Software Deployment on Network Storage Based Systems

Todd Poynor
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto

HPL-2001-257
October 12th , 2001*

software deployment, software installation, diskless, network storage

This report briefly summarizes a number of recent investigations related to software deployment on network storage based systems that our team recently performed. A specific context in which this was investigated was for a pool of general-purpose servers in a single data center that are to be allocated among different applications and/or different customers according to demand or level of service purchased. Separating software installations from server machines increases the modularity of the data center, such that the appropriate level of compute power can be more easily provisioned for a specific software environment and the available compute power can be more efficiently allocated among software environments. A number of additional capabilities and concerns unique to "diskless" software deployment are discussed, as are general topics in improving the modularity and flexibility by which software packages are installed, configured, combined into software stacks, and shared among servers in cluster environments.

1. Introduction

This report briefly summarizes a number of recent investigations related to software deployment on network storage based systems that our team performed as part of the Distributed Service Utility research project at HP Labs. This information is being published to the research and development community in order to document some of the ideas, results, and ongoing projects that sprang from that effort.

Our use of the term *network storage based systems* refers to architectures where storage resources may be dynamically paired with compute, I/O, and other resources that make up a general-purpose server system – that is, the server has no local disk. The term encompasses both the so-called Network-Attached Storage (NAS) and Storage Area Network (SAN) architectures and associated protocols.

Here we are concerned with those portions of the software lifecycle called *software deployment*, and in particular these activities in the breakdown of that process by Carzaniga *et al* [13]:

- ***Install***: initial insertion of a system into a consumer site.
- ***Activate***: starting up the executable components of a system.
- ***Deactivate***: shutting down any executing components of an installed system.
- ***Update***: a special case of installation that can often rely on the fact that many of the needed resources have already been obtained during the installation process.
- ***Adapt***: modifying a previously installed software system, initiated by such events as a change in the environment of the consumer site.
- ***Deinstall***: removal of a system from a consumer site.

The activities from their breakdown that are not included are *release* and *derelease* (*retire*), which are activities of software producers. We also considered including configuration change tracking for troubleshooting (as a part of fault management or security management), such as to track software revisions installed, or to record when a patch was installed or a configuration parameter was modified.

2. Motivation and Design Principles

Here software deployment is investigated in the context of systems comprised of, among other things, numerous general-purpose servers in a single data center. The servers are to be allocated among different applications and/or different customers (in the case of a service provider's data center hosting multiple customers) according to demand or level of service purchased. Diskless servers for executing applications and other software are employed in service of these goals:

- **Efficient redeployment of compute power and applications** through separation between software installations and processing resources, allowing these two components to be associated in modular fashion such that the set of servers running a software installation may vary dynamically (and cost-effectively) according to need. Each change in association is to occur relatively quickly due to: the "stateless" nature of the servers; leveraging the previous work to install software from media and, to some degree, configure the software; and improvements in the process by which software is automatically configured to run on new servers and in dynamic environments.

- **Decreased hardware costs** due to lack of per-server disks and even I/O busses.

We use the term “software environment” for a collection of OS and application software, plus associated control information, housed on network storage resources that corresponds to a traditional installation of OS and application software on a local disk. The installation of a software environment occurs not to install the software on a particular server, but in order to make the environment available to run on any number of appropriate servers. We wish to quickly redeploy servers to execute different software environments according to need, as in response to changing workloads or customer or resource sets. By separating software environments from servers, we can accomplish this in a more dynamic (and cost-effective) fashion than when software environments and servers are tightly bound together. The appropriate level of compute power can be more easily provisioned for software environments, and the available compute power can be more efficiently allocated among competing environments.

Such an architecture has the above advantages, and others noted below, over other projects, such as IBM Océano [14], that are also aimed at dynamically allocating data center resources but that employ traditional directly-attached disks. Other research projects that make use of network storage for efficiency of server reallocation include Muse at Duke University [17].

The same techniques may be used not just for general-purpose servers based on such operating systems as UNIX and Windows but also for reprogrammable networking infrastructure functions running on multi-purpose platforms.

3. Sharing and Replicating Software Environments

There are well-known benefits to sharing software environments and/or individual software products among multiple servers, and even across multiple customers hosted at the same data center. Sharable software packages also lend themselves to efficient replication on multiple machines. This can be greatly beneficial in manually constructing a cluster of identical servers and in automated deployment of additional instances of software (as to automatically meet increased demand). Sharing installations helps speed the time to construct or replicate an environment, helps reduce the amount of disk space consumed by duplicate static files, and offers advantages in centralized management of environments over the deployment life cycle, such as to reduce the need for duplicate actions applied to multiple environments. Although these concepts also apply to traditional systems with local disks, for network storage based systems these capabilities are vital to deriving the full benefit of the architecture.

Most tools for replicating installations are geared toward automating the steps necessary to perform the installation of a cluster of identical servers with local disks that are expected to stay in that configuration for some time. For our purposes we want to quickly and possibly automatically redeploy servers to execute differing software environments according to need in highly dynamic fashion.

For effective shared deployment of software, important considerations include whether software subscribes to a file system layout policy that enables cross-system sharing of software and whether the installation procedures are organized into one-time *install* and per-replication *configure* phases, as described in the subsections that follow.

3.1 File System Layout Policy

The policies by which OS and application files are placed in the file system hierarchy play a large part in the way software is shared. This is influenced by such factors as:

1. Is there a distinction between the directories to which sharable, read-only files are installed and the directories in which administrator-modified configuration files and programmatically-written log files and other per-instance files are installed or created?

2. Is there a clear separation between the directories to which separate software products install sharable files, such that there is a clear mapping between an OS or application product and the directories needed to obtain access to the product (and only that product)? Preferably, each product installs all sharable files beneath a single product root directory.

For example, the HP-UX file system hierarchy is organized (à la System V.4) into sharable vs. host-private mount points [3], with structure for separating optional applications from the base OS, to ease and speed replication of software in NFS-mounted clusters. Linux software has traditionally been less consistent in adherence to such a file system layout policy. In fact, many separate packages install into common directories **/usr/local/bin**, **/usr/local/lib**, **/usr/local/man**, etc., in direct conflict with point #2 above. There is a move toward greater organization of the layout in such Linux distributions as Red Hat, which adopts the Filesystem Hierarchy Standard (FHS) [12].

3.2 Per-Server Configuration

Many of the options for efficiently sharing and replicating software configurations boil down to separating the installation task into a one-time *install* action and a per-target *configuration* action. In this fashion, read-only bits may be shared among multiple servers, as via NFS mount points, and host-private information may be configured per server using such means as package-specific per-server configuration performed by scripts. For example, the HP-UX installation utility SD-UX subscribes to this model [4, 5].

On systems that either do not subscribe to amenable file system layouts and/or do not implement a per-server configure phase in the installation process, it is possible compensate for this to some degree using automation. We can "wrap" the existing tools to help automatically factor out installation and configuration steps and installation directories, or we can modify the tools and software packages to add explicit *install* and *configure* phases. Methods of automatically determining *install* vs. *configure* actions include determining "deltas" of file information as in the NT **SysDiff** [10] and HP-UX **mkpkg** [11] utilities, and instrumenting the file system protocols (designs for which are still under investigation and may be described in a future report). Potential future directions include enhancing existing installation tools and system software to enable these features.

Of key interest has been investigation into methods of improving the flexibility of software configuration, in part so that even software with highly complex interactions with other software and other network resources may be efficiently replicated to new machines. Although not necessarily specific to network-storage-based systems, we briefly note some of these here. Improved languages for expressing dynamic portions of configuration files using markup languages such as XML have been one area of investigation. Another area that has been pursued is a framework for software to dynamically configure and respond to changes in the surrounding environment, such as to establish and break ties with other software and hardware services, as needed over the lifetime of execution of the program [15]. Some additional investigation that was performed looked at management information models by which software and servers may be described in machine-readable formats for automation of deployment (such as expressing dependencies on other software and hardware platforms), expressing service level requirements, and so forth. This effort looked for improvements over existing mechanisms such as the DMTF CIM Application Management Model [18], which has limited support for distributed systems

[20] due in part to its origins in traditional network operations management [21], and the Marimba/Microsoft Open Software Description (OSD) [19], which has a number of limitations noted in [22]. The general topic of dynamically reconfigurable and replicable software remains of high interest as part of the utility computing programs at HP Labs [16], and work continues in these areas.

3.3 Avoiding Maintenance of Local Disks

A number of products exist for replicating an installation onto multiple servers with local disks. For HP-UX, Ignite-UX is used for this purpose; on Linux the most popular tools include VA SystemImager [1], REMBO [2] (which also replicates NT and other Windows OS'es), Red Hat KickStart [6, 7] (based on RPM [9] packages), and IBM Linux Utility for cluster Installs (LUI) [8].

Part of the operation of these tools is to load remote software packages or file system images onto a local disk at boot time. For Network storage based systems, downloading of files to local disks is replaced by NFS mounting of remote file systems at a considerable increase in initial speed of redeployment.

Reallocating a physical disk device from one customer to another exposes the danger that data belonging to the previously allocating customer remains on the disk for potential access by the newly allocating customer. For this reason, it is normally necessary to reformat the disk, wiping out all data stored previously, when reallocating a disk to a new customer, a rather time-intensive operation (the IBM Océano project [14], for example, does this). Network storage based systems can avoid the need for reformatting physical devices by controlling access to most storage resources, providing only file-system-level interfaces to files created by the customer and not providing general disk-block-level access to these resources.

3.4 Management of Software Environments

A number of management tasks are needed for the individual software environments, such as to browse, delete, etc. the environments, as well as the "catalog" of available environments. If environments consist mainly of file system images residing within the file system, existing file system management tools may provide a minimum of functionality, but may not be sufficient to implement an easily managed system, especially for a very large number of environments. There are also a number of additional actions that may be useful for shared software environments, such as to independently upgrade a particular software package or select a subset of servers that are to be upgraded, and to modify the set of servers that are to execute a certain environment. Potential future activities were to address this topic.

3.5 Security Implications

Data centers shared among multiple customers have certain special security considerations, including protection against unauthorized access to the software deployed. Common file system access controls probably suffice for protection of file system images against snooping or corruption by other than the owning customer. It may, however, be advantageous to allow instantiation of the same "base" file system image for multiple customers, and even the details of what software a node is running may expose more information than one would like.

Different customers may not want to share read-only file system mount points due to such concerns as security, performance, and deviations from usual practices of accounting for disk space used. When a customer creates a software environment that includes a software package used by multiple customers, even sharable portions of its file system image could be copied from a protected location to a new copy for that customer only, in order to alleviate this concern. There is little new security guarantee provided, however, since shared access to an uncorrupted original image is still assumed, unless access is only granted to a trusted party higher up in the service provider chain (such as the data center owner).

4. Automated Software Environment Assemblage

The modularity principles that separate software environments from compute power can be extended to increase modularity of the software packages that comprise an environment. Improvements in, or automation of, the processes by which different software packages are assembled into a software environment, upgraded or deleted in multiple software environments at a time, etc. could provide substantial improvements in manageability. This can also play a part in satisfying goals of modularity between software environments and compute power by automating the process by which a software environment is assembled from components appropriate for a particular server’s hardware.

Rather than requiring an administrator to manually install and configure each software environment, we can provide tools and automation to help construct software environments, such as to assemble together Red Hat Linux 6.2 plus an Apache Web server plus the appropriate files that comprise the Web content for a customer (as we prototyped). This could enable such activities as:

- Combining software packages and data on the fly to match changing demand. For example, a “hot” web site could be moved from a shared Web server system to a dedicated system, or the appropriate OS for a particular server could be combined with OS-independent Web site content as required by the server architecture.
- Performing version control, such as to backtrack to a known good configuration after an installation that goes awry.
- Upgrading software simultaneously for several environments. For example, the Apache Web server software could be upgraded to a new release for each software environment that includes the Apache Web server.

One approach is to base the solution on installation tools and software packaging formats that separate the tasks of installation and configuration, using techniques as previously employed for diskless clusters (and for single-point administration clusters). A variant of this idea is to write new scripts for existing software packages that encode knowledge of how to access NFS mount points for software shares and to configure combinations of software. Another approach, which may be employed to some degree with unmodified toolsets and software packaging, records the effects of an initial install for later replay in another environment or perhaps for undoing the installation. Existing models include NT **SysDiff** [10] and HP-UX **mkpkg** [11]. An area of potential further investigation concerns how these deltas may be manipulated to achieve automatic assembly of environments.

5. Conclusion

Modular server farm architectures based on network storage have clear advantages in quick and efficient allocation of server resources dynamically matched with software environments according to policy. This report has provided some background on this design principle and has described a number of directions that have been considered, including a number of areas that we continue to be pursue, for improving the deployment and lifecycle management of software in such an environment.

Acknowledgements

Many of the architectural principles on which this report is based were originally espoused by Lance Russell, Bert Munoz, and Tung Nguyen of HP Labs. The ideas and positions on the application of these principles to software deployment documented here were formulated by a team at HP Labs led by Tom Wylegala, the membership of which also included Steve Hoyle, David A. Barrett, Dan Conway, Ilan Ginzburg, and Baila Ndiaye.

References

[1] VA SystemImager documentation at http://systemimager.sourceforge.net/.

[2] REMBO documentation at http://www.rembo.com/docs/rembo/.

[3] *HP-UX 10.0 File System Layout White Paper* at **/usr/share/doc/file_sys.ps** on an HP-UX 10.0 or above system.

[4] *NFS Diskless Concepts and Administration* white paper at **/usr/share/doc/NFSD_Concepts_Admin.ps** on an HP-UX 10.01 or above system.

[5] *File Sharing and Other Helpful Facts for HP-UX 10.0 Developers* white paper at **/usr/share/doc/dev_apps.ps** on any HP-UX 10.01 or above system.

[6] JWCS information on Red Hat KickStart at http://www.redhat.com/mirrors/LDP/HOWTO/KickStart-HOWTO.html.

[7] Red Hat Linux 6.2 Reference Guide section on KickStart at http://www.redhat.com/support/manuals/RHL-6.2-Manual/ref-guide/s1-kickstart2-howuse.html.

[8] IBM LUI information at http://oss.software.ibm.com/developerworks/opensource/linux/projects/lui/.

[9] RPM home at http://www.rpm.org/.

[10] *12 Steps to Cloning Windows NT Systems with SYSDIFF.EXE* at http://www.winntmag.com/Articles/Index.cfm?ArticleID=1.

[11] Staelin, Carl. *Mkpkg - A Software Packaging Tool*, HPL Technical Report HPL-97-125R1 at http://lib.hpl.hp.com/techpubs/97/HPL-97-125R1.html.

[12] Filesystem Hierarchy Standard at http://www.pathname.com/fhs/.

[13] Carzaniga, Antonio, *et al*. *A Characterization Framework for Software Deployment Technologies*, University of Colorado Technical Report CU-CS-857-98, 1998 at http://www.cs.colorado.edu/users/alw/doc/AvailablePubs.html.

[14] Appleby, K., Fakhouri, S., *et al*. *Océano – SLA Based Management of a Computing Utility*. Proceedings IFIP/IEEE International Symposium on Integrated Network Management 2001.

[15] Poynor, Todd. *Automating Infrastructure Composition for Internet Services*. To appear at the 15th USENIX Large Installation System Administration (LISA) Conference, December 2001. Also published internally at HP as HP Labs Tech Report HPL-2001-212.

[16] Wilkes, John, Goldsack, Patrick, *et al*. *Eos – The Dawn of the Resource Economy*. Proceedings HotOS VIII, May 2001.

[17] Chase, Jeffrey B., Anderson, Darrell C., et al. *Managing Energy and Server Resources for a Hosting Center*. Proceedings 18th Symposium on Operating System Principles (SOSP), October 2001.

[18] DMTF *Understanding the Application Management Model* at http://www.dmtf.org/spec/Whitepapers/CIM_Applications_wp.pdf.

[19] OSD -- Describing Software Packages on the Internet at http://www.marimba.com/products/whitepapers/osd-wp.html

[20] CIM Tutorial: Applications and Namespaces at http://www.dmtf.org/educ/tutorials/cim/extend/apps.html.

[21] Aschemann, Gerd and Kehr, Roger. *Towards a Requirements-based Information Model for Configuration Management*, ICCDS IV, 1998 at http://gemini.iti.informatik.tu-darmstadt.de/~kehr/publications/iccds98.pdf.

[22] Hall, Richard S., *et al*. *Software Deployment Languages and Schema*. University of Colorado Technical Report CU-SERL-203-97, 1997 at http://www.cs.colorado.edu/users/rickhall/deployment/SchemaPaper/Schema.html.