Automated Upgrades in a Lab Environment

Paul Riddle – University of Maryland, Baltimore County

ABSTRACT

Back in the late 80s and early 90s, when disk drives were expensive, it was more economical to buy one server and configure it with enough disk space to support several "diskless" workstations. Now that disks are cheaper, most workstations now come with internal disks which contain an entire bootable operating system.

Most vendors provide ways of automatically upgrading multiple "diskless" workstations; unfortunately, the same is not true for "diskfull" configurations. Upgrading "diskfull" workstations typically involves either a lot of manpower or a lot of tedious, repetitive work. In any moderate to large sized network, something needs to be done to automate the upgrade process.

This paper describes a scheme which we use to upgrade our various networks of Silicon Graphics workstations. Interestingly, it relies on the same technology that allows "diskless" workstations to boot over the network.

Introduction

Our upgrade scheme works using diskless booting. Each workstation boots over the network from another workstation, which we designate as an "upgrade server." Once booted, the workstation runs an upgrade script (written in *Perl* [1]) which partitions its system disk, creates filesystems, installs an operating system distribution, and then installs customized system files. When finished, the workstation reboots from its system disk. This scheme allows for unattended system upgrades and has proven to be quite flexible; we have used it to upgrade two separate networks of SGI Indigos from Irix 4.0.5 to Irix 5.2.

What We Were Looking For In An Upgrade Scheme

Automation

The upgrade procedure should not require that we physically visit every workstation. This is a problem in our environment where many workstations are located in private offices to which we don't have easy access. Visiting each machine also requires a lot of manpower and can be error-prone; operator errors can lead to machines being upgraded incompletely, improperly, or not at all.

Flexibility

An upgrade scheme should be able to deal gracefully with different sized system disks, different models of a vendor's workstations, etc. It should be able to repartition and create filesystems on the machine's local disk, if necessary.

Reliability

The upgrade procedure should be reliable. It should never leave a machine in a partially-upgraded state. If an upgrade is interrupted or otherwise fails, it should pick up where it left off, or start over the next time the machine is rebooted. It should have some way of notifying the system administrator when an upgrade fails or completes successfully.

Speed and Convenience

Upgrading should be reasonably fast and should not require a lot of downtime. Alternatively, it should be automated to the point where it can be done overnight, when there is less demand for workstations and network bandwidth.

Our Environment

The University of Maryland, Baltimore County is one of the largest educational installations of Silicon Graphics (SGI) equipment in the country. There are approximately 200 SGI workstations on campus, spread out over about 8 different administrative domains and 10 subnets. The abundance of SGIs required us to come up with some way of keeping them up-to-date with the latest release of Irix (SGI's flavor of UNIX). We chose two different workstation networks to use as "Guinea Pigs" for testing our upgrade scheme.

For one upgrade environment, we used three student labs consisting of a total of about 90 SGI Indigos, some with entry level (RPC) graphics, and others with extended (XS24 or Elan) graphics. Each workstation has a 420-megabyte internal system disk. The workstations are spread over two different subnets.

A second upgrade environment consisted of about 30 SGI Indigos, mainly with low-end graphics, and 10 SGI Indy systems. Some of these machines have 420-megabyte system disks and others have 1-gigabyte system disks. All are on the same subnet.

For both environments, the task was to upgrade from some revision of Irix 4.0.5 (4.0.5F in some cases and 4.0.5H in others) to Irix 5.2.

Alternatives To Our Approach

We evaluated several other methods of upgrading before choosing to implement one based on diskless booting. Each of these has its advantages, but fails to meet our requirements in one or more ways.

Upgrading Systems Individually

The most obvious and straightforward upgrade strategy is simply to upgrade systems manually, one at a time. We discarded this idea quickly because it was too time consuming. It also requires physically visiting each workstation. Additionally, manually upgrading a workstation is a tedious process which involves many steps. When many workstations are upgraded in this way, it can lead to subtle differences and inconsistencies between systems.

Manual Disk ''Cloning''

A faster method is to upgrade manually one of each different type of system, and then upgrade the rest of the machines using a sector-by-sector disk copy. This is much faster and more reliable than upgrading individually, but still requires physically visiting each and every machine. The disk cloning doesn't extend to systems with differing system disk geometries, either. For example, you can't clone a 1-gigabyte system disk onto a 420-megabyte system disk; it just doesn't work. Operator error also creeps into the picture; although you're less likely to end up with inconsistencies between systems, there is still a good chance that machines can be missed or otherwise improperly upgraded.

Although this method doesn't really meet our needs, we did use it for awhile because it is simple and straightforward. Trained student employees provided the manpower.

Upgrading Running Systems With *rdist*[2]

Still another approach was to use *rdist* or a similar tool to upgrade a running system[3]. This worked well for a minor OS revision, but was not capable of handling a major revision such as upgrading from Irix 4.0.5H to Irix 5.2.

Using Unused Swap Space For Upgrade Filesystem

Another method was to use unused swap space to create an upgrade filesystem. SGIs allow swap to be removed from a running system, so it was possible to dynamically delete enough swap to create room for the upgrade filesystem, boot from there, and upgrade the system disk over the network. However, this approach is not 100% reliable, since there's a chance that adequate swap space may not be available at upgrade time. Also, this approach doesn't allow for repartitioning the system disk during the upgrade, since part of the disk is in use as the upgrade filesystem.

Our Solution

In designing an upgrade scheme, we worked to come up with a solution that satisfied all of our criteria: automation, flexibility, reliability, and speed. An important requirement was to avoid having to visit each workstation individually. This ruled out any solution involving disk "cloning" or upgrading individually from CD-ROM. We worked around this by having workstations copy the operating system over the network from a server.

In order to do this, the workstation needs to be booted to a state where its network interface is operational and its system disk is not being used. Enter diskless booting.

Diskless booting is an attractive solution because it allows for complete control of the system disk when performing the upgrade. The disk can be reformatted, repartitioned, mounted, unmounted, etc. at will. However, diskless booting is not without its problems. The booting protocol requires that the upgrade server be located on the same logical network as the client being upgraded. Many simultaneous upgrades can place an undesirable load on the network. The next section describes how we worked around the former problem. For the latter problem, we place limits on the number of simultaneous upgrades at the expense of time.

The Upgrade Procedure

Doing upgrades is a three-step process. First, you need to configure each upgrade server to support diskless clients. Then, you must do a prototype installation for each different type of environment you are supporting. Finally, each workstation needs to be configured to boot from the upgrade server and then rebooted to start the upgrade process.

Configuring Servers For Diskless Booting

The first step in configuring the upgrade server is to build the diskless booting area. Let's assume that the hostname for the upgrade server is *sonata.* The upgrade area is rooted on sonata under */upgrade.*

The upgrade area contains everything that a workstation needs to boot diskless over the network and perform its upgrade procedure. A minimal number of OS files are necessary to support a diskless environment. All prototype and site-dependent distribution trees also live under the upgrade area.

Prototype distributions are located under */upgrade/proto*. Under recent releases of Irix, machines with different graphics boards and/or processors require slightly different installations of the operating system. Each installation requires a separate prototype tree. For example, if your site has R4000 Indigos with entry (RPC) graphics and R3000 Indigos with Elan graphics, you would need two prototype distributions, which might be called */upgrade/proto/4krpc* and */upgrade/proto/3kelan*. (The names are arbitrary; you can choose whatever names you want.) Under these trees would be two prototype Irix installations, one for both machine architectures. Prototype distributions are either disk images or filesystem images generated by *dump*; the next section describes how to generate them.

We were able to work around the need for multiple prototype distributions by making several modifications to the default Irix distribution provided by SGI. This was done at the cost of a few extra megabytes of disk space on each system, which we decided was an acceptable tradeoff. The specific details of our modifications are beyond the scope of this paper, but we will make them available via FTP along with the rest of our upgrade tools.

Site distribution trees are located in */upgrade/dist*. Once the client has copied the appropriate prototype distribution, it uses *rdist* to copy selected site distribution trees. These trees contain system files which need to be modified from the defaults supplied by SGI, and any additional site-dependent files which need to live on the workstation's local disk.

The main purpose of site distribution trees is to separate customized files from standard files. This reduces the possibility that customized files will be lost when doing an upgrade.

The */upgrade* tree must exported to all clients. Each client mounts */upgrade* as its root filesystem. To allow for multiple simultaneous upgrades, we export */upgrade* readonly and take pains to ensure that the clients do not try to write to it.

Building Prototype Environments

To build prototype installations, we manually upgraded one of each type of workstation and then copied the resulting installation onto an external hard disk. Putting the distributions on an external disk allowed us to move them around from machine to machine, thereby enabling us to set up installation servers on different subnets. We found that a 1.6 gigabyte external drive was large enough to hold two separate prototype installations.

For networks with identically-sized system disks, we used *dd* [4] to copy the disk image over to the external drive. This is the fastest way to do things. Unfortunately, it doesn't work on networks where workstations have system disks with differing geometries. In this case, we used *dump*, [5] which is slower, but works on any disk regardless of geometry and partitioning. *Dump* also requires a separate prototype file for each filesystem on the client's disk. For example, rather than a single disk image, we might have two separate *dump* images called */upgrade/proto/3kelan.root* and */upgrade/proto/3kelan.usr*.

Once we built all of the prototypes, we attached the external drive to the upgrade server and mounted it under */upgrade/proto*.

The Upgrade Procedure

Workstations upgrade themselves using the following procedure. First, each client must be configured to boot diskless from its upgrade server. On Silicon Graphics boxes, this is done by setting two variables in non-volatile RAM (nvram) on each client:

```
client# nvram diskless 1
client# nvram bootfile \
     bootp()sonata:/usr/etc/boot/upgrade/unix
client# /etc/reboot
```

We did this for each of our clients using a simple shell script. Other methods include *rdist, cron,* etc.

When the workstation reboots, it loads the kernel image specified in nvram and mounts */upgrade* via NFS[6] from the upgrade server, *sonata,* as its root filesystem. It then starts up the init process, which in turn runs */etc/rc*.

/etc/rc begins by reading the "netaddr" variable from nvram, which contains the client's IP address. It looks up this value in the */etc/hosts* file to determine the system's hostname, and then configures the network interface.

Next, */etc/rc* execs the upgrade program, which is a Perl script. The script has four basic functions:

1. Repartition the local disk (optional). If desired, the existing partitions can be used.
2. Create new filesystems on the local disk.
3. Copy a prototype Irix distribution from the upgrade area to the local disk. As mentioned earlier, the prototype distributions are virgin Irix distributions installed directly from CDROM, with no modifications. This is done using *dd* or *dump*, invoked using a remote shell from the client to the upgrade server. A sample *dd* command would look something like

   ```
   rsh sonata dd ibs=32768 obs=1450 \
      if=/upgrade/proto/3kelan | \
      dd ibs=1450 obs=32768 \
      of=/dev/rdsk/dks0d1vol
   ```

   The blocking factor is determined by taking the MTU of SGI's ethernet interface and subtracting 50 bytes for TCP overhead. The output from the disk image is sent directly to the client's raw disk device.
4. Copy any number of site-specific distribution trees on top of the new OS distribution. This is where all customized system files are

installed. The copy is done by invoking *rdist* on the server via remote shell. For example:

```
rsh sonata rdist -c \
  /upgrade/dist/3kelan client-hostname:/
```

Once this has finished, the upgrade script resets the nvram variables and reboots the system with the newly installed operating system. If any part of the upgrade is interrupted (due to someone turning off or resetting the machine, power failure, etc.), the upgrade procedure will start over when the system reboots.

Performance Observations

We found that it took approximately 20 minutes to copy a 420-megabyte disk image over a lightly loaded ethernet using *dd.* Using *dump*, The procedure took about 30 minutes. By comparison, a direct sector-by-sector disk copy took around 10 minutes. Unfortunately, ethernet doesn't have quite the bandwidth required to upgrade more than one workstation at a time. We found that the most efficient way to get the upgrade done was to write a script that upgrades each workstation sequentially, and let it run overnight.

Future Work

Currently, our scheme requires that the root, swap and usr partitions be allocated to specific partitions on the local disk. This works fine for just about any workstation. However, we would like to expand this to be a bit more flexible and support customized configurations.

Our scheme is also very dependent on NFS. We'd like to eliminate NFS from the picture (except where it is required for diskless booting) and switch to a different method of copying the prototype areas. FTP[7] appears to be a very attractive solution.

Maintaining separate prototype distributions can eat up a lot of disk space. We have a couple of ideas which would alleviate this problem. One is to use actual running systems as prototypes; however, this would require a different upgrade server for each individual machine type, which may be difficult to do in terms of network topology. Another solution, one which we may implement in the future, would be to have each client do a direct install from a CDROM server. Unfortunately, this tends to be a slow process, and also requires a front end (such as *expect*[8]) to drive the installation process. Expect scripts would need to be tailored for each different Irix release, which would be tedious and problematic.

The current method we use to initiate an upgrade is somewhat of a kludge. It would be nice to have a server/client type protocol which allows the admin to start upgrades remotely and monitor their progress.

Author Information

Paul Riddle is a Systems Programmer with Academic Computing Services at the University of Maryland, Baltimore County (UMBC). He has been working at UMBC since 1989. When he graduated in 1992, he made the transition from underpaid student to full-time employee. Currently, Paul works with *sendmail* and DNS, and helps to keep the student labs running, among other things. Someday he hopes to become motivated enough to get a Master's degree, too. Reach him via U.S. Mail at The University of Maryland, Baltimore County; 5401 Wilkens Avenue; Baltimore, MD 21228. Reach him electronically at paulr@umbc.edu.

Availability

We expect to have the final version of our upgrade software ready by September 1, 1994. It will be available via anonymous FTP from *ftp.umbc.edu* in the directory */pub/sgi/upgrade*.

References

[1] Wall, L., & Schwartz, R., *Programming Perl*, O'Reilly & Associates, Inc., 1990.

[2] "rdist(1C) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[3] Manning, C., and Irvin, T., "Upgrading 150 Workstations in a Single Setting", *Proc. 7th Usenix Systems Administration Conference (LISA VII)*, 1993.

[4] "dd(1M) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[5] "dump(1M) Manual Page," *IRIX Reference Manual*, Silicon Graphics, 1993.

[6] "NFS Protocol Specification," *Networking on the Sun Workstation*, Sun Microsystems, 1986.

[7] Postel, J. & Reynolds, J., "File Transfer Protocol (FTP)," *RFC 959*, Network Information Center, 1985.

[8] Libes, D., "Using *expect* to Automate System Administration Tasks", *Proc. 4th Usenix Systems Administration Conference (LISA IV)*, 1990.