Introduction | **Microsoft NOC Structure** | **Mail Archive** | **Project Aladdin** | **Hotmail: Unix to Windows** | **Microsoft Projects Details** | **Misc Documents** | **Hotmail Operators Guide**

Converting a UNIX .COM Site to Windows

Microsoft Internal Distribution

Abstract

This white paper discusses the approach used to convert the Hotmail web server farm from UNIX to Windows 2000, and the reasons the features and techniques were chosen. It will focus primarily on the planners, developers, and system administrators. The purpose of the paper is to provide insight for similar deployments using Windows 2000. We will discuss the techniques from the viewpoint of human engineering as well as software engineering.

Early results from the conversion, which was limited to the front-end web servers, are:

n Windows 2000 provides much better throughput than UNIX.

n Windows 2000 provides slightly better performance than UNIX.

- There is potential, not yet realized, for stability of individual systems to be equal to that of UNIX. The load-balancing technology ensures that the user experience of the service is that stability is as good as it was before the conversion.

- As this paper will show, while the core features of Windows 2000 are able to run the service, its administrative model is not well suited to the conversion.

The observations related here are derived from experience gained at a single site. More work would be needed to establish whether they are representative.

Version History

Version	Date	Author	Comments
Draft 1	8/8/2000	David Brooks	Initial draft for review
Draft 2	8/22/2000	David Brooks	Rewrote abstract; added a section on initial experiences; other updates

Table of Contents

Project Overview

Microsoft acquired Hotmail at the end of 1997 as a going concern. The service's creators had defined a two-layer architecture built around various UNIX systems:

n Front end web servers, built with dual Pentium systems on racked motherboards, running Apache on FreeBSD (a configuration with no need to install licensed software)

- Back end file stores, built with Sun Enterprise 4500 servers, running Solaris 2.6 (Sun's UNIX) and with all user data stored on RAID arrays, accessed using very simple filing semantics

- Incoming mail listeners, built on Sun Sparc 5 processors, and interacting directly with the back end

- Name/password verification engines, build on Enterprise 4500 servers

- Member Directory, built on PCs with NT and SQL

The conversion of the Hotmail web servers to Windows is an ongoing project with several rationales. The team was hoping for better utilization of the existing hardware resources. The superior development and internationalization tools are important. A Microsoft property should eat its own dogfood. Finally, we wished to use the conversion experience as a model for other UNIX conversions that we hope to carry out in the future.

The first phase of the conversion, described here, was limited to the web servers. Appropriate hardware was already in place, and the planning and development staff were confident that they already understood how to perform the conversion successfully.

There were several constraints on the conversion process, which are probably typical of the average Internet site:

- Hotmail has established an 8-week cycle of version upgrades, and there was a desire (and some partner pressure) to keep that cycle going.

- It is essential to keep the service running continuously.

n The staff is small, and there was not an opportunity to add staff.

Critical Features of Hotmail as a .COM Site

We believe Hotmail is instructive as an example of the large Internet server site. It is one of the largest such sites on the planet, so we should be judicious in applying its principles to sites that are comparatively *very* small, and don't have the issues deriving from multiplication of resources.

As stated above, we are concentrating on the front-end web servers. Although some of the following comments are also applicable to the database machines, we will not address them specifically in this paper.

1) Restricted, well-controlled application. The application under UNIX was a collection of CGI programs, serving about 100 distinct URLs, which have been converted to an ISAPI module. The programs are written in C++. The entire application is under the control of one team, and its architecture is well understood by all of the teams (dev, test and operations). Updates are only due to scheduled code releases, or hotfixes. This contrasts with a site like microsoft.com, which has many different authors and continuous updates.

2) Lights-out administration. All the servers are in a controlled facility that may be staffed by contractors, and it should not be necessary for skilled staff to visit the individual machines for any reason. Machines should be self-monitoring, and Operations staff should be able to maintain them remotely using minimal interaction.

3) Multiple identically configured machines. This leads to a need to have all regular system administration functions, including OS and application update, be scripted, rapid, reliable, and non-interactive. There is simply not time for an administrator to interact personally with all machines. A load-balancing mechanism routes customer requests from a virtual address to one of the real servers.

4) System costs suffer multiplicative effects. Adding a VGA monitor or a second NIC to a server, or running a serial cable to it, may be pocket change when applied to a single machine. Purchasing several thousand such devices, however, becomes a significant investment and has to be thoroughly justified.

5) 100% availability. A large Internet site must provide service 24x7. Furthermore, the full capacity should be available all the time. Hotmail's load fluctuates daily according to the time across the US, but not by much; the international usage is high.

6) Simultaneous upgrade. The pervious two points mean that the servers must be upgraded essentially simultaneously, unless some kind of server affinity mechanism can be implemented per user session. Since a typical user interaction involves several clicks, it would not be good for a user to jump backwards and forwards between code releases; the problems would range from inconsistency in style to (apparently) half-implemented new features.

7) No personal machines or accounts. All machines are assumed to be secure because of physical location and electrical isolation. Generally speaking, when an administrator is operating on the server or a scheduled tasks runs, full administrative privileges are given. This increases the danger, but reduces the load required to maintain and synchronize accounts.

8) Remote monitoring. All performance monitoring

is done by querying the server or by automated reports, and monitoring uses the single NIC. In Hotmail's case, there is plenty of spare network headroom on each server for this monitoring not to penalize the primary operation.

9) No architectural limits on growth. An Internet site expects to keep growing, and built-in limits that seem unreasonably high in the early days will one day loom up and need to be fixed, using resources that should be enhancing the site. Hotmail has grown from 9 million accounts when it was acquired by Microsoft, to 100 million in July 2000, without significant changes in the hardware or software architecture.

The final four items are more closely related to Hotmail's architectural choices, but we believe they are representative of the market.

10) Scale-out. The Hotmail website is built from several modules, with each module present in different multiples and able to be scaled out almost indefinitely. In this phase of the project, we are considering the front-end, the web servers that house all of the user interface logic and some parts of the business logic. Among the servers, the majority ("front doors") runs some code in response to each click, and these were the primary targets for conversion. The machines are single-board x86 PCs, moderately powerful, using Apache, running on FreeBSD version 3.0, to deliver content. Fortunately, these servers are good Windows 2000 hosts.

There are also some servers that serve static content and will be almost trivial to convert once the front doors have been converted. Administration of these servers will use the same methodology as the front doors. They also run on FreeBSD, using the server "boa", which is optimized for serving static content rapidly.

11) Configuration conservatism. There are more than

3,000 front door machines, all identically configured. Having the servers essentially identical is important to the operators' ability to administer the site. The approach to the hardware is very conservative: once a hardware configuration is established, it is easy to keep rolling out copies rather than try to qualify a newer model.

This conservatism also applies to the software design. The need to run the project on Internet Time [1] has an impact on this project in several ways: in this case, designers always need to be improving the application and there is little resource left over for redesigning the basic architecture. Furthermore, the various modules of the site are developed independently, creating a force for stability in the internal protocols.

12) Design for stability. Virtually continuous uptime and a consistent response time are crucial. This is achieved by some overcapacity, and highly reliable load-balancing hardware (Cisco Local Director). Local Director is just another module in the scale-out solution.

13) Controlled and understood systems. A fact about UNIX is that it is easy for an administrator to ensure that there are no irrelevant services running. As well as giving the potential for maximizing performance, it is useful to be sure that there are no random TCP/IP or UDP ports open that could be used as a basis for an attack. To some, this transparency is intrinsic to UNIX, but it also comes from a greater familiarity among system administrators with its internal workings.

The headless nature of the systems, and their remote location, have a profound influence on the way the systems are administered. Headless operation means that any direct interaction will be through a remote session (telnet or Terminal Server); nobody will be able to detect an important dialog on the console [2] , and even a bluescreen is not apparent. Remote operation

means that there is a specific cost associated with walking up to the machine. The site is serviced by contractors whose job is mainly limited to replacing failed servers and rebooting on demand; it is possible to attach a monitor and keyboard to a running system, but that is operationally an exception.

Advantages of UNIX

Commonly, although not strictly correctly, the generic term UNIX describes a family of operating systems that are deployed on a variety of systems. Although their internal design may be different, the variants appear to their end-users as the same system, with minor (and annoying) differences in usage. There are two variants in use at Hotmail: FreeBSD, which can be used without license cost and is available in source form, and Solaris, which is bundled with Sun hardware. Linux, which is just another UNIX variant, was not used at Hotmail.

The following sections will examine facts about UNIX (specifically FreeBSD) as they relate to the conversion problem. We also consider Apache as an intrinsic part of the UNIX-based solution, in the same way that IIS is an intrinsic part of Windows 2000 Server.

1) Familiarity. Entrepreneurs in the startup world are generally familiar with one version of UNIX (usually through college education), and training in one easily converts to another. When setting up a new enterprise, it’s easy to work with what you know than to take time investigating the alternatives.

2) Reputation for stability. Both the UNIX kernel, and the design techniques it encourages, are renowned for stability. A system of several thousand servers must run reliably and without intervention to restart failed

systems. For Windows 2000, we must first prove the stability in the same environment, and we must then convince the rest of the world.

Apache is also designed for stability and correctness, rather than breadth of features or high performance demands.

3) FreeBSD is free. Although there are collateral costs (it's not particularly easy to set up) the freedom from license costs is a major consideration, especially for a startup. The free availability of source also means that it can be fairly simple (or it can be very difficult) to make local changes [3] .

4) Easy to minimize. The typical UNIX server is taking care of one task, not acting as a desktop and development platform for a user. It is particularly easy to cut down the load on the system so that only the minimum number of services is running. This reduced complexity aids stability and transparency.

5) Transparent. It's easy to look at a UNIX system and know what is running and why. Although its configuration files may have arcane (and sometimes too-simple) syntax, they are easy to find and change.

6) Preference for text files. Most configuration setups, log files, and so on, are plain text files with reasonably short line lengths. Although this may be marginally detrimental to performance (usually in circumstances where it doesn't matter) it is a powerful approach because a small, familiar set of tools, adapted to working with short text lines, can be used by the administrators for most of their daily tasks. In particular, favorite tools can be used to analyze all the system's log files and error reports.

7) Powerful but simple scripting languages and tools. Again, familiarity and consistency among UNIX implementations is the key. Over the years, UNIX

versions have evolved a good set of single-function commands and shell scripting languages that work well for ad-hoc and automated administration. The shell scripting languages fall just short of being a programming language (they have less power than VBScript or JScript). This may seem to be a disadvantage, but we must remember that operators are not programmers; having to learn a block-structured programming language is a resistance point. Scripts that combine executables into pipelines are simple to build incrementally and experimentally, and even the experienced Hotmail administrators seem to be taking that approach for special purpose scripts (using CMD) rather than authoring with one of the object-oriented scripts.

On the other hand, PERL (another language that has grown organically with a lot of community feedback) is more of a programming than scripting language. It is popular for repeated, automated tasks that can be developed and optimized by senior administrative staff who do have the higher level of programming expertise required.

Problems of Windows

Consider the above list of UNIX strengths to be also a list of Windows weaknesses. However, there are some specific issues that need to be called out.

1) A GUI bias. Windows 2000 server products continue to be designed with the desktop in mind. There are too many functions that are either too difficult or impossible to perform using a text-based interface.

Why is this important? There are several reasons:

- GUI operations are essentially impossible to script. With large numbers of servers, it is impractical to use the GUI to carry out installation tasks or regular maintenance tasks.

- Text-based operations are more versatile; an administrator can usually do more to a system (good and bad) than is provided by the restricted, planned methods using the GUI.

- There is in place at Hotmail an established secure channel into the production system, using a text-based secure shell interface.

- Using a GUI amounts to hiding the true system modifications from the system administrators and operators. UNIX operators like the sense of control that comes from their ability to modify system tables and configuration files more directly.

- Operating a GUI through a slow network connection can be too slow to be useful. Although this is less important, it can still be a consideration when there is a need to administer or diagnose a system through a dialup connection.

There are, indeed, many non-GUI administrative programs provided in the core Windows 2000 product and in the Resource Kit. The problem is that the collection is somewhat arbitrary, incoherent and inconsistent. Programs seem to have been written to fill an immediate need and there is stylistic inconsistency and poor feature coverage.

2) Complexity. A Windows server out of the box is

an elaborate system. Although it performs specific tasks well (such as being a web server) there are many services that have a complex set of dependencies, and it is never clear which ones are necessary and which can be removed to improve the system's efficiency.

3) Obscurity. Some parameters that control the system's operation are hidden and difficult to fully assess. The metabase is an obvious example. The problem here is that is makes the administrator nervous; in a single-function system he wants to be able to understand all of the configuration-related choices that the system is making on his behalf.

4) Resource utilization. It's true that Windows requires a more powerful computer than Linux or FreeBSD. In practice, this is a less important constraint. When you are building a large operation, you will use smaller numbers of relatively powerful systems. The PC systems in use at Hotmail are perfectly capable of running Windows, and the machine's basic power is the same whether it is run with UNIX or Windows. For most of the time, it is only executing application code and most of the extra elaboration is not apparent.

5) Image size. The team was unable to reduce the size of the image below 900MB; Windows contains many complex relationships between pieces, and the team was not able to determine with safety how much could be left out of the image. Although disk space on each server was not an issue, the time taken to image thousands of servers across the internal network was significant. By comparison, the equivalent FreeBSD image size is a few tens of MB.

6) Reboot as an expectation. Windows operations still involves too many reboots. Sometimes they are unnecessary, but operators reboot a system rather than take the time to debug it. For example, a service may be hung, and rather than take the time to find and fix the problem, it is often more convenient to reboot. By

contrast, UNIX administrators are conditioned to quickly identify the failing service and simply restart it; they are helped in this by the greater transparency of UNIX and the small number of interdependencies. Some reboots are demanded by an application installation, and are not strictly necessary.

7) License costs. As we will see when discussing load balancing, the license cost of Windows software is a major consideration when converting from the unencumbered UNIX implementations. Although there were no costs to the Hotmail project, as a Microsoft department, the team did consider the software costs in order to make the conversion a useful model for future customers.

- They used Server in preference to Advanced Server (no features of Advanced Server were necessary).
- They reluctantly used Services for UNIX and Interix, to get access to features that were not adequately provided in Windows. Future releases of Windows will have the features that would make it unnecessary to add those subsystems and avoid their notional cost.
- No business analysis was undertaken to determine whether the benefit of the conversion would outweigh the notional cost of the Windows licenses.

Strengths of Windows

1) Windows has more resources behind its

development. It does have greater complexity than the free UNIX distributions, and used wisely (and with knowledge) that can lead to a more effective solution. For example, IIS is more self-tuning than Apache.

IIS and Windows have many more tuning parameters than Apache and FreeBSD. The problem here is to make them comprehensible to new administrators.

2) The development platform, specifically Visual Studio, is a major advantage. Even before the conversion to Windows was contemplated, Hotmail developers used Visual Studio on NT4 to develop and debug their code. The code was eventually recompiled for UNIX when the first level of testing was complete. There is nothing approaching the power of Visual Studio on any UNIX, let alone the free ones, with the possible exception of the Java development tools.

The superior development platform has also had a positive operational impact in the live site. In the first days of deployment, some server threads went into a CPU-consuming loop. Using Visual Studio, Hotmail developers were able to find the application-level problem in a few minutes. That would have been impossible using UNIX tools.

3) Vastly better monitoring infrastructure. UNIX has some rudimentary event reporting and performance monitoring tools, but nothing to approach the integrated power of the event logging and performance monitoring features. Again, it is necessary to use them wisely; event logging in particular has a human and system overhead that we'll talk about later.

4) Better hardware detection. Setting up UNIX on a new PC is difficult, requiring a more intimate knowledge of how the hardware is built. That's an up-front cost; given the existence of multiple identically configured systems, cloning an established system doesn't present the same problems.

5) Internationalization. The software tools available in Windows to provide multiple localized solutions are far ahead of most UNIX systems.

Hotmail Architectural Decisions

Project constraints

The constraints called out earlier (the 8-week upgrade cycle, the need to keep the service running, and the small number of staff) produced enough pressure on the development and administrative staff that the team agreed to devote one cycle to the platform conversion and not change the application during that time. This allowed the developers and testers to focus on the specific conversion issues. During the conversion, the application itself was the same on both platforms. This means that a user may have successive pages served by either platform, and not notice the difference.

The same constraints led to a desire not to change operational practices without good reason, because of the investment in training staff at all skill levels, and the feeling that the fewer things were changed, the fewer were the potential blocking problems.

Finally, the economic necessity of not adding technical staff to the conversion means that there was no consideration given to major re-architecture of the application.

Installation Methodology Conserved

There is in place a method of remotely bootstrapping a server to a new OS and application suite, and converting one rack (21 machines) in about 20 minutes.

Replicating the installation capability was a goal of the project, and conserving as much as possible of the infrastructure to do it was strongly desired.

Conversion to ISAPI

The web server application suite consists of about 90 different transactions, each corresponding to a click on a web page. Using Apache, each one is implemented as an executable program using the CGI interface, and run in a separate process managed and owned by the web server. Processes are the natural way of encapsulating a single stateless transaction using UNIX.

Converting to Windows, the development team decided not to use the CGI interface to IIS. Creating a new Windows process is more expensive than creating a UNIX process. Instead, the team converted the CGI code to run as an ISAPI application, in which the transactions are processed by code that (in the most basic implementation) runs within the IIS process.

Running in process will be more efficient than running as a CGI, because the process creation overhead is avoided. We could have brought that advantage to UNIX. Apache supports the same concept; the equivalent to an ISAPI filter is called a module. Naturally, we did not waste time building the module implementation just to throw it away.

Conversion from CGI to ISAPI was essentially automated by using a filter that effectively presents the standard CG interface (using data streams and environment variables) to the user code. Because the application code was well written and did not make assumptions about its environment, the major part of the conversion went very smoothly and did not require significant unexpected engineering [4] . There were some intentional pieces of re-engineering:

- The spell, dictionary, and thesaurus functions

were rewritten to use Microsoft technology from Office and Encarta. The UNIX versions use binaries from Merriam Webster. The spellcheck feature is much improved; there are coverage problems with the dictionary data that need to be addressed.

- The SMTP service of IIS was used to handle outgoing mail, replacing a UNIX standard mail service.

- Virus scanning of attachments used an external UNIX utility from McAfee; this was replaced by its NT equivalent.

The most challenging, and anticipated, problem with converting from CGI to ISAPI derives from the forgiving nature of the CGI architecture. Memory leaks, unclosed files and similar problems can be tolerated, because they are automatically cleaned up when the CGI process terminates. Even an occasional abort is tolerated; it results in an invalid page to one customer, but does not usually affect any other part of the system.

By contrast, ISAPI modules share a process with the web server, as do Apache modules. Resource leaks will accumulate, and crashes have the potential to bring down the server (although not the entire service, thanks to load balancing). There are process isolation techniques available in IIS to minimize these problems, but the team decided to use the in-process model for full efficiency. Among the actions taken:

- Use a private heap that is cleared at the end of each web transaction.

- In testing, monitor for resource leaks and fix them.

- Implement an IIS heartbeat monitor that will quickly notice and restart any failed IIS service.

Converting to ASP was not considered. That would

have been a complete rewrite of the application, with no great advantage (Hotmail does not use a WinDNA infrastructure, for example). In fact, the implementation uses some ASP ideas and terms, as much of the user content is determined by template files that look like ASP files, but the interpretation engine is completely homegrown. One motivation for borrowing ASP syntax was to use Microsoft development tools (for example, to aid internationalization).

Load Balancing Technology

Hotmail has a large investment in Cisco Local Director; every web access goes to an LD, which redistributes the load among real servers. Hotmail chose to continue with LD, rather than use the Windows load balancing technology, because the infrastructure was in place and did not need to be reconfigured (reducing the learning curve). Also, LD fits the Hotmail model well; it is possible to place up to 400 servers behind the virtual address, and each Hotmail cluster can have over 300 identically configured servers.

Another major issue is the potential cost. Although Hotmail uses Microsoft software without license fees, we must consider this project as a model for real customers. Use of WLBS requires Advanced Server, but Server provides all the other features used by Hotmail. Using list prices, the cost comparison for a farm of 3500 servers is:

- Using WLBS (hence Advanced Server): $15M+
- Using LD and Server: $6M+

This does not take into account any extra PCs necessary to handle WLBS overhead (administrative, as well as the cycles needed to redirect the load) or the plans by Cisco to further reduce the cost of LD by building it into their network switches.

When considered in the context of a large web farm, WLBS has a serious economic disadvantage that can only be justified by the value of its administrative and monitoring tools. There is considerable competition in the IP load balancing market, which drives costs down; the numbers quoted above are based on the price we paid in mid-1999, around $17,000 per unit. An existing system that has load balancing in place will presumably have adequate tools, so the added value of WLBS, in terms of operational flexibility and superior monitoring, must be considerable if it is to be economically justified.

System Creation, Mastering and Installation

OS installation and configuration

Each of several thousand systems must be converted to the new operating system and application suite, and this process must be carried out while the service is operating, and within a short timespan. Required are a mechanism for packaging the image and a method for delivering it. Among the special requirements:

n Each server already has a name and static IP address; to fit in with existing operating practices and configurations, they should retain the same name and IP address. Using a static address, compared with DHCP, makes system administration simpler and more transparent. A machine’s name relates to its physical position within a cluster.

n It should be possible to convert a machine without physical access.

n It should be possible to revert systems quickly to FreeBSD in case of serious problems with the Windows

conversion.

- Downtime for reboots and service restarts should be minimized.

Several technologies were investigated and rejected. In most cases, there were blocking issues that were seemingly small, but without guarantee of resolution the team had to adopt a method that they could control. Some of the issues were:

- RIS can be used for automatically installing an image from a server when a machine is initially booted. Drawbacks include: physical access is required to the machine (to force a network boot), and the system requires that an IP address be supplied with DHCP (DHCP is not used at Hotmail, because of the requirement for static IP addresses). It was impossible to control the name of the new server as required. In addition RIS was not supported for installing Server, although it was known to work.

- AppCenter is intended for this kind of application. However, the initial release of AppCenter is targeted for small installations. It also lacks some features needed by application installation and update.

- Unattended setup performs a standard installation across the network; because of all the file copying and calculation involved, it is too slow.

The team opted to extend an existing technology, "kickstart". This uses the OS existing on the machine to bootstrap an image, prepared using sysprep, and then run scripts to perform the remaining configuration tasks that need to be carried out after the install. The image copy is sufficiently fast, and the post-install steps are minimal.

IIS configuration

It proves to be difficult to configure IIS in a precisely controlled way. The metabase is obscure and poorly documented, and produced too many surprises. Furthermore, a system created using sysprep does not produce a ready-to-run metabase.

Consequently, it was necessary to construct the metabase by using scripts. The scripts were a mixture of command files that repeatedly call the *mdutil* utility, and some special-purpose pieces of scripting code (VBScript in this case, although any language that supports COM would work). The scripts are run as part of the mini-setup step that follows construction of the operating system on the target computer.

Figuring out the metabase structure, which elements needed to be set, and how to suppress the unwanted elements (for example, the trees defining the default and administration site) was the most complex and error-prone part of the entire setup design. Considerable reverse engineering was necessary. Major improvement is needed in the way the metabase is described to users, and the way that administrators can script the commonest tasks.

Tuning and hardening the system

The task was to tune the system for the best combination of throughput and performance, and also to harden it against attack from outside. This required attention in several areas:

- System configuration, in removing all unnecessary system services and making sure the remaining services are configured as effectively as possible.
- Registry settings for performance and security.
- Metabase settings for performance and security.

The team was unable to find a comprehensive set of published settings that covered the above areas, perhaps because there are so many sets of demands on system configuration in general. However, we feel that configuring a system to be a locked down web server will be a common enough task that it would be useful to establish and publish a set of recommended actions and settings.

Use of Active Directory

Active Directory (AD) is a key addition in Windows 2000, yet it has been difficult to justify its use in the web server farm context.

Users in AD

AD is generally used to manage populations of users and machines. At Hotmail, it is not interesting to use AD to manage customers. User privileges and restrictions are already handled by the Hotmail application code, and there is no concept of granting or restricting access to customers within the Hotmail infrastructure. Furthermore, there is a constantly changing population of many usernames (over 100 million in July, 2000), a size that may be beyond the capabilities of Windows 2000.

The site has users in another sense: administrator accounts that are used to manage the machines by hand or by script. However, all administrators are fully trusted in the system (once they are inside the firewall), and it is normal to allow them to log in with full administrative privileges. This is the equivalent to the UNIX *root* account. It is useful to allow single sign-in, to allow an administrator to move from one machine to the next, and also to add new users at a central point;

however, these needs are easily met by NT4's NTLM.

Computer systems in AD

There is a stronger argument for entering the servers in AD. This will provide integration with DNS, and holds out the potential for administrators to classify machines in whatever ways they find useful operationally.

The Hotmail server farm is organized as a series of clusters, each containing several hundred servers. These machines must be named systematically. In practice, server names are duplicated between clusters, as they are identified uniquely by the fully qualified domain name (each cluster is a subdomain). This presents a problem for AD, which (apparently because of NetBIOS compatibility) does not permit duplicate short names anywhere within a set of subdomains. Getting rid of the NetBIOS legacy will be a great boon for Microsoft.

This apparently trivial restriction was enough to postpone the idea of constructing an AD, which in any case is additional work without obvious benefit. It was necessary to maintain the names of systems through the upgrade, because of legacy monitoring and administrative tools. Existing administrative mechanisms were adapted and did not need the benefits of AD. It is expected that, later, administrative staff will be able to develop tools that can make use of AD (for example, the ability to query on servers with a particular characteristic may be useful) but for now there is no need to break into the circle.

The Windows DNS service, operating without AD, proved perfectly capable of handling the load, and was able to take up the data from a UNIX BIND server easily. Windows DNS is used at the site for both internal and external name resolution.

Application Installation and Update

Application update styles

It is naturally a requirement that a web-based service operate continually, without customer-visible degradations of service. This is not just a matter of pride; even a loss of availability for a few minutes every month can produce too much degradation in the perceptions and (assuming we publish uptime numbers) the availability measurement.

It's a solution, but a weak one, to put servers behind load-balancing equipment and take them out of service when required for upgrade or other maintenance. The challenge is to keep each server running continuously as much as possible. Except for operating system upgrades, a system based on FreeBSD and Apache can keep operating while the application is upgraded, and Windows should be able to do the same.

Application updates at Hotmail are of two kinds: content and code. Content updates change only data files, generally those that directly determine what the customer sees on the screen, and they are carried out on their own schedule. Apache can handle both content and code updates without stopping the service. Updates can be rolled out directly, when the data is updated in place. They can also be timed, when the updates are put on the servers in a staging location together with an update batch job that will be triggered at the desired moment. The timed update is used when it is important for the application's integrity that the entire site be updated simultaneously, something that is impossible to achieve when updating several thousand servers across a single network.

Application update techniques

Apache running under UNIX supports both kinds of updates very simply. A CGI application can be replaced, even while the old file is being executed, and the next execution will use the new file. The same is true of content. If Apache's own configuration files must be updated, there is a procedure to signal the server to reset itself and reread its configuration, and that takes around a second.

Unfortunately, IIS 5.0 does not support either kind of update well. When IIS accesses content directly, it locks the folders. Fortunately, this doesn't apply to most of the Hotmail upgrades. The bigger issue is updating the ISAPI filters, which must be done while the IIS server is stopped. The entire process can take a minute or so.

The Hotmail staff has invented a technique that uses a thin ISAPI filter (the "shim"). It loads the application as a separate DLL and passes on all the ISAPI requests. It also watches for updates to the application DLL in a predetermined place, and when it is notified of an update it maps the new DLL, sends it all new requests, and allows the old requests to terminate before removing the old DLL. This technique has been made available to the IIS team.

Intellimirror

The team investigated, but decided not to use, Intellimirror-based update. First, Intellimorror requires AD to be implemented. Second, Intellimirror (working with the Installer) only makes updates to applications when a user logs in or when the system is rebooted. Since user login is an irrelevant activity in this context, and the whole idea is to prevent a reboot, Intellimirror-based update does not meet the need.

Distribution mechanism and format

The UNIX implementation packaged new code as a compressed file using the UNIX *tar* format, and distributed it (and the necessary installation code) using the UNIX *rdist* utility.

The team investigated use of MS Installer technology for a packaging format. Although it would probably have met the requirements (including the ability to unpack versioned files into specific locations, make registry changes, and run arbitrary code during installation) it proved too difficult to learn, despite the availability of a few decent authoring packages. The team stayed with the zipfile method of packaging.

The UNIX *rdist* mechanism is also well suited to installation and updates on a large number of identical machines. From a central location, the administrator can iterate over a list of servers and push packages to them. The *rdist* daemon (service) running on the remote systems will extract files from the packages into their specified locations and run arbitrary commands before and after installation. This is approximately equivalent to MS Installer features, with the additional ability to push distributions over a list of machines. The Hotmail team implemented a version of the *rdist* daemon to run on Windows.

Monitoring and Logging

Network Operations Center

The Hotmail infrastructure is monitored remotely, in an operations center located with the development staff in the Sunnyvale campus. There are many tools in place to monitor the performance of the server farm. Some of

these measure the systems by their external behavior, and they did not need modification. Others use information gathered by the servers themselves (performance counters, disk statistics and so on). It proved to be relatively simple to write scripts that would extract the desired information from the Windows performance counters and send them to the Operations consoles.

Autonomous monitoring

Some of the self-test and monitoring features of the servers are performed by customized operations (usually scripts) executed at predetermined intervals. These intervals are anything between a minute and a week.

Using FreeBSD, such tasks are scheduled by the *cron* service. Jobs are scheduled by being listed in a file, one line per job. Changing the file is easy to accomplish using the command line (or *rdist*), and replacing the entire file is a good way to ensure that each server has exactly the schedule of jobs that the administrator intended. Jobs can be scheduled to execute once, or at intervals down to one minute.

Although the Windows Task Scheduler service is fundamentally able to look after such jobs, the interfaces provided in Windows does not measure up to the task.

n The usual interface is the GUI, which is appropriate for setting up jobs on a machine at a time, is labor-intensive and error-prone.

n The command *at* is deprecated, is not able to schedule repeated jobs at a frequency of less than one day.

n The command *jt* was offered by the Task

Scheduler team, but it is unsupported and awkward to use (it was intended for testing).

- None of the three interfaces offers an easy way to replace the current task schedule entirely.

The team met the need by running the *cron* service provided in Services for UNIX. As described earlier, relying on Services for UNIX (or any other package subject to extra license costs) provides a bad model for other customer deployments. We have provided input to the Whistler command line team for an improved interface to Task Scheduler.

Logging

There was a minor issue concerning the UNIX integrated logging feature (*syslog*). The kernel, standard services, and application code can write lines of text to *syslog*, and a single configuration file is used to determine the destination of the text lines. Thus an important alert can result in a console message and email, while an informational message can be written to a log file. The administrator can change the destinations without code having to be recompiled.

An application like Hotmail often uses the application access to *syslog* to write statistical data of business interest (such as creation of a new account or sending of an email message). Administrators can use other tools to analyze the logs, archive them, or simply count occurrences and throw the logs away. Typical usage is at the order of one event per second; the high performance associated with the kernel log is not required.

There are no features in Windows 2000 that provide the same combination of convenience and configurability, although the kernel event log comes close. For convenience, and also to avoid recoding, the team elected to use the *syslog* feature from the Interix

subsystem, introducing the issues about notional cost that have already been discussed.

Whistler introduces the Enterprise Event Log, a lightweight WMI feature, which seems to provide the desired functionality. A closer examination of the kernel logging may show that it too can meet the need, Any replacement should involve trivial change to the existing application code (perhaps even using a macro); it would be desirable not to have to recode calls to *syslog* in order to keep down the amount of source code conversion.

Ad-hoc Maintenance

There are occasions when, after deployment, the administrators need to make a configuration change consistently across the entire farm. The *rdist* mechanism can be used for configuration changes; if the change is simple then *rsh* can be used. The key fact about UNIX that makes this work is, again, that all system administration tasks can be done from the command line.

Windows should provide the same functionality, given some means of aggregating a group of servers and some way of performing an operation consistently across all the servers. Single commands, pipelines, or scripts (command scripts or COM-based scripts) would be appropriate actors; however, scripts need to be downloaded, executed (and, if necessary, cleaned up). There should be the ability to defer the activity until a specific time, presumably using the improved Task Scheduler. In other words, Windows must support old-fashioned batch processing.

One specific example of a feature that is not accessible

to the batch model is Network Interface Card customization; for example, there have been requirements to change the card's speed from 10 Mbps to 100Mbps (at a specific time) or to change the MTU setting. The configuration model of an Ethernet NIC varies between manufacturers, and the standard GUI is driven by a schema that is found in the registry. Such a GUI is not at all adaptable to the batch model. It is possible to make the required changes to the registry, but that would require a subsequent reboot, which is not acceptable. A brief period off the network, while the card resets itself, is the most downtime that can be accepted.

The Hotmail team, with help from a network engineer, developed a rudimentary application that would put a specific value in the registry and (using an undocumented interface) reset the card in a way that will make it pick up the new value. We strongly urge that the feature be put into the shipping system.

Converting the UNIX Administrator

Helping UNIX system administrators with the transition to Windows is an experience in itself, and much was learned. Again, this is data from a single corporate experience, but we suspect it is fairly typical. Here, then, is the human engineering overview.

Initially, the plan to convert from FreeBSD to Windows was met with responses ranging from skepticism to hostility, in a way that should be familiar to those who share the attitudes of the various UNIX communities to Microsoft software.

We engaged with the operations staff by asking them to define what their everyday tasks are, in all areas of

operating system and application maintenance. Instead of a set of tasks, we were handed a set of the UNIX commands and features that were used to carry out those tasks. While this did not directly meet the need, it gave us an opportunity to address all of the features directly, and show that Windows has an exact equivalent in the core system, or in the Resource Kit, or easily provided with a script. There were very few cases where no satisfactory alternative could be found. Essentially, this was throwaway work, as the eventual solution solved the problems in a more Windows-like way, but it was an excellent opportunity to gain the confidence of the operations staff.

It was clear from the responses that some people from the UNIX side of the house cannot distinguish our different systems that are marketed under the Windows brand; there was an inbuilt assumption that Windows 2000 shares the features and faults of Windows 95. Those who were somewhat familiar with Windows NT were not aware of the range of the non-GUI offerings (to be fair, neither were we); the set of commands in the product and the Resource Kit is fairly broad although, as we have seen, there are gaps and they lack stylistic consistency.

Other staff members, not members of the regular operations team, carried out the conversion. When deployment came near, the Operations staff had to learn the new tools and paradigms. Their existence proved enough; the main interest of Operations staff is, after all, to run and administer the system, and once they found that there were tools, whether custom-built or standard, that did the job well enough, they were able to take control and gain a sense of ownership. Some standard one-day courses were also given to the staff, to prepare them for handling system debugging, hotfix application and so on. By this time, the staff had become convinced that Windows is, after all, a real operating system with surprising richness.

To summarize:

n Make the message clear: Windows 2000 is a modern operating system; it's not Windows 9x.

n Gain the confidence of the skeptics by showing them that it is a real operating system, and not so different from other operating systems in fundamental ways, by showing some basic command-line tools that monitor the system in action.

n Use the self-interest of operations staff to ensure that they have full authority over their areas of responsibility.

Conclusions

These are the main lessons that we can extract from the Hotmail conversion.

1) We need a consistent, comprehensive, thoughtful approach to integrated management of a set of servers. This does not necessarily mean that we should slavishly follow the UNIX model of iterating through a list of machines with an *rsh* command, or pushing configuration files to a list of machines. The fundamental goal is to be able to manage machines as an aggregate; doing this through a GUI is not necessarily evil, so long as it can be done remotely, and once. The point applies to application distribution as well as to system tuning.

2) NLBS is at an economic disadvantage, due to its association with Advanced Server, and Hotmail operations staff were sufficiently satisfied with the existing solution that they did not feel the need to investigate NLBS's operational advantages.

3) The metabase needs to be ripped out and replaced

with something that is much easier for an administrator to see and understand, and be confident that there are no hidden surprises. The IIS6 planners have heard this opinion.

4) It should be easier to tune and lock down a single system, and have the changes propagated to all systems in a given class.

5) Windows is too complex to understand at first, particularly during a conversion from UNIX. There are just too many things about it for a planner in a startup to understand. Typically there is little time to attend training. The problem is most Computer Science graduates come to their startups already understanding enough about UNIX to be confident that they can use it effectively. We do need to be careful to balance the complexity and transparency carefully.

6) The basic need for an Internet site, converting from UNIX to Windows, is to be able to quickly replace their application and operational methodology with something at least equally good. Improvements that come for free are good, but implementing new technologies and programming methods will need to take a back seat so long as they delay the main purpose, which is to keep a site online and competitive. Anything else is a cost that needs to bring a clear benefit.

---

[1] Use of this term can be something of a conceit; here we mean that there are economic pressures to roll out new versions of the application on a rapid cycle, typically 8 weeks. This means the team is constantly in "ship mode". In addition, new systems are built out barely ahead of the demand.

[2] Whistler will enable a Terminal Server user to log

on to the console, but as we see later Terminal Server is not an ideal solution.

[3] For example, there was a need to reduce the MTU parameter of the TCP/IP interface. There was no command available to make the change, but the code in the network stack was easy to find, modify (one line) and rebuild.

[4] One notable exception: the Windows library call to perform case-independent string matching was found to be unexpectedly inefficient, presumably because of the internationalization concerns that are not present on the simpler UNIX systems.

Introduction | Microsoft NOC Structure | Mail Archive | Project Aladdin | Hotmail: Unix to Windows | Microsoft Projects Details | Misc Documents | Hotmail Operators Guide