



The following paper was originally published in the
Proceedings of the Large Installation System Administration of Windows NT Conference
Seattle, Washington, August 5–8, 1998

A Comparison of Large-Scale Software Installation Methods on NT and UNIX

Michail Gomberg, Rémy Evard, & Craig Stacey
Argonne National Laboratory

For more information about USENIX Association contact:

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

A Comparison of Large-Scale Software Installation Methods on NT and UNIX

Michail Gomberg, Rémy Evard, & Craig Stacey

Mathematics and Computer Science Division, Argonne National Laboratory

Abstract

Our computing environment consists of hundreds of UNIX and NT-based computers. We have a coherent UNIX software installation model that scales comfortably to hundreds of machines. We have spent a great deal of time in the last year learning to understand the software installation and support mechanisms for a similarly large NT infrastructure. In this paper, we examine the underlying requirements for large-scale software installation support, and compare and contrast the NT and UNIX environments, identifying strong points, weak points, and issues.

1. Introduction

The computing environment in the Mathematics and Computer Science Division of Argonne National Laboratory consists of nearly a thousand computers, including supercomputers, servers, workstations, desktop machines, and laptops. The majority of these are running UNIX, but the number of NT machines is steadily growing, particularly on desktops.

This set of machines is managed by a group of seven system administrators, meaning that, like most systems administration groups, we have more than enough work to go around. Our ability to administer the environment relies on scalable techniques. We have to manage large sets of machine as if they were one single system, most preferably from one single location. We were originally nervous about our ability to do this in a growing Windows NT environment due to its reputation as a system requiring hands-on administration. Software installation was one such area that we were worried about.

On each of our UNIX architectures, including SunOS, Solaris, Linux, FreeBSD, AIX, and IRIX installations, we use a common method of installing software. It scales well, in that we are able to install a given piece of software once and have it automatically work on every machine of a given type. This system is described in detail below.

As we began to look at software installation on NT in detail, we began to feel that our fears were justified. The usual software installation methods on NT require the administrator to sit in front of an individual machine, answer questions interactively, wait for the software to load, possibly reboot the machine, and then do a lot of tweaking in order to make the software available to other people who might log in to the machine. This approach doesn't scale to hundreds of machines. Fortunately, there are ways around most of these problems, but, as described below, they are not uniformly applicable to all NT applications. As a result, we started a long-term project to understand NT application installation better, to test the various installation tools, and to develop a philosophy and method of software installation that scaled as well as the methods we use on our UNIX machines. Ideally, we would learn some new approaches with NT that we could then apply to our UNIX systems to help some rising problems on those systems.

In this paper, we describe the software installation methods that we use on our UNIX machines, because they provide a context for our desired result. We explain what we have learned about NT applications and why we feel they should be considered differently than typical UNIX applications. Finally, we describe the system that we have begun to use, explain how it helps us, and identify remaining problems.

This is an interesting time for NT administrators because the range of tools is changing so quickly, due primarily to Microsoft's growing effort to help solve scalable administration problems. Many of the tools that we are now using didn't exist when we started this project, and most likely, some tools will be released between the time this paper was written and the time it will be presented. However, we believe that our desired solution, our observations on applications, and our current approach will still be of use to other administrators who have to surmount similar problems.

2. UNIX Software Installation

When we started our NT application installation project, we initially looked at our existing infrastructure to identify the components that we felt worked well for us. These helped shape our concept of an ideal NT solution. We present a brief overview of our UNIX software installation here.

All of our UNIX machines are interconnected over our internal LAN. We manage several different flavors of UNIX, but we have the same software installation methods for each of them, so we will only discuss the Solaris implementation in detail. Although we don't use Depot [1] because it doesn't solve all of our problems, any readers familiar with that software installation method will recognize the concepts presented here.

The software installed on our Solaris machines can be broken into two different categories:

1. Software that is installed on the local machine. This includes the operating system, some daemons, and any licensed third party software that has to run on a specific system.
2. Software that is installed on a networked file system. This includes the vast majority of software installed on Solaris, including programs that have graphical user interfaces, application suites, and simple command line utilities.

Locally installed software is usually installed at build time by our Solaris build scripts. If we decide to install a new piece of software locally onto a machine, then we install it on every Solaris machine. This is very important - it is much easier to manage all of our Solaris computers as a single unit if each of them has an identical set of installed software. We modify the build scripts to ensure that new Solaris machines also get this software. As mentioned above, this is a rare occurrence.

Most of the installed software on our Solaris machines is installed into a networked file system. In our case, the path to this networked file system is `/software/solaris`, but this same concept is more commonly available as `/usr/local` at other sites. Each Solaris computer mounts this single network repository. This means that, once a piece of software is installed in `/software/solaris`, every Solaris box immediately has access to it. In the overwhelming majority of cases, no extra work is required on any computer in order to make the software available.

The `/software/solaris` filesystem is exported via NFS. It doesn't deliver software quite as quickly as local disk would, but that has never been an issue. It will scale reasonably to several hundred client machines. Beyond that point, we will replicate the directory and serve it from multiple servers, which will require a minimal amount of work.

One serious problem with this approach is that if the network server goes down, then all of the Solaris hosts are crippled. In our environment of relatively reliable networks and computers and a relatively small number of administrators, we deem this to be an acceptable tradeoff. Further, we replicate the `/software/solaris` file system to a disk on a second server, and have the clients automatically fail over to the second disk when possible. This helps when the server has a problem, but doesn't solve the case of a network outage. In practice, this has not been an issue. (If the network is down, there are usually bigger problems.)

When a piece of software is installed, we put it in its own directory under `/software/solaris`. The path is hardcoded to include the software's category and version number. For example, emacs is in `/software/solaris/apps/packages/emacs-19.34.6/`. All emacs files of this version are stored under this directory. This allows us to install many different versions of emacs without any collisions between them. This is crucial.

Not every package can be easily convinced to live under one directory, but in practice, we have always been able to get around this. UNIX software is typically easy to modify with configurable files, and in the most cases, can be fooled with symbolic links.

All that the user needs to do to access most applications is to put this directory in their PATH environment variable: `/software/solaris/apps/bin`. We populate this directory with symbolic links to the actual executables that live in their own directories, and, using the "soft" [2] system, we modify the user's startup shells to update any environment variables necessary. We can change the default

version of an application with one quick command. Again, we trade off a minor performance hit (symlink resolution) for increased flexibility of administration.

Our UNIX installation is not without problems. As the size and complexity of applications grows larger, and the cost of local disk grows cheaper, we find that we would like to install more applications on individual machines. Doing so is more problematic than installing them centrally. Also, an increasing number of third-party applications have arbitrary restrictions that nearly force us to do by-hand installations on individual machines. Our general feeling is that in these cases, those vendors are borrowing the worst ideas from the world of PC applications.

The key points of our UNIX installation methods are:

- Each machine is identical to every other machine of the same type.
- Most software is installed in a network file system, and in this case, no extra work is required to make software available on all machines.
- All information related to an application version is kept in one place -- the directory for that software.
- No extra work is required to make software available to all users.
- We make heavy use of symbolic links to glue different areas of the file system together into a seamless interface for users.

3. NT Software Installation

One of the hardest tasks that we have with our NT environment is installing software. Some people have expressed surprise at this. After all, they point out, it's much easier to install an application under NT. All you have to do is put the CD in the drive, click setup, and answer some questions as InstallShield goes to work. One would expect that

UNIX software management would be much harder, since there is no uniform way of installing applications, configuring them requires editing various arcane files, and sometimes they even have to be compiled. This is true; if one's goal is to build a single machine for a single user and install a given set of applications, it is much easier to do on NT than on UNIX.

However, managing a large environment for many different users is entirely different from building an isolated machine. Ironically, it is the NT application's focus on the user and on the single machine that primarily makes it so much harder to maintain on a large scale. But this is not the only difference between software on NT and software on UNIX, and not the only reason that large-scale NT software installation is difficult.

In this section, we identify the issues that have made NT application installation challenging in our environment. It is our hope that some of these issues will be addressed in future releases of NT and the various applications. Understanding these issues (and why we consider them to be issues) is also key to understanding how we install applications.

3.1. NT Software is GUI-based

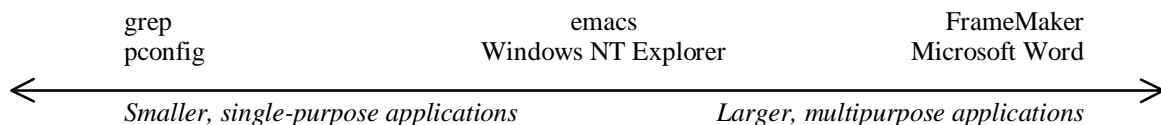
The first difference noted by most people is that NT's applications generally have graphical user interfaces, while UNIX applications may or may not. We find that whether or not an application as a GUI really has no bearing on how difficult it is to install.

A somewhat related issue is whether or not administration tools should be based around GUI. Generally, GUI-based tools are harder to automate and extend. Large environments all have their own particular requirements, so administration tools for them require customization. At the present time, command line tools are easier for us to use when managing a large, diverse environment.

3.2. NT Software is More Complex

Most of the applications that we install on NT have a different focus than those we install on UNIX. To illustrate this point, see the line in Figure One. On the far left-hand side of the line are small, single-

Figure One: Software Continuum



purpose command line utilities such as “cat”, “grep”, and “tracert”. On the far right are large, multipurpose monolithic applications such as “Office 97” and “Internet Explorer”. These applications tend to provide a very large set of features from a single interface. Applications like emacs and perl lie somewhere in between.

The majority of the applications that we install on UNIX are on the left side of the line, while the majority of the applications that we install on Windows NT are on the right side of the line. This is not to say that UNIX has no large, monolithic productivity-based applications. It certainly does. But in our environment, UNIX owns the left side of the line, while Windows NT owns the right side. We believe that the wealth of these types of applications for NT is the primary reason for its steady rise in popularity.

The focus of the program generally does not have an impact on installation. However, the applications on the left side of the line are often easier to install, regardless of their OS. There are several reasons for this.

- Footprint. The smaller the application, the smaller the fixed storage footprint. When applications are stored on network file servers, the application load time becomes an important factor. For small programs this is often not an issue, but for very large ones, the startup delay can be painful. As a result, one leans toward installing the application on a local disk rather than on a network file server, making it much more difficult to install and update.
- Options. Typically, the larger and more complicated an application, the more options it has at install time. And, during the lifetime of the application, the larger, more comprehensive applications seem to want more add-ons and more components. This makes it quite a bit more difficult to get a single installation that will satisfy a large group of users.
- OS integration. Large applications are very nearly operating systems in and of themselves. (Emacs used to be accused of this, but it has been far outclassed by applications in the NT world.) They depend on a large portion of the OS, and they often have a very strong reliance on the OS being set up in an exact manner. On NT, this has resulted in the applications being tightly integrated with the OS. The problems with this under are described in the next section.

It is interesting to note that we are seeing an increasing number of large, GUI-focused

applications available for UNIX. This is one of the areas where we hope that our lessons learned from NT application installation will help us out on the UNIX side.

Also, there are more utility-style programs available for NT, including ports of perl, various UNIX utilities, and a batch of new, NT-specific tools. Unfortunately, it is still difficult to install networked versions of these tools to be shared by all the workstations from a single location, usually because they are too tightly bound to a specific machine.

3.3. NT Software Lives in a Single-User World

Historically, NT applications come from an environment where they could make certain assumptions that are no longer valid:

- The application would only be installed on a single computer.
- Only one user used the computer.
- The application would be installed on local disk.
- The application might have to add things to the operating system.

This set of assumptions, combined with the usual large size and complexity of NT applications, has resulted in a set of conventions for NT application configuration that make large-scale installation particularly difficult.

- NT applications are too intertwined with the OS. Most applications depend on mfc42.dll or shell32.dll to be present on the machine. When these files aren't there or are different from the expected version, software will replace them with the version that they were built with. This can cause other applications to crash or act oddly. Since they also often copy in other DLLs, they tend to bloat system32, making it more difficult to install network versions of the application, and also rather difficult to decide what is safe to uninstall. Also, applications that are written for Windows95 will often try to write to areas that are protected under NT and require Administrator privileges.
- NT applications are too machine-specific. Most applications write their configuration information into the registry. Since the registry is machine-specific, the software is only configured correctly for that one machine. The usual solution, which doesn't scale well, is to copy portions of the registry to other machines. Discovering which portions of the registry to

copy, and doing so without causing side effects, is a challenge. Some software also makes use of a user's environment variables, which, unlike UNIX, are set up on a per machine basis, as part of the local registry.

- NT applications are user-specific. This shows up in two ways. First, when an application is installed, it will often add itself to the current user's Start menu. Other users, if they login, will not be able to run the application, even though it's installed on the machine. This can be fixed by making sure every application is added to the 'All Users' portion of the Start menu. Not all applications are smart enough to do this yet. Second, applications tend to store user-specific data in various places on the local machine, including the registry, user directories, application directories, and even operating system directories. So, for example, one user may login to a particular machine and be able to read some other user's email because the email program wasn't programmed to handle multiple users.
- Roaming profiles aren't quite right. Roaming profiles attempt to solve some of the above problems, but they create new issues. When applications are installed on a single machine, roaming profiles end up pointing to applications that aren't there. If a roaming profile is loaded onto a workstation that has a differing installation of the same app, they munge the correct settings.

Because of these differences, we have to approach NT software support completely differently than UNIX software support. Overcoming the above problems requires special tools, special care, and a lot of planning.

4. Possible Installation Approaches

In order to decide on which installation methods we'd ultimately use, we had to evaluate and consider as many options as we could. Some of these have been used by other sites, some of these we came up with on our own. Our final solution was a mix of these.

4.1. Installation By Hand

The first option is the obvious: by hand. You sit at the machine, pop in the CD-ROM, answer all the questions, wait while it copies files, you answer some more questions, and eventually you're finished. For a minor optimization of this technique, you can

put copies of the CDs or floppies on the network, and install from that.

The by-hand method is usually done by sitting at the console of the machine, but it's also possible to bring up a remote console of the machine using various third party software, including the SMS Help Desk application. Neither of these is what you really want to do, as the by-hand approach almost always involves too much of time overhead. Worse, for each iteration of the install, you introduce the chance for things to change from machine to machine, such as accidentally selecting different options.

We decided that by-hand installs should be avoided if at all possible. (However, if you have a lot of free labor lying around, this may be the easiest solution.)

4.2. Installation at Build Time

For a default set of software that you know will seldom change, arranging for the software to be automatically installed at build time is ideal. This method is automated, it scales, and it remains consistent for each machine. After an automated build with software install, you have a base machine with a known set of software. You know exactly how that machine is configured.

In order to automatically install software at build time, you usually run a command script after the initial OS install, or initiate an automated administrator login that executes a series of install scripts. In our current solution, we ended up doing both.

The first option, running a command script, we use to apply the latest NT service pack and hot-fixes -- admittedly more of an OS issue than a software issue. We could make more use of this, however the system needs to reboot after the application of the service pack, so we turned to the second option.

To use an automated login, it's best to create an account dedicated to that purpose. That way you can disable it when you're not using it, but, more important to the task, you can assign it a specific login script. The script should execute any post-install cleanup that couldn't be done before the first reboot, and then launch an installation script of your choosing, be it a batch file, a perl script, a CMD file, or what have you. Properly done, this can spawn the appropriate package installs, one at a time, and in the order you specify. The order can be important -- for example, getting perl or Winzip on the system first would be prudent as later installs will want to use them.

We experimented with a third option: using the OEM install tools found in the Resource Kit [3], notably sysdiff. With sysdiff, you take a snapshot of a machine before an installation and a snapshot after. Sysdiff records the differences, which, presumably, can then be applied to any other machine to install that same piece of software. We found, however, that sysdiff did not scale well. We started using it when we were first mass-building our NT environment. As is common in these cases, what we thought should be a default build at the start of the project and what we eventually discovered should have been the default build were vastly different, the latter being a much larger set of applications. With each attempt to add an application to the sysdiff package, it took more and more tweaking to the various configuration files to make the creation of the snapshots actually succeed. Sysdiff also had serious problems when our machines weren't compatible at a hardware level. More often than not, as the default build grew, sysdiff would fail miserably, usually with a fatal error at the last step of the procedure, creating the distribution directory.

However, installing software at build time has one serious problem. It only fixes the installation problem once, which is fine if you never expect to install a new application in the future. However, if you do upgrade your software, you have to modify the software that is installed at boot time. This changes the known base, so you are likely to end up with inconsistent machines. Worse, you probably will also have to install this application on machines that were built months before. This essentially puts you back where you started -- trying to figure out how to install software on existing machines.

Installing software at build time, then, is not a complete solution, but it's certainly useful to be able to do it.

4.3. Automated Remote Installation

What you really want is some way of automatically installing software on remote machines. We considered several different methods.

SMS has other uses besides remote control. It also has a feature called the "package command manager". This allows the administrator to specify software 'packages' that can be installed on workstations in one of two ways. She can schedule compulsory software packages that will be installed at a certain time. For optional software, the user can be presented with a list of software that can be installed at login. The second option helps with the problem of buying licenses that you don't need,

allocating expensive software packages only to those who need them.

In order to use the package command manager, you need to bundle your software into a package. Creating a package involves finding or creating a single-command install option that is usually silent to the user, defaults to all the right options and preferably runs in the background. For some applications, this is trivial. For others, it can be quite painful, but we've yet to find a package we absolutely can't do this with. As a last ditch resort, we wrap the setup program in a Visual Basic program that basically clicks the correct buttons automatically. (This borders on ridiculous, but it works.)

It's also possible to do remote installs without SMS. The administrator wraps the software installation in the same kind of package, and then uses rsh or rlogin to connect to the machine and execute the setup command locally. The Microsoft supplied rsh is less than ideal, so we use the Ataman package. The advantage of using rsh is that it runs when you tell it to, in contrast to Package Command Manager, which has its own internal scheduling mechanisms. If you need to push a hotfix out right away, you will want to use rsh.

An administrator can also use the NT scheduler service to execute commands on remote machines. This is again similar to the package command manager method of installation. We only briefly toyed with this idea, as it involves configuring the machine to run the scheduler service as a user with administrative privileges, and we were unsure of the security aspects involved. The other options seemed to provide the same functionality, but this method is an option to consider.

In essence, remote installation consists of three issues:

- Wrapping the software in some kind of package that essentially turns the entire install into one single command.
- Initiating that command on a remote machine. SMS handles this the most gracefully.
- Worrying about the permissions with which the remote command will execute.

4.4. Installation Via Replication

Another approach is to replicate a disk image onto the local disk. (We've heard of some sites that boot into linux in order to update Windows95 FAT partitions remotely.) The replication can come from a CD or a networked disk. The problem with this

method is that it essentially involves constantly erasing the local disk, which has two side effects.

First, this prevents any user data being stored locally. This may not be so bad if your users are careful about keeping their data on a network server, and if you store profiles on the servers. But we've had interesting problems with that solution, particularly in an environment that has a lot of laptops. In terms of a time investment, this can be almost as bad, or perhaps even worse, than manual software installations.

Second, there's an inherent SID problem replication brings up, in that that the machine's SID needs to be retained, or in the case of building new machines, a new SID has to be created. Finally, not all machines in an environment share an identical hardware base.

The replication solution just doesn't scale to a large environment, at least not with today's available replication software.

4.5. Remote Booting

Finally, one can consider remote booting by using a boot PROM on the network card on a machine with no local disk. From an administrative standpoint, this is a dream setup. From a user standpoint, this is the worst possible solution, as you've sacrificed local disk performance for ease of installation. If this is the option you go with, you should probably consider some of the Windows terminals that are available, or wait for the NT 5 caching schemes.

5. Requirements for a Software Installation Solution

When we first started installing NT machines in our workplace, we took the simple solution to installing software: we installed it by hand. That worked fine as long as there were only a few machines, and those of us using them knew exactly what we were doing. But the situation quickly got out of hand, and we realized we were spending a lot of time walking from one machine to the next, a lousy solution that we've never had to implement on our UNIX machines.

At that point, we went through our analysis described above - understanding how NT software is installed, considering various options for large-scale installation, and identifying what we liked about our solution for UNIX.

We decided that whatever solution we developed had to have the following requirements.

- The installed applications need to achieve reasonable performance. Our first attempt was to use a network installation for major software. Performance was miserable. Not only did users complain, but in some cases, the software crashed. This was in part due to slow network storage, and in part because the applications didn't handle non-local disk installation very well. So, this meant that we had to plan to install most software onto local disk.
- The distribution to each machine had to be automated. We refuse to do more walking from machine to machine. We'd like to visit each computer only twice in its lifetime: once to install it, and once to take it away. We should be able to do everything else remotely. Note that this isn't because we're lazy, it's because visiting each computer simply doesn't scale.
- Our solution should attempt to keep the application from interfering with the operating system. This would help reduce mysterious problems with the operating system and collisions with other applications.
- We would like every machine that we manage to be as identical as possible, differing only when the hardware configuring forces it. The operating system, the patch levels, and the applications should otherwise be the same. This allows us to generalize across the machines, to simply know, without looking, that an application is installed or that some command should work. Unfortunately, this doesn't work in the NT world for two reasons. First, no machine has that much local disk space. Second, the licensing cost would overwhelm us.
- More realistically, we decided to define several classes of machines: developer machines, visualization machines, student machines, secretary machines, and so on. Computers in a particular group all have the same software installed on them. This is not quite as good as having one flavor of machine, but it still lets us generalize.
- We've learned not to trust applications that have been known to mess with the OS. We'd like to have a way of discovering what they did at installation, and what they did over time. Ideally we would have a way of checking each machine regularly. This would help us maintain our goal of keeping every machine the same.
- Our solution should be as simple as possible. (It's currently not.) For example, it should be

possible to build a machine and all the software on it by popping a floppy in it, booting, and walking away. Hours later, it should have all software installed on it correctly. Afterwards, when we update software across the net, it should only take one command, and it should automatically happen on all of the appropriate machines.

- We have a lot of students who work with us for short periods of time. They should be able to come up to speed on the tools that we're using relatively quickly. This means that an encyclopedic knowledge of NT and NT tools should not be required.
- Our method needed to help us keep track of installed software so that we could ensure that we are legal with respect to licenses for installed software.
- Finally, we're aware that the NT world changes pretty quickly. We didn't want to spend a lot of time building a custom solution that would be useless in a year. Instead, we wanted to build it out of existing tools, using Microsoft-sanctioned methods, so that we would be more likely to be able to adapt to changes.

6. Our NT Software Installation Approach

There are three main components to our current NT software installation method.

1. A repository of packaged software.
2. A build procedure that installs a pre-determined set of packaged software on new machines.
3. An update process that installs packaged software on existing machines.

When we have a new application to install, we package it, put it in the repository, and arrange for new machines to install it. Then we initiate a network push that installs that software on all existing machines. This solution doesn't quite meet all of our ideals (for example, packaging software is not particularly simple), but it's a rational method that has reduced the amount of legwork and helped to keep our machines running consistent sets of software.

6.1. The Software Repository

The software repository is a networked share where we store software that is ready to be installed. Each of these applications is "packaged", meaning that it

can be completely installed by running a single command.

The majority of the work here comes in creating the package. In many cases, applications already have a silent or unattended install option, and the package simply consists of running 'setup' and directing it to the correct "answer" file for its defaults.

In other cases we have to use additional software to prepare the package for unattended install. Internet Explorer and Microsoft Office, both required the use of their resource kits to create an unattended distribution.

And then there are the applications that have no concept of unattended install. For these, we write simple VB apps which send the correct key sequences to the installer applets.

6.2. The Build Procedure

We use the NT 4.0 unattended installation procedure and our own boot floppy to initiate the install process. We found that we can significantly reduce the initial build time by following the Microsoft recommendation to remove portions of the NT distribution from the distribution location on the network. The initial build time, which includes formatting a portion of the workstation disk, takes about 25 minutes per machine on a fast ethernet network. The initial distribution applies the latest service packs and hot-fixes, installing them by using the "run at install time" feature in unattended install mode.

Once the OS is installed, our build procedure adds a registry key for auto-logon of a predetermined account at boot. The machine reboots, and logs in as that account. The logon script for that account registers the machine with the SMS database. Next, the script initiates the step that installs all of the packaged software. First it looks to see what class of machines this computer is (for example, secretary or developer), from which it can determine which applications need to be installed. This length of time that this takes depends on the number of applications that we add, but 10 to 15 minutes is about average.

Finally, the login script makes some local registry modifications, copies in some shortcuts, and adds printers.

Setting the network correctly is tricky, especially since the machine is usually moved from our lab to someone's desktop after it has been built, which may require that it get a new network address. When the machine is initially built, we use DHCP to let assign an IP address from the server's pool of available

addresses. This is how DHCP servers are normally used. However, we don't use Dynamic DNS, and we want DNS to work for every machine, so we have to make sure that once a machine is built, it gets the same IP address from that point on. We accomplish this by configuring the DHCP server to reserve a specific IP address for a client MAC address. We obtain the MAC address from SMS during the part of the build process in which the new machine adds itself to the SMS database.

6.3. The Update Process

Once a piece of software is packaged, installing it on a remote machine is simply a matter of remotely invoking the package with the right permissions. A number of ways to do that are discussed in Section 4.3.

For the most part, we use SMS to push software packages out, because it runs with the right permissions and it gracefully handles machines that are temporarily off. Occasionally, if we're in a hurry, we use rsh rather than SMS. At the moment, these pushes are initiated by hand, but they could be automated to ensure that every machine of a certain class always has the correct software.

6.4. Observations on Our Approach

The approach we've taken is working very well for us. We integrated the best parts of what we researched, and came up with a reasonably automated system that requires little administrator presence at the machine after the initial build.

Being able to specify that all machines get the latest hot-fix without having to leave our office is a big win from an administrative standpoint. In fact, that very situation came up at our site after the recent denial of service attacks aimed at .gov sites.

We are able to keep a steady base level of machines by ensuring that when a software package is added to the default build for new machines, it also gets installed on the existing base of machines. This keeps guesswork at a minimum when it comes to troubleshooting or license tracking. It's about as close as we can come to a unix environment, where all applications sit in a common nfs-shared repository. We at least can be assured of a common platform.

Using these methods, the only differences between machines within the same group (general workstation, development, etc) are at a hardware and driver level. It's still not the perfect world, but it's at least as level a playing field as unix.

We have a few problems left. We don't have a way of carefully monitoring all machines to see what has changed during an installation, and we occasionally run into problems with this. While we think our current solution is pretty simple, packaging a piece of software can sometimes be fairly tricky. Our biggest problems, though, are associated with the basic NT installation issues that we can't fix: the tendency of an application to be too closely tied to a specific machine and user.

7. Comparing NT and UNIX Installation Methodology

Having implemented fairly comprehensive software installation methods for both NT and UNIX, we find that neither of our solutions is completely satisfactory. Indeed, there are features of our UNIX machines that we wish we could use on NT, and aspects of the NT installs that would be very handy on UNIX.

7.1. Our UNIX Wishlist

Here are the features of our NT application installation that we wish we could easily duplicate on UNIX:

- Better fault tolerance. When the UNIX file server with all the applications crashes, which is rare, everyone is in trouble. On NT machines, nearly all of the applications are local, so people can continue to work even when the network crashes. One way to fix this with UNIX would be to have smarter automounters that can handle server failover. Another would be to install applications locally on each machine, just as we do now with NT.
- A common push/pull method that's better than rdist.
- Even fewer machine-specific applications. We're worried about the trend we see in some UNIX software that forces us to install applications locally and then node-locks the licenses.
- A one-step configure, build, and install mechanism similar to NT's InstallShield.

7.2. Our NT Wishlist

- Symbolic links.
- A clean, enforced separation of the operating system, applications, and user space.
- A clean separation of multiple versions of the same application. It's very handy to be able to

try out a beta version of an application without having to move completely away from the stable version.

- More useful profiles. Profiles go a long way towards capturing a user's configuration, but not enough applications use them, and roaming profiles just don't work right. Storing profiles on the server, with options to copy, link or create machine-specific profiles, similar to the irix method, would be a better solution.
- Less reliance on the local registry. Too many applications use it for storing configuration and state information. This doesn't allow for a consistent work environment for users that use more than one machine. We'd like to see software installs stay away from the HKLM registry tree altogether, and instead of editing HKCU, putting configuration settings into a globally available repository.
- Consistent silent installation mechanisms. There are too many different installer technologies for Windows, each with its own learning curve and methodology, and many of these installers don't support a silent install option, forcing ugly hacks.

8. Conclusions

We set out to develop a scalable software installation procedure for NT that was as useful to us as our existing UNIX installation strategy. After learning about NT application issues, many tools for NT installation, and trying out various options, we have developed a solution that, while unlike UNIX strategy, is still fairly scalable. During the process, we identified some features of NT that we would like to replicate on UNIX, and vice-versa.

Our remaining difficulties with large-scale support of NT application installation are intrinsic to the way in which NT applications interact with the operating system. We know that some of these issues will be addressed in future versions of NT, and look forward to that event.

9. Author and Project Information

Michail Gomberg is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. He was the lead technical architect for this project. His email address is gomberg@mcs.anl.gov.

Rémy Evard is the manager of Advanced Computing Technologies and Networks in the Mathematics and

Computer Science Division at Argonne National Laboratory. He is actively pursuing research in systems administration, with the hope of making it less difficult and more fun. His email address is evard@mcs.anl.gov.

Craig Stacey is a systems administrator in the Mathematics and Computer Science Division at Argonne National Laboratory. He was the one whose soles were saved with this procedure. His email address is stace@mcs.anl.gov.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

10. References

- [1] Colyer, Wallace & Wong, Walter, *Depot: A Tool For Managing Software Environments*, LISA VI Conference Proceedings, 1992.
- [2] Evard, Rémy and Leslie, Robert, *Soft: A Software Environment Abstraction Mechanism*, LISA VIII Conference Proceedings, 1994.
- [3] Microsoft Windows NT Workstation 4.0 Resource Kit, Microsoft Corporation, Microsoft Press, 1996.

Appendix A: Tools

The following is a list of many of the tools that we've found to be very useful.

NT Resource Kit:

Srvinfo	A very useful utility to look at services on a remote machine from the command line.
Shortcut	Allows us to copy .lnk files to the right places without remaining linked to the source location.
Windiff	A graphical comparison tool that can be used to look directory differences.
Instsrv	Allows installation of services from the command line.
rkillsv	Allows processes to be killed remotely.
rconsole	Remote console for NT.
setupmgr	GUI used to create a base answer file for the unattended setup. We had to do a lot of tuning to this file to get it to be really useful.

NT Rollout tools:

Sysdiff	Creates snapshots and difference databases of a machine after software is installed.
IEAK	Internet Explorer administrator kit. It creates a custom distribution, which can then be installed in unattended mode.
ORK	Office resource kit. Same function as the IEAK, but for MS Office.
Winnt	This is the NT installer that comes on the NT 4.0 CD. We use it to install NT in unattended mode.
regedit	Merges registry edits into the machine registry.

NT Services:

Dfs	Lets us create a single distribution location, spread over multiple disks and multiple servers.
SMS	SMS provides a database of clients on the network, along with the ability to schedule commands for

execution across the clients. Also includes software auditing and remote control.

SMS Package Command Manager

This service comes with BackOffice resource kit and allows SMS jobs to run unattended.

DHCP Server	We use DHCP to get the machine onto the network initially, without having to assign an IP address at build time. Once the machine is built, we use DHCP to assign a specific IP address to it, by getting the MAC address from SMS.
-------------	---

Scheduler Service

Can be used to run jobs remotely, however there are security issues regarding 'system' account.

Other Microsoft Tools:

DOS 6.22	Boot disks need to be DOS 6.22 in order to fit all required files on it. Windows95 or 98 system files take up too much room.
MSLANMAN	We use LAN Manager for DOS to copy the NT files onto the workstation disk.
MSDN CDs	Incredibly useful for getting the latest information, especially pertaining to the reduced TCO initiatives from Microsoft.

Windows Development Tools:

Visual Basic	We use VB to run queries on the SMS database, and create automated installations for applications that don't have an unattended install option.
--------------	---

Third party:

Ataman rsh	Execute commands remotely on machines.
Perl5	We use the Activeware version. Without this we would be sunk, or at least very unhappy.