’ signal. Since our terminal servers are protected by the same UPS system that protects our compute servers, we decided that the benefit of being able to remotely debug machines was worth the risk of leaving the servers susceptible to a global ‘‘halt’’ due to a power glitch.

We also ran into problems with garbage characters appearing when older telnet clients, such as those provided with SunOS 4.1.X and HP-UX 9.X, were used. We suspect that those binaries were not able to cope with some option that the Xyplex was attempting to negotiate. Compiling new telnet binaries on these machines helped eliminate some, but not all of the problems. This only affected a minority of our machines that were still running older operating systems, so we opted to ignore the problem and require that telnet connections to the consoles via the terminal server be launched from our Solaris based machines.

The Future

We are still in a state where improvements to our computing environments are bounded by a lack of time or manpower rather than a lack of ideas. We recognize that some of our solutions presented here, while adequate for our immediate needs, still leave room for improvement. Some of the projects we are currently working on are listed below:

- **Canonical hostlist project:** We have several databases that require hostname information, including the corporate DNS file, our local NIS hosts file, our local netgroup file, and various location databases. Each of these lists is currently independently maintained. Adding a host to the networks requires manually updating each of these.
  We are in the process of implementing a new methodology. A single file will contain a minimum amount of data which is manually entered by the system administrator. All the rest of the information will be generated by scripts which will collect MAC addresses, network port connections, console port collections, and other data to build a master database of all host information. This database will be used to build DNS and NIS host tables, netgroup files, lists for update scripts, LSF configuration files, and documentation. The principle is to keep data entry to an absolute minimum and to have only a single source for the manually inputed data. By facilitating automated updates of the various data files, we avoid inconsistency problems.
- **Cluster monitor:** We currently monitor our systems by using scripts which process raw

data produced by syslog, LSF, and HP OpenView. LSF gives us a good overall analysis of the total throughput of our cluster. The syslog reports the hardware problems, and HP OpenView supports a variety of monitoring options. Our objective is to develop a system that will give us more detailed reporting and debugging information when there are problems. In addition, we are developing methods to allow the cluster to automatically isolate problems and perform automated fixes without human intervention.

- **Documentation:** We sometimes forget that computing environments are setup for the benefit of our users rather than for the entertainment of the system administrators. Documentation is an essential ingredient for ensuring that a computing environment can be efficiently used and is essential to prevent system administrators from being bogged down by an endless barrage of frequently asked questions. We will be making a major effort to improve our on-line documentation currently maintained on our internal web site.

Final Thoughts

We hope our experiences will be valuable in helping others plan ahead for growth in their computing installations. We have learned that it is important to solve small problems on a small scale before they expand to large problems on a large scale. Careful planning and a vision of the future is necessary to design systems that will scale easily without major renovations.

Availability

Please contact the authors at <lisa98@cmdmail.amd.com> regarding availability of scripts referenced in this paper.

Author Information

Lloyd Cha is a MTS CAD Design Engineer at Advanced Micro Devices in Sunnyvale, California. Prior to joining AMD, he was employed by Rockwell International in Newport Beach, California. He holds a BSEE from the California Institute of Technology and a MSEE from UCLA. He can be contacted by USPS mail at AMD, M/S 361, PO Box 3453, Sunnyvale, CA 94088 or by electronic mail at <lloyd.cha@amd.com> or <lloyd.cha@pobox.com>.

Chris Motta is the manager of the CMD Systems and Network Administration department. He holds a BSME from the University of California at Berkeley. He has held a variety of systems administration positions including UNIX and networking consulting. Electronic mail address is <Chris.Motta@amd.com>, and USPS mail address is M/S 366, PO Box 3453, Sunnyvale, CA 94088.

Syed Babar received his master's degree in computer engineering from Wayne State University in Detroit, Michigan. He works at Advanced Micro Devices in Sunnyvale, California as a Senior CAD Systems Engineer. He can be contacted via e-mail at <Syed.Babar@amd.com> or <Syed_Babar@hotmail.com>.

Mukul Agarwal received his MSCS from Santa Clara University. He joined NexGen, Inc. in Milpitas, California as a CAD Engineer in 1993. He switched to systems and network administration in 1995 and has been a System/Network Administrator ever since. Reach him via e-mail at <mukul.agarwal@amd.com>.

Jack Ma holds a BSCS from Tsinghua University and a MSCS from Computer Systems Engineering Institute. He was a UNIX software developer at Sun Microsystems before joining Taos Mountain at 1995, where he now works as a networking/UNIX system consultant. He can be reached electronically at <ylma@netcom.com>.

Waseem Shaikh holds a master's degree in computer engineering from the University of Southern California and received his bachelor's degree in electrical engineering from University of Engineering and Technology in Lahore, Pakistan. He was a System/Network Engineer at Steven Spielberg's Holocaust Shoah Foundation, a System Consultant at Stanford Research Institute, and is now working as a System/Network Consultant with Taos Mountain. He can be reached at <shaikh@netcom.com>.

Istvan Marko is a self-educated Computer Specialist currently working as a System Administrator employed through Volt Services Group. He can be contacted via e-mail at <imarko@pacificnet.net>.

References

[AMD97] Unpublished internal e-mail correspondence, AMD Austin, TX, 1997.

[Auspex98] "Auspex Support For Cisco Fast EtherChannel," *Auspex Technical Report #21*, Document 300-TC049, March 1998.

[Cisco97] Cisco Systems, Inc. "Fast EtherChannel," *Cisco Systems Whitepaper*, 1997.

[Cisco98] Cisco Systems, Inc. "Understanding and Designing Networks Using Fast EtherChannel," *Cisco Systems Application Note*, 1998.

[Evard94a] Remy Evard, "Soft: A Software Environment Abstraction Mechanism," *LISA VIII Proceedings*, San Diego, CA, September 1994.

[Evard94b] Remy Evard, "Tenwen: The Re-engineering Of A Computing Environment," *LISA VIII Proceedings*, San Diego, CA, September 1994.

[Evard97] Remy Evard, "An Analysis of UNIX System Configuration," *LISA XI Proceedings*, San Diego, CA, October 1997.

[Furlani91] John L Furlani, ‘‘Modules: Providing a Flexible User Environment,’’ *LISA V Proceedings*, San Diego, CA, September 1991.

[Godzilla98] ‘‘Godzilla,’’ Columbia TriStar Pictures, 1998.

[Harrison92] Harrison, Helen E, ‘‘So Many Workstations, So Little Time,’’ *LISA VI Proceedings*, Long Beach, October 1992.

[HP94] Hewlett-Packard Support Services, ‘‘Cold Network Installs – Configuring/Troubleshooting Guide,’’ *Engineering Notes – Document CWA941020000*, December 12, 1994.

[HP98] Hewlett-Packard Company, ‘‘Ignite-UX Startup Guide for System Administrators,’’ 1998.

[Limoncelli97] Tom Limoncelli, Tom Reingold, Ravi Narayan, and Ralph Loura, ‘‘Creating a Network for Lucent Bell Labs Research South,’’ *LISA XI Proceedings,* San Diego, CA, October 1997.

[Platform97] Platform Computing, ‘‘AMD’s K6 Microprocessor Design Experience with LSF,’’ *LSF News, Platform Computing*, August 1997. http://www.platform.com/content/industry_solutions/success_stories/eda_solutions/eda_stories/AMD.htm.

[Rath94] Christopher Rath, ‘‘The BNR Standard Login (A Login Configuration Manager),’’ *LISA VIII Proceedings*, San Diego, CA, September 1994.

[Sun98] Sun Microsystems, Inc. ‘‘SPARC: Installing Solaris Software,’’ 1998.