Morgan Stanley's Aurora System: Designing a Next Generation Global Production Unix Environment

Xev Gittler, W. Phillip Moore and J. Rambhaskar – Morgan Stanley

ABSTRACT

The challenge: To come up with a distributed systems environment that would allow Morgan Stanley to centrally manage tens of thousands of systems spread out over more than 30 offices on virtually every continent on the globe in a fully production fashion.

The solution: The Aurora System.

History

Morgan Stanley is a global investment banking company. The day to day business activity of the firm depends in a large part on the stability, reliability and functionality of its technology. The firm trades on most exchanges around the world, and as such, there are very few hours in a week when trading activity is not occurring somewhere on our networks.

Until November 1993, Unix computing activities within Morgan Stanley were divided into two distinct groups. One business unit within Morgan Stanley, the Fixed Income Division, maintained its own computer management staff, called the Fixed Income Research (FIR) group. The Fixed Income Division required significant computing capacity, and, more importantly, realized and encouraged growth and use of equipment to its fullest potential. FIR introduced Unix systems in 1987 and by 1993 had approximately 1500 systems, 90% Sun and 10% IBM. FIR had nearly complete authority over all computing related decisions.

In contrast, Information Systems (IS) was responsible for the vast majority of Unix systems throughout Morgan Stanley, reporting to many different business units with different needs and desires. In some instances, IS was allowed to suggest technological solutions to problems and implement those decisions. In many cases, however, it was presented with systems to manage with no initial consultation, and any suggestions, especially those that required any additional expenses, were overridden by the business unit. IS introduced Unix in 1990 and by 1993 had approximately 3500 systems, all Suns. IS and FIR rarely cooperated.

In November 1993, FIR, IS and all other computing groups merged into the Information Technology (IT) department, consolidating all computing related activities under a single organization with significant decision making authority. After the merger, in mid-1994, the top distributed systems engineers from both IS and FIR formed into the Core Infrastructure Group (CIG). The mandate given to this group was to create a common distributed systems technology platform for the firm. The CIG had the unique opportunity to dream up the ideal Unix operating environment for our needs and make it happen.

Design Goals

Up until (and through) the merger, our network expanded via the installation of new workstations at a rapid pace. As a result, the system components that we had in place were starting to show signs of strain. For example, our file system distribution was becoming unwieldily, taking too long to complete and systems that should have been identical were no longer in sync. Additionally, replicated copies of the system data accounted for almost 40% of all used disk space, which was far too much overhead.

The pre-Aurora environment was functional, and under ordinary circumstances we could have made it last a few more years without a major overhaul. However, because the systems would undergo a major upheaval in any event as a result of the merger, we decided that we might as well go whole hog and design the perfect system.

In addition to remedying shortcomings of the pre-Aurora systems, our perfect environment needed to address the following issues:

Scale – A major criteria of the environment the CIG developed was its adaptability to scale. We define scale somewhat differently than most, based on our diverse physical requirements in our various locations. Morgan Stanley is not set up in a traditional campus setting. We have over 5000 systems distributed globally in over 30 locations. Our largest offices have over 1000 systems, while the smallest offices have fewer than five. The bandwidth between offices varies from 56Kbs to hundreds of Mbs. The largest offices have hundreds of support staff, while the smallest have no support staff.

Based on this infrastructure, any system that we designed not only had to scale upwards in the traditional sense, but it also had to scale to support the smallest sites as well. This requirement means it must fit on small systems, use less bandwidth and not depend on many different servers to support the environment.

System Usage – Given the nature of our business, even minutes of down-time are unacceptable. Our systems must be operational 24 hours a day, 7 days a week. Except for a few hours a week, trading is always going on. Additionally, we have a fairly static user environment. Users log on to particular workstations, and typically stay logged on until the system reboots. Running jobs and screen configurations are painful for the users to restart. As a result, system reboots only occur on the order of months.

Because our environment must run with zero down time, in designing Aurora we tried to avoid products that only ran in university or research labs. Every part of the environment that we use must either be commercially supported, or we must support it internally. It is important to realize that we were not designing a research project. Project Andrew, for example, which was a research project, took three or four years of determined effort to stabilize after the initial rollout. We did not have this luxury.

Global Usage – Many of our users travel extensively between offices for their work. We designed the system based on the concept that wherever a user went, when she sat down at an available workstation and logged in, it would be her environment. If a trader in New York jumped on a plane to Singapore and logged in, all his files, the programs that he normally ran and the data he normally accessed would appear with no operator intervention and minimal performance degradation. There actually are regulatory and contractual restrictions on what and where users can access data and programs, however there are mechanisms other than visibility for enforcing these restrictions.

In order to manage a global environment of this size with the small number of people we have in our operations staff, we had to provide a single operating environment worldwide. The sacrifice that we made in doing so was to place restrictions on developers and business units about what they could place into the environment, and what they had to go through the operations staff to do. However, with proper hooks to allow for customization, we felt that this would not be an onerous burden in relation to the economies of scale that this scheme would provide us.

Design Focus

Rather than merely merging the existing FIR and IS systems, the CIG decided to provide a necessary level of interoperability for the short term, and focus on creating a new system from scratch, using the best proven technology available. We used the opportunity to throw out many of the bad features and unused functionality of the old system. We did not provide backwards compatibility by default, we provided it only if it was proven that it was required functionality that was not provided in the new system. We would take the best of the old systems and the best that the marketplace had to offer. For no particular reason, we named the system "Aurora".

In designing each aspect of the environment, we always kept in mind the 4 Rs: Redundancy, Reliability, Recoverability and Reproducibility. In addition, before we built something ourselves, we surveyed the market to determine if a product was available that met our needs. Unfortunately, in far too many cases, the products either were not close enough to our needs, or required significant local customization. If a vendor's product provided functionality that was close to meeting our requirements, we attempted to use Morgan Stanley's size, buying power and clout to encourage the vendor to modify the product to meet our needs.

Finally, it was important that the system we designed was abstract enough to support multiple hardware architectures, yet still provided an interface specific enough so that developers could create both system and user applications that run on all Aurora platforms.

System Design

There are four key components to Aurora:

1. Global Look and Feel (Global Desk)
2. Global File System (AFS)
3. Global Configuration Database (DSDB)
4. Global Homogeneous OS Configuration

Global Desk

Choosing the Window Manager

The change that was most visible to users was the replacement of the window manager on the desktop. Before Aurora, different groups were using different window managers, including olvwm, mwm, twm, tvtwm, fewm and others. In order to avoid having to modify the user's window manager again for a long time, we peered into the future and decided to go with the Common Desktop Environment (CDE), which appeared to have enough vendor acceptance behind it to win the desktop manager wars. However, CDE, as it was implemented at the time, did not have sufficient functionality for our environment. In particular, it had the following drawbacks:

- No support for multiple displays.
  In our information intensive business, our users need to be able to see as much

information as possible. As a result, a large percentage of the desktop workstations have between 2 and 4 displays.

- Inadequate Workspace Management
  CDE did not have a graphical window manager. We have users that actually make use of up to 60 workspaces. Navigating through windows is done via toggles that move either one window forward or one back. While this may be sufficient with a small number of workspaces, it is not adequate for a large number of workspaces.

In order to address these shortcomings, we partnered with TriTeal, the company mainly responsible for the development of CDE. At our encouragement, they added the following major features to CDE that we felt were required (along with many other minor mods and bug fixes):

- Support for multiple displays, including support for multiple control panels
- Graphical Workspace Manager, similar to the workspace manager found in olvwm
- Modifying workspace buttons, so that if a user has 60 workspaces, he doesn't need 60 buttons
- Enhancements to assist non-ICCCM-compliant applications

One of our key requirements with TriTeal is that anything they develop that is not explicitly Morgan Stanley specific would be rolled back into their product. We did not want to be stuck with a "consulting special" that would cause us to be unable to upgrade to new versions, and we did not want to have to support the product ourselves.

X Window Server

Until recently, we were using X11 compiled from MIT sources, rather than relying on the vendors to provide us with a working version. We did this because the vendors generally lagged behind MIT by a few years, and when they did implement a server, it was slower and buggier than MIT's. However, in the last few years, the vendors' implementation have gotten much better, to the point where they provide significant benefits over the MIT server. In particular, they provide real customer support, as well as support for all devices they sell and Display Postscript. As such, for Aurora we have decided to use the vendors' native X11R5 server.

Rollout

Because this was such a visible change, we decided to split the rollout of Global Desk from the rest of Aurora and implement it in advance of the rest of the Aurora changes. In doing so, we believed that when we converted users from their old IS or FIR system to Aurora, they would not even notice a change. We also took advantage of the window system rollout to upgrade all users to use a new standard profile system.

Profile System

The profile system was one of the pieces that we reused from the pre-Aurora environment. It allows us to set up users with standard sets of configuration files, and gives us the ability to easily update users' profiles and ensure that their profiles are correct. By giving the users hooks to allow local customizations, we are providing a commonality and standardization, while ensuring that users will not try to get around the system. The profile system works as follows:

There are a number of base configuration files, like base.profile, base.vuewmrc, base.envfile, etc. These files are common for all users. In addition, there are "model" files. These are typically set up by business function. The files are located in a central directory, and are set up via symlink to the users' home directories. This allows us to make global changes easily, without editing files in each user's home directory. For example, the "Governments Trading Desk" model might set up paths, environment variables and window system configuration to include all the programs that they use. The "Sysadmin" model on the other hand, might have all the "etc" directories in its path. In addition to the common files and the mode files, there is a .custom directory that may contain optional override files.

There are three commands available: installprofiles, checkprofiles and restoreprofiles. A user is set up with the installprofiles command. This saves all old profile information (which can later be recovered using restoreprofiles), and installs symlinks or copies of files from the central installation directory. The files that are installed contain references to other files in a model directory, and to files in the user's $HOME/.custom directory. The model of the user is determined by the contents of the $HOME/.model file. In addition to static files tat are copied or symlinked from the common location, there are "action files", that specify programs to run to populate other files in the user's home directory. For instance, the ".printer" file, which contains the name of the default printer, is generated by picking a printer based on the location of the user as obtained from the corporate database. However if that file already exists, the program assumes the user does not want it changed, and just leaves it.

The checkprofiles command compares the contents of the user's configuration files to the contents of the common configuration file. When a user calls the help desk with a problem, one of the first things the support personnel do is run a checkprofiles. If the contents differ, the user can reinstall the standard profiles, and check for the problem again. If the problem is gone, we have significantly reduced the problem set. In some instances, we may even tell the user that it is his own problem, and he must fix it himself.

This model provides a powerful resource for easily making global or departmental changes without having to run around and modify files in every directory.

Global Filesystem Project

Limitations of the Pre-Aurora Environment

Our pre-Aurora distributed filesystem environment is built entirely with NFS. We have gone to great lengths with the free-ware automounter, amd, to try to maximize the functionality provided by the underlying NFS technology, but it has, at best, several limitations:

- Redundancy/Replication
  Replication is obtained by using locally developed scripts which use rdist and track to force remote copies of "replicated" data to remain in sync with a master copy. Redundancy is obtained by configuring amd to mount key filesystems from one of potentially many sources. Regardless, when an NFS mounted filesystem hangs, it hangs. If you are paging off of an NFS mount, there is no transparent fall over mechanism to allow use of available backup filesystems – you core dump or hang.
- Building-wide shared namespace
  NFS does not perform well enough to use reliably over low speed WAN links, so practically speaking, a building-wide namespace is the best we can do.
- Network Intensive
  NFS caching is minimal, in core memory, and has no consistency. Heavy use of filesystems over the network results in heavy use of the network. The above limitations had wide ranging implications on how the pre-Aurora environment was built, such as how many server were necessary, how many copies of data were required, etc.

Design Goals

The features we wanted to have in the Aurora network filesystem included the following:

- Better Redundancy and Automated Replication
- Global access to shared files
- More efficient use of the network
- Better Security mechanisms

File System Selection

There are not too many options to choose from to meet the above requirements. NFS over TCP and CacheFS are solutions which do not meet all of our requirements and are not available on all the platforms we need to support. DFS would appear to be a possibly better choice, but the technology is not yet mature enough for use in a production environment.

AFS is currently the only production, supported distributed filesystem technology available meets a significant portion of our requirements. Some of the features of AFS which are key reasons why it was chosen are:

- Local Disk Cache
- Guaranteed Cache Consistency
- Logical Volume Management
- Automated Data Replication
- Transparently Available Redundant Data
- Superior Performance over WAN links

AFS is not the perfect solution for use as a Global Filesystem, however. There are a number of problems we have encountered, most of which we have been able to work around, and some have introduced new constraints on the design of the environment:

- One Global Cell
  The Ubik protocol used by the AFS database server does not scale well enough to allow us to have one global cell, covering more than 30 interconnected offices. This restricts us to having one cell per building.
- Inter-cell Data Distribution
  AFS provides a mechanism for replicating data within a cell, but there is no mechanism for distributing replicated data between different cells. We have had to develop a distribution mechanism from scratch internally.
- Kerberos Support
  We already used Kerberos on our systems. The version that we use, however, was different than the version required by AFS. As a result, we had to provide bridging mechanisms between the two systems.
- Sparse File Support
  There is none. Sparse files in non-replicated volumes work by accident, but sparse files in replicated volumes do not. This precludes use of AFS by a large class of internal data files, which will have to remain in NFS until we have true sparse file support in AFS, or eventually DFS.
- Byte-Range Locking
  File level locking is supported, but not byte-level range locking. This prevents many PC-based application from using AFS for user-data files, since use of byte-range locking is so prevalent in that environment.
- Backup Systems
  The AFS backup system is very weak, and third party support for AFS backups was non-existent. Early attempts to restore huge amounts of data have been very disappointing. We partnered with another vendor (Boxhill) and had them write a module (vosasm) that interfaces with Legato's Networker to provide volume-level backups. However the technology is still far from optimal.
- No Per File Permissions
  The file permission semantics change significantly between UFS and AFS. While

there are more options for directory permissions, there are no per file permissions.

- Significant Departure from UFS Semantics
  Error checking write() is not good enough anymore; you have to check close(). This change requires coding changes to bullet proof some applications, many of which we do not have control over (third party applications). The behavior of mmap()ed file is also significantly changed, so migration of data and user applications to AFS is not trivial in some cases.

Global Virtual Cell

The goal of a single, globally consistent view of a shared filesystem is still possible, even within the limitations of the technology provided by AFS. Although the granularity of individual cells is at the building level, this is hidden in the user view of the filesystem. Users will almost always access data through our "canonical name space", not being aware that some files come from the local cell, while others come from foreign cells.

Inter-cell access is provided by mount points for each cell, collected under /ms/.global; see Figure 1. The two character names refer to our locations, for example tk is Tokyo, ln is London, etc.

The pathnames used by users, applications, etc, reference a canonical pathname space. The top level view is shown in Figure 2. The four directories dev, dist, group and user make up the canonical namespace. A user home directory, e.g., /ms/user/w/wpm, is just a pointer into the global namespace to the actual location of the volume for that user; see Figure 3. Thus, wpm's home directory becomes transparently relocatable from cell to cell, without requiring changes to the canonical pathname to his home directory.

The same approach has been taken for the "dev" (Development Volumes, e.g., source code) and "group" (Shared Group Volumes), and together these 3 classes of data cover all the non-replicated RW data we maintain in AFS.

The final class of data, distributed replicated data, is available under /ms/dist, which consists of mount points for volumes assumed to be obtained from the local cell. Our internally developed volume distribution system automates the task of actually replicating the data between cells.

Real-time/Batch Auditing

There are no built-in mechanisms for auditing the state of AFS fileservers. In order to support AFS in a production environment we have had to spend a significant amount of time developing software to audit the state of AFS, both in real-time (monitoring AFS server process error logs) and in batch.

```
% fs lsm /ms/.global/*
'/ms/.global/bk.a' is a mount point for volume '#a.bk.ms.com:ms.cell'
'/ms/.global/ln.a' is a mount point for volume '#a.ln.ms.com:ms.cell'
'/ms/.global/ex.a' is a mount point for volume '#a.ex.ms.com:ms.cell'
'/ms/.global/mg.a' is a mount point for volume '#a.mg.ms.com:ms.cell'
'/ms/.global/tk.a' is a mount point for volume '#a.tk.ms.com:ms.cell'
'/ms/.global/sa.a' is a mount point for volume '#a.sa.ms.com:ms.cell'
```

Figure 1: Mount points

```
% ls -al /ms
total 17
drwxr-x 2     afsadmin 2048 Dec 19 1994  .
drwxr-xr-x 16 root      512 Jul 18 03:08 ..
drwxr-xr-x 2  afsadmin 2048 May 13 00:28 .global
drwxr-xr-x 2  afsadmin 2048 Jan  5 1995  .local
drwxr-xr-x 68 afsadmin 4096 Jul 17 13:06 dev
drwxr-xr-x 2  afsadmin 2048 Jul 10 14:31 dist
drwxr-xr-x 4  afsadmin 2048 Jun 22 18:02 group
drwxr-xr-x 29 afsadmin 2048 Jul  5 10:53 user
```

Figure 2: Top level view

```
% ls -al /ms/user/w/wpm
lrwxr-xr-x 1 afsadmin 27 Feb 10 10:56 /ms/user/w/wpm ->
                                    /ms/.global/sa.a/user/w/wpm
```

Figure 3: Global namespace pointer

Volume Management System (VMS)

In a multi cell environment such as ours, we are required to replicate data between cells. We developed a system to automate the distribution of data, using incremental vos dump/restore technology, and a configuration database to maintain timestamp information and master/slave volume relationships.

Canonical vs. Distributed Volumes

We have introduced the concept of a "canonical" volume, which is the master source volume for the "distributed" copies maintained in each cell globally. Changes are made to the canonical volume, and then incrementally dump/restored to the distributed volumes in each cell. The distribution mechanism works by dumping the backup copy of the canonical volume to a file, and forking multiple "vos restore" commands in parallel to all the cells.

Incremental Propagation

Incremental propagation is accomplished by managing the timestamps in a database external to the filesystem. The act of updating backup volumes or moving distributed volumes from server to server will invalidate the Last Update timestamps on the volumes themselves. Thus, when a volume is released for distribution the timestamp from the backup copy of the canonical volume is updated in the database.

Once the distributed volume in a given cell has been successfully updated, this same timestamp is updated in a separate table in the database for this volume in the given cell. Upon subsequent releases of a given volume, the timestamp for each separate distributed volume is the time from which an incremental dump of the canonical must be performed to bring it up to date.

Authenticated Access to Privileged Commands

In our pre-Aurora NFS-based environment, it was easy to delegate permission to distribute and maintain a portion of our distribution tree to non-root users, as rdist requires no special root privileges to distribute files from to server to server. AFS requires special privileges to dump/restore volumes.

VMS uses Kerberos mutual authentication to determine the identity of the user, and then performs various restricted operations on the user's behalf, using the user identity as the key to lookup authorized operations. This mechanism allows development groups to create new AFS volumes for development of a new release of an application without the necessity of system administrator intervention.

Our goal is to automate all the normal operational procedures for AFS which require special administrative privileges. Most of the processes which are automated have multiple steps and are very error prone when performed by human beings, even experts. VMS automates these steps, reducing operator error, and eliminating the need for giving junior administrators special privileges.

Growing Pains

Although the implementation of AFS for our Global File System is well on its way to success, it has not been without its problems.

WAN Issues

Filesystems across WAN links have historically been forbidden because of the ease of inducing heavy use of the WAN and the poor behavior of the filesystem technology (NFS). Use of AFS over the WAN is significantly better, given the behavior of the RX protocol under high retransmission scenarios. But 10MB of data over a 64KB link is still a heavy load regardless of the efficiency of the transfer mechanism.

We have to be very careful to ensure that access of non-replicated globally available data is minimized, since we have market data critical to our traders flowing over the WAN. Use of technology such as Cisco's custom queueing helps to minimize the impact of the problem, but a loaded WAN link still needs to be avoided if possible. This will prove to be a challenge as production usage of non-replicated data increases. Tools for analyzing WAN access and pinpointing the culprits during a saturation condition have not yet been deployed.

Training and Support

AFS is a radical new technology from the point of view of support personnel. Learning to manage the filesystem, debug and analyze problems, etc. requires understanding a new set of technology and tools. Training has been and continues to be a challenge, as we struggle to get existing administrators up to speed on both the vendor provided technology and the system we have developed internally to implement the global filesystem.

Current Status of the Global Filesystem

We currently have 26 buildings (i.e., potential cells) globally with UNIX hosts of some kind, and by the end of 1996 will have AFS available in all of them. The size of these installations ranges from 3 or 4 hosts to over 2000, and thus the server infrastructure varies from site to site.

In the larger cells, we have installed dedicated AFS file and database servers. In medium sized cells, we have dedicated servers performing both file and database server functionality. In smaller sites, existing servers are simply having additional disk space install for AFS service, and these servers will share functionality with other CPU and database functions.

We currently have approximately 20 GB of replicated data in AFS, and over 200GB of read/write user data (i.e., source code, user home directories, and group directories). We expect the amount of replicated data to grow steadily, but non-replicated data growth could potentially be explosive, with as much as a Terabyte on line by the end of the year.

Distributed Systems Database (DSDB)

Design Goals

As anyone who has attempted to manage a large set of systems centrally knows, there is an absolute need for a central repository of configuration information. This includes the traditional user and host information, but also extends to machine information, configuration files, software information and a legion of other information. In addition, the configuration database must provide significant dependency checking, so that, for example, a user cannot be deleted until all groups and mail groups that he owns are gone, any projects that he has responsibility for are reassigned, etc. The Global Configuration Database is part of virtually every aspect of the system.

Before attempting to build a system from scratch we surveyed the marketplace for possible solutions. Unfortunately, most products address environments in which there is only minimal central management, with many departments wanting their own autonomy. As mentioned earlier, we solved this problem using political, rather than technical means. We felt that allowing multiple administrative domains introduces the possibility, even likelihood, of local configuration changes. Our goal was to maximize homogeneity and minimize local changes. These products are designed to optimize for local customization.

We also felt that these products focus on allowing distributed autonomy meant that the functionality to allow total central administration suffered. Additionally, we wanted whatever system we used to tie in seamlessly to the other databases that were scattered around the company, such as the Human Resources and Inventory databases. Since we had already built databases like this in our pre-Aurora systems, we decided that we had the experience and knowledge to do a better job of building this database ourselves.

The goal of the system that we built, the Distributed Systems Database (DSDB) was to create a configuration database so that, in the event that we lost every building at Morgan Stanley, we could rebuild the entire physical environment with a backup of the DSDB and a lot of money. Note that this does not mean user data, only system configuration information. All system configuration information is primarily stored in DSDB, and only derived on the system itself.

Additionally, the database that was designed is a configuration database only. There is no real-time access component of it. For example, the source for the NIS maps are maintained in DSDB, but there is a separate process that dumps the information from DSDB to NIS, and NIS handles the real-time lookup queries.

In the pre-Aurora environment, we had two databases of information that contained information in the NIS maps and machine configuration information. However as we grew, we discovered some major problems with them:

- They were two separate, non-interacting systems
- There was very little consistency checking. The checking that was done was typically only on the input side. When someone deleted a user, there was no checks to see if that user was in other groups, mailgroups, etc. As a result, there was a lot of old cruft lying around.
- Because of the size of our maps, changes took over an hour to dump from the database and propagate worldwide.
- They were not designed to scale as much we were scaling our environment.
- One of the databases was based on a lightweight, locally developed database engine that was very flexible, but in which complex queries took a long time

DSDB was designed to replace both these databases and provide significant added functionality. DSDB was designed to:

- Provide extremely strong consistency checking
  You cannot add objects that do not fall into pre-specified criteria (for instance, you cannot add a host entry unless the network already exists) and you cannot delete an object that is referenced by anything else (you cannot delete a host for which there is an fstab entry in the database).
- Interact with other internal databases
  Other organizations had databases that were primary sources of information of information, such as the Human Resources database and the Inventory database
- Ensure information exists in only one place
  When a user changes their name, we no longer have to update it in dozens of places. They submit the appropriate form to the human resources department, and a short time later, the Gecos Field in the password file reflects the change.
- Complete historical information
  Information in the database is never deleted, it is aged. Using this, we can
  - get a snapshot of our environment at any given point in time

- trace who did what to the environment
- Provide incremental propagation
  We can request all changes from the database from a given time. This allowed us to implement incremental propagation, which means that we now propagate all NIS changes worldwide within minutes, instead of days.
- All data is world readable (internally)
  There is no private data in the database. Based on the nature of the data that we were storing, we felt that this allowed us to easily avoid complex security issues.

In additional to the standard NIS information, we keep virtually all configuration information in this database. Some examples:

- fstab file information
- processes to start when a machine is rebooted
- machine configuration information, such as the devices on the system, the names of the mounted file system, etc

Primary vs. Secondary Information

There are two types of information kept in DSDB, primary and secondary. Primary information is that information for which DSDB is the primary source, such as the data in the password and host files. Secondary information is that information for which DSDB is simply a repository for ease of searching and reporting, such as the amount of memory on a machine, or the type of the graphic card, or even the user's real name (for which the human resources database is the primary source).

Primary information is typically entered manually as the user or machine or service or whatever is added. Secondary information is usually loaded in an automated, batch format. There are nightly audits that are run to collect information from all machines and upload this information to the database. This information allows us to create reports of system characterizations and usage.

There are a number of programs that run to get information out of the database. One such program is the NIS updater, which distributes the NIS maps incrementally. Another is the program to update the fstab file on a system after it has rebooted.

Architecture

Because the majority of our pre-existing internal databases are based on Sybase, we based DSDB on Sybase as well. All updates to the information are done through stored procedures. These procedures are responsible for the strong consistency checking.

The history mechanism that we use provides us with a means to do incremental propagation. For the NIS maps, every NIS server is a master. When it retrieves information from the database, it stores a timestamp in the YP_LAST_MODIFIED key of the map. Next time it queries the database, it asks for all changes since that timestamp. The sybase procedures to do this are tuned to make this a very fast operation.

Operating System Configuration

Design Goals

The primary goal of the OS Configuration portion of Aurora is to design an Operating System configuration which allows us to manage easily tens of thousands of hosts spread around the globe. The OS configuration is only concerned with system data, not with the data associated with, for example, user home directories, exported NFS partitions, or Sybase data partitions. We are addressing the files that are required to run the core system – those file under root, /usr, /var, etc.

Homogenous Support for Heterogeneous Hardware

We want to maximize the uniformity of the machine configurations to simplify administration, however we also require support for multiple hardware platforms, simply as a means of leveraging the right hardware for the right job to meet the varying needs of a wide variety of applications. From a hardware and operating system point of view, the system must support heterogeneous hardware platforms, however the operating environment must be as uniform as possible across all architectures.

Homogeneity is not easy to accomplish. Doing so in a heterogeneous environment is extremely hard. However we have shown in our previous environments that creating an environment like this reaps large benefits, including economies of scale and the ability to allow users to use the best tool for the job.

Rapid, Automated Installation and Reconfiguration

Hosts should be easily and rapidly reconfigurable. An installation mechanism which takes two hours to dump data from a CDROM, and then requires an administrator to update several locally maintained files, either from a backup system or from memory, is simply not practical when installing hundreds of machines. Furthermore, if a trader's workstation dies, minutes make a real difference. Our users require us to replace user desktops in under 10 minutes. Server OS reconfiguration should be just as fast, not accounting for possible restoration of data such as Sybase partitions.

Installation of systems are often done by people without the root password, such as electricians. When a machine is removed from the vendor's shipping material, the installer should be able to plug it in, turn it on and wait till he gets the login prompt. The only work a system administrator should do is define the ethernet address in the host configuration database.

Control-Alt-Delete

When there is a problem with a user's desktop, it is not always appropriate to take the time to discover what the problem is. Often, time is of the essence, and the user just wants to get back on-line as quickly as possible. We allow users to do the Unix equivalent of "Control-Alt-Delete". If there is a configuration or disk problem, the user on a Sun, for example, simply types "L1-A", and then "boot net". The machine is completely scrubbed, reinstalled and rebooted in under 10 minutes, with no intervention by an administrator and no root access required. In fact, in Aurora it takes less time to reinstall a system than it does to fsck it's root partition.

In actual practice we discourage use of this feature by end users. The system administrator needs to make the decision that the problem requires a reboot, and then about whether discovering the root cause of the problem is more important than getting the machine back up again quickly.

Support for Machine Models

All customization of machine configuration will be handled via the configuration database, by associating the customizations for a particular type of machine with a "model".

For example, a "desktop" model would be used for most user workstations with graphical displays attached to them. However, the model for a generic headless CPU server in a comm room "server", may differ only sightly from "desktop". The former will support use of mirrored or striped filesystems perhaps, something we don't currently support on the desktop.

A "sybase" model may differ from the generic "server" model by configuring special directories in /var, installing a special version of a kernel (or configuring a special loadable kernel module for use).

An "afsfileserver" model will have a much larger number of locally copied replicated files in order to make the machine as independent as possible on the primary service it is providing.

Upgrade to new technologies

We used SunOS 4.1.3 and AIX 3.2.5 in the pre-Aurora environments, and have taken these operating systems to the limit in many ways. The vendors have begun putting more effort into the new operating systems, and the most modern hardware is only supported on the most recent versions of the operating systems. We chose to implement the OS configuration portion of Aurora entirely on Solaris 2.x and AIX 4.x in order to take advantage of the advanced available in these OS release.

Building from Scratch

We could have saved ourself some time by making use of the binary compatibility modes available in upgrading the operating systems. We made a conscious choice not to do this. Rather, every single program, binary and script was carefully examined, and its existence in Aurora had to be justified. This allowed us to bring forward only those applications, scripts, and methodologies which are required in the new environment, leaving behind a lot of historical infrastructure which is no longer necessary or simply outdated.

Dataless AFS Client Design

Many of the above design goals are met by implementing a dataless AFS client. In this design, local copies of replicated files (e.g., /usr/vice/etc/afsd) are present only in order to bring up afsd, from which point on, everything is accessed from AFS. Every file kept local to the machine has to justify its existence based on this criteria.

Most of the traditional pathnames for entire directories and most configuration files are merely symlinks into AFS. Prior to starting afsd, /ms is a directory which contains a symlinks to /localfs, where the real copies of locally maintained files reside. Once afsd is started, the AFS namespace overlays this, and files are no longer accessed from /localfs.

Minimizing the contents of /localfs is one of the keys to minimizing the installation and reconfiguration time of a host. The amount of data on the current Solaris 2.4 model is less than 20MB. At boot time, all local files are checked against "correct" copies of the file in a central AFS repository. If files are different, new files are copied to the local disk, and a reboot occurs. This allows us to always keep local files up to date.

One of the problems we had to solve with the AFS model was AFS cache initialization time. We use a fixed number of files for afsd start-up, and the creation of 400+ files in a large cache normally takes as much as 30 minutes to complete. This time is reduced to less than a minute by enabling asynchronous I/O during the afsd start-up.

Summary

As of the writing, the environment has reached the following milestones:

AFS

We have rolled out AFS to both the old and new environments. AFS is currently available on about 50% of our machines worldwide. By 12/95, AFS will be available on all systems.

Global Desk

The implementation of Global Desk is complete. Currently, Global Desk is rolled out to 25% of our systems. We expect 100% installation by Q196. The rollout of Global Desk is slowed because each user must be changed and converted to the profile system.

DSDB

DSDB is running for Aurora and in parallel on our old systems, supplying all NIS information. In will be rolled out to all systems by 12/95. New functionality is being continuously added.

OS Configuration

There are currently about 50 "pure" Aurora desktop systems, based on Solaris 2.X. Work to support AIX 4.X is underway. Additionally, we have about a dozen production servers running under Aurora. We expect desktop rollout of the Aurora OS Configuration by Q196.

Author Information

Xev Gittler has been a Unix Systems Administrator/Architect for over 10 years. He received his B.A. in Medieval and Renaissance Studies in 1987. He started administering systems in universities, moved to R&D labs and finally to Wall Street, along the way managing systems from 3-5,000 machines. He co-founded and runs the New York System Administrators Group (NYSA), a local-area group affiliated with SAGE. He has no outside interests other than his wife and his child. He can be reached at xev@morgan.com.

W. Phillip Moore received a B.S. in both Physics and Math from the University of Oregon in 1986, and then entered the Ph.D. program in Physics at The Ohio State University, from where he escaped with an M.S. in 1988, after publicly confessing he was in reality an engineer. He then fled to Osaka, Japan where he spent 4 years as a UNIX/Network Systems Administrator for Matsushita Electric Works. In 1992, he joined Morgan Stanley Japan in Tokyo and worked in the Distributed Systems group, and became a founding member of the Core Infrastructure Group. Phil now works in the Morgan Stanley's New York office and heads the Global Filesystem Project. He can be reached at wpm@morgan.com.

J. Rambhaskar, received his B.E. degree in Mechanical Engineering from Shivaji University, India in 1990. During the Summer of 1991, he join the Systems Group, College of Engineering, Ohio University as a System Administrator. He joined Morgan Stanley as a System Administrator in October 1993. He later joined Morgan Stanley's Core Infrastructure Group in October 1994. Since then he has been involved in the OS Configuration and Global Filesystem projects. He can be reached at jram@morgan.com.

References

James Gettys, "Project Athena", pp. 72-77, *USENIX Conference Proceedings*, 1984.

John H. Howard, "On Overview of the Andrew File System", pp. 23-26, *USENIX Conference Proceedings*, 1988.

Michael Leon Kazar, "Synchronization and Caching Issues in the Andrew File System", pp. 27-36, *USENIX Conference Proceedings*, 1988.

Lessons Learned from Project Athena, Dan E. Geer, Jr. pp. 221-247, *Distributed Computing: Implementation and Management Strategy*, Edited by Raman Khanna, Published by PTR Prentice Hall, 1994.

John Leong, Project Andrew, in *Distributed Computing: Implementation and Management Strategy*, pp 203-220, Edited by Raman Khanna, PTR Prentice Hall, 1994.

Appendix – Details of OS Configuration Design

This section provides the details of how we designed the OS configuration for Solaris 2.X. In particular, it explains the installation process, the run time file system layout, how we update local system files, and how the start-up scripts work.

Installations

The Solaris 2.x Jumpstart installation procedure has grown more manageable since the SunOS 4.x suninstall. It provides pre-configuration information, etc. However Jumpstart does not provide complete automatic configuration. In addition, the overhead on the installation server is high, requiring 200+ MB of UFS disk space, and it does not support client builds across a router.

The design requirements for clients installation specified that the installation time should be under seven minutes, the effort to install a client should be the absolute minimum and the resources required on the installation server should be minimal. Our goal was to allow someone to simply plug the hardware into a power source and the network, and turn it on.

The steps involved in the base Solaris2.x JumpStart (netinstall) are very simple.

1. Client send a rarp request and get the IP Information, then tftpboots inetboot
2. With the IP Information it then contacts the bootparams for root file system information, etc. This root file system entry in bootparams is KVM specific.
3. NFS mounts root and loads up the kernel.
4. The kernel remounts root by contacting bootparams again and starts up init

To achieve our design requirements, we did the following.

1. Fixed rarpd to dynamically assign IP information if no information is available for that host.
2. Fixed inetboot to autodetect KVM values so architecture information is not required in bootparams
3. Patched the NFS kernel module to detect KVM values so that it can do the right thing when it queries bootparams to remount root.
4. Truncate the /export/install (NFS install tree) to a very small distribution, approximately 20 MB per architecture
5. Separate out the installation server functionality into 2 parts
    a. local network functions services like rarp, tftp and bootparams
    b. NFS root filesystem.
6. Dropped pkgadd to install client packages as it was too time consuming. Instead, we created a prototype of the root partition and we use cpio to install it on the new filesystem.

Using this method, a single installation server can install an arbitrary number of clients. The only limitation is the bandwidth between the client and server, and the number of simultaneous installs. The actual installation process is as follows:

1. The machine is powered up, and a network boot is started
2. A rarp request is sent out to the network
3. The dynamic rarpd provides a hostname and ipaddr
4. The machine then tftps the inetboot kernel and runs the inetboot kernel
5. inetboot detects the KVM type and NFS mounts the proper NFS root associated with the KVM type
6. The machine loads the /kernel/unix and all the required kernel modules and starts up init
7. init runs /sbin/rcS and /sbin/rc2
8. /sbin/rc2 starts up AFS and basic NIS services
9. After the basic Services are started up, the model script is run, which formats the boot disk (sd0), sets up the AFS cache and uses

```
exmktaaa root=saaa2:/export/fid/bootnet/sparc.Solaris_2.4
sun4c=saaa2:/export/fid/bootnet/sparc.Solaris_2.4.sun4c
sun4m=saaa2:/export/fid/bootnet/sparc.Solaris_2.4.sun4m
wsmodel=localhost:aurora_install cellname=localhost:a.sa.ms.com
```

Figure 4: Typical bootparam entry

```
% ls -l /usr
lrwxrwxrwx 1 root other 19 May 24 11:47 /usr -> ./ms/dist/sunos.5.4
```

Figure 5: Example symlink during startup

```
% ls -al /ms
total 10
drwxr-xr-x 2 root other 2048 Jun 5 21:41 .
drwxrwxr-x 20 afsadmin sux 2048 Jul 27 15:53 ..
lrwxr-xr-x 1 root other 23 May 9 14:21 dist -> ../localfs/root/ms/dist
```

Figure 6: Contents of /ms before AFS is running

```
% ls -l /ms
total 34
drwxr-xr-x 2 afsadmin root 2048 Dec 19 1994 .
drwxr-xr-x 23 root root 1024 Jul 26 17:18 ..
drwxr-xr-x 2 afsadmin root 2048 May 13 00:28 .global
drwxr-xr-x 2 afsadmin root 2048 Jan 5 1995 .local
drwxr-xr-x 67 afsadmin root 4096 Jul 25 18:44 dev
drwxr-xr-x 2 afsadmin root 2048 Jul 28 14:30 dist
drwxr-xr-x 4 afsadmin root 2048 Jun 22 18:02 group
drwxr-xr-x 29 afsadmin root 2048 Jul 5 10:53 user
```

Figure 7: Contents of /ms after AFS is running

cpio to copy files to the local disk. The machine then reboots from the local disk.

Figure 4 shows a typical bootparam entry. "wsmodel" is the model name for a machine. "cellname" is the AFS cell name to use for installations. It need not be the local cell. "sun4m" and "sun4c" are the NFS roots depending on the KVM.

Boottime and Runtime Layout of the File System

The contents of the root file system on an Aurora workstation is minimal; it only contains files that are required to bring up AFS. The remainder of the contents of the root file system are symlinks back into /ms (which is our AFS mount point). Since /ms is not available before starting AFS, we have a UFS /ms which has symlinks pointing to /localfs. When AFS starts up, these symlinks are hidden from the system; see Figure 5 for an example. Figure 6 illustrates the contents of /ms before AFS is running. /localfs contains files that will be overlaided when AFS is started up. Figure 7 shows the contents of /ms after AFS is running.

Updating Local File Systems

A master copy of all files that are required on the local disk at boot time is kept in an accessible AFS directory. Every time a machine reboots, a script is run to check if there is any file in the repository that differ from the files on the local disk, if new files have been added, or if old files have been removed. In addition to the common repository, there is also a provision for overriding files on particular machines.

The update script takes about thirty seconds to run and makes sure that all files in UFS are identical to the master's. If any changes are applied to the UFS as a result of this script, the machine is rebooted. This allows us to update kernels and other locally required files and ensure that they are always up to date.

Start-up scripts (/etc/rc*)

Start up Scripts needed before AFS are maintained in the UFS. The rest of the start-up scripts are symlinked back into an AFS directory. For example, the SunOS default /sbin/rc2 executes scripts from /etc/rc2.d. In Aurora, /sbin/rc2 runs scripts from /etc/rc2.preafs.d and then from /etc/rc2.d. /etc/rc2.d is a symlink back into AFS. This means maintaining scripts across all machines becomes trivial.

Machine specific start scripts are started out using "start" which is a locally written script which reads from central file and evaluates whether a particular process should be started on a particular machine. By using this mechanism, we avoid the need for different rc scripts on different machines.