CHOS, A METHOD FOR CONCURRENTLY SUPPORTING MULTIPLE OPERATING SYSTEM

R. Shane Canon*, Cary L. Whitney, Lawrence Berkeley National Lab, Berkley, California 94523, USA

Abstract

Supporting multiple large collaborations on shared compute farms has typically resulted in divergent requirements from the users on the configuration of these farms. As the frameworks used by these collaborations are adapted to use Grids, this issue will likely have a significant impact on the effectiveness of Grids. To address these issues, a method was developed at Lawrence Berkeley National Lab and is being used in production on the PDSF cluster. This method, termed CHOS, uses a combination of a Linux kernel module, the change root system call, and several utilities to provide access to multiple Linux distributions and versions concurrently on a single system. This method will be presented, along with an explanation on how it is integrated into the login process, grid services, and batch scheduler systems. We will also describe how a distribution is installed and configured to run in this environment and explore some common problems that arise. Finally, we will relate our experience in deploying this framework on a production cluster used by several high energy and nuclear physics collaborations.

INTRODUCTION

One challenge for system administrators running a shared computer resource is the, often, divergent software needs of the users. This is particular common for a compute farm used by various High Energy Physics collaborations. PDSF, a cluster run by the National Energy Research Scientific Computing Center at Lawrence Berkeley National Lab, is just such a cluster. It provides computing for STAR, ATLAS, ALICE, KamLAND, SNO, along with experiments in Astrophysics. Recently, PDSF was faced with requests for different versions of Linux by different groups. In this paper, we present our solution to this problem. The solution is a utility, which we term "CHOS" for CHroot OS or CHoose OS.

OVERVIEW

When we have encountered request for different OSs in the past, the cluster was partitioned and the different OSs were installed on the partitions. While this satisfies the user's requirements, it has several drawbacks. The most notable is the groups are only able to run on a subset of the cluster. This can adversely impact our goal of trying to obtain high utilization on the cluster and limits the overall throughput available to a group. Another approach was needed. The solution should possess the following desired traits.

- Support multiple OSs concurrently on each node
- Not require partitioning the cluster
- Be nearly transparent to the users
- Integrate with the batch/scheduler system
- Easily deployable across the cluster
- Scale with the number of requested OS releases

The following assumptions were made.

- Only Linux distributions were expected to be supported
- All of the Linux OSs would use the same kernel.

After some experimentation, we found that performing a change root or chroot into an alternate OS tree could provide the basis for a solution. The remainder of the paper will describe how we accomplished the other objectives and some of the issues that had to be addressed.

DESIGN

While the chroot provides the beginnings of our solution, it still lacks many of the desired traits. Most obvious is even basic utilities such as ps, failed to function correctly and home file systems are not visible. This is easily resolved by mounting user file systems as well as virtual file system under the chroot tree. However, doing this for more than few OSs would become bulky, since each OS might require dozens of mounts and various auto-mounters. The desired approach should allow a common tree to be used for several OSs. This could not be accomplished with standard tools running in user-land. In order to provide a complete environment to the users, other approaches were required. Ultimately, a custom kernel module was employed to provide the needed functionality.

COMPONENTS

Here are the various components used in the framework we developed:

- A base directory that all the chroot OSs will use. We use /chos. This directory contains a series of symbolic links residing in the base directory that pass through a special symbolic link in the proc file system (/proc/chos/link). This is depicted in Figure 1. In addition, any file systems that are required by the users must be mounted underneath

*canon@nersc.gov

this directory. This would also include virtual file systems such as proc.

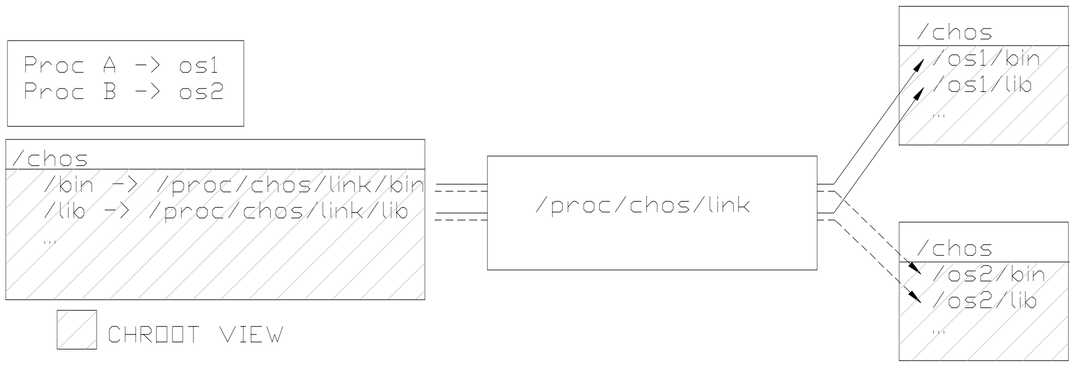

Figure 1 This illustrates the /chos directory and gives an example of how the symbolic links are resolved for two processes (A and B). In this example, process A has CHOS defined to use OS1 and process B has CHOS defined to use OS2. Notice that in the kernel and base OS view, the /chos directory exists and the OS trees are accessible below the /chos directory. In the chroot view, the /chos directory appears as the root directory.

- The custom kernel module which provides the special symbolic link. This will be described in further detail below.
- A pluggable authentication method (PAM) module to transparently select the OS for a user upon login.
- Modified grid services to implement CHOS for Grid initiated jobs. Typically this would include the various job managers.
- A custom job starter for the batch scheduler system to automatically choose the correct OS when starting a batch job.
- A Linux distribution installed in an alternate tree. This can be installed locally on a node or in an networked accessed file system, such as NFS

The Kernel Module

The core of our framework is the custom kernel module. This module provides a symbolic link that possesses the following characteristics:

- Resolves to a path that is dependent on the process requesting the lookup.
- The path is settable by a process through a user-land interface
- Child processes automatically inherit the path of the parent, unless the process explicitly changes it.

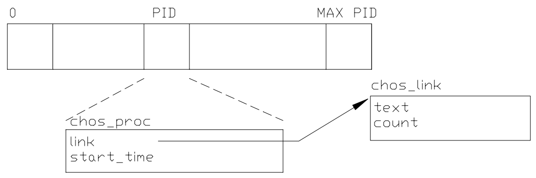

Figure 2. A diagram of the *chos_proc* array. This is an array of *chos_proc* structures. This structure contains a reference to a *chos_link* structure which contains the actual text.

To set the path, the target path is written into another file in the proc file system (/proc/chos/setchos).

An array of structures is created that has a number of elements equal to the maximum process identifier (PID), as illustrated in Figure 2. This approach was chosen for its ease of implementation and speed. If the PID space were too large to make this approach appropriate, a dynamic linked list could be used. Each process is associated with the element equal to its PID. When a process writes a new value into /proc/chos/setchos, the associated element is updated. The path that is written into the file is checked against a set of allowed paths. If the path is allowed, then a new path structure is allocated and the process entry is updated to reference this structure. In addition, the process creation time [1] is stored in the associated entry.

When a process tries to resolve the special link, the module checks the processes corresponding element in the array. If the element has a path assigned to it, then the stored creation time is checked for consistency with the actual process. This insures that a new process using a recycled PID is handled correctly. If everything checks out, the path is passed back to user land. If the element does not have a path associated with it or the checks fail, then the routine walks up the process tree (via the parent process) until an element is found that passes all the checks or the top (PID 1) is reached. In the later case, a default value is returned, otherwise the value for the ancestor is returned. Once the value is obtained, it is stored in its associated entry in the array in order to speed up subsequent calls. Once again, the stored creation time is used to verify this cached entry.

Other Components

While the kernel module is the most specialized component, the other pieces are equally important in allowing CHOS to work seamlessly for the user. The key remaining components are the PAM module, the job starter, and the alternate OSs. The role of these pieces is to make the alternate OSs appear to the users like a native installation of their required OS. In addition, modified Grid daemons are used to allow transparent access for Grid services.

The PAM module [1] checks the user's home directory for a file called *.chos*. If this file exists and it has a valid OS path for its contents, then the PAM module will automatically perform the steps to initiate a CHOS session. This allows a user to automatically obtain a specific OS upon login. The administrator simply needs to specify the PAM module in the relevant configuration file (i.e. /etc/pam.d/sshd) and the user edits their *.chos* file to specify the preferred OS.

Seamless integration was also desired for jobs initiated via the Grid. To accomplish this, a modified job manager is used for the various Globus gatekeeper gateways. Similar to the PAM module, the modifications allow the job manager to examine the user's *.chos* file to determine which OS to use.

For full integration, the batch scheduler system should be capable of running jobs in a CHOS OS. Furthermore, most users would prefer that jobs automatically run using the same OS under which the job was submitted. To accomplish this, a special job starter was written. Similar to the PAM module, the job start automatically performs the steps to configure CHOS for the user. However, rather than check a special file, the job starter looks for an environment variable (CHOS). If this variable is set to a valid path, then the appropriate steps are performed and the job automatically runs under the preferred OS. The PAM module, automatically sets this environment variable, so no further steps are required by the user.

The final crucial piece is the actual alternate operating systems. As stated earlier, only Linux distributions work with CHOS. The alternate OSs must be installed in some path accessed beneath the /chos tree, since other paths will not be accessible after the chroot. The alternate OSs can be either present on the local file system or accessed via a networked mounted file system such as NFS. The OSs includes the full tree that would normally be present on a natively installed system. This can be created by archiving and restoring an actual installation into a sub-tree. Alternatively, for distributions that use RPM for their package management, the "--root" directive can be used to install RPMs to an alternate path.

SECURITY

Code running inside the kernel or with high privileges needs to be designed with extra thought given to security and CHOS is no exception. The current release of CHOS has implemented several features to make it robust. First, the chroot system call has been imbedded in the handler that sets the path for the special link. This means that the calling routine no longer needs to run with root privileges. This does move some of the validation into the kernel making it more complex. However, since the interfaces between the kernel space and user space are more constrained, this approach is more robust.

The change root system call is privileged for a reason. If a user can change root to an arbitrary directory, they can easily construct a tree that could allow them to elevate their privileges. To prevent this, CHOS allows the administrator to limit which paths a users can select. These paths would point to the various trees the administrators has installed or validated.

While supporting additional distributions and releases might imply greater security risks, this need not be the case. All services would typically run out of the base operating system. So the administrator would focus their security efforts on this. Furthermore, set-UID programs can be disabled in the CHOS OSs, in order to limit the additional risks from these operating systems. The only remaining potential vulnerabilities would primarily be cross-user based exploits, which are generally less of concern. As a result, CHOS can potentially help improve security on a system. For example, if users need access to an older release with known vulnerabilities, the administrator can provide this environment under CHOS, while running key services under a more secure and better maintained base OS.

USES

While we initially designed CHOS to solve the immediate problem of different groups requiring different Linux distributions for production, we have since discovered other uses. For example, in the past we have designed sophisticated login files to initialize the environment for users in various groups. With CHOS, we could instead provide a specific CHOS OS for each group and do the customization on the CHOS OS. This means we can adjust the environment for a single group without inadvertently affecting other groups. CHOS can also be valuable for groups migrating to a new OS release or distribution. They can continue their production in the stable OS, while using CHOS to migrate and test applications in the new OS. CHOS can also allow the base OS to be upgraded independently of the OSs used by the users. This allows the administrator to keep the base OS up to date and secure with less impact on production users.

Perhaps a more intriguing use of CHOS is with the Grid. Currently, many projects developing Grid frameworks build binary packages and distribute these across the various resources. This approach typically assumes a standardized OS across the sites. As more large collaborations attempt to exploit the Grid this may present a problem similar to what was encountered on PDSF. Using CHOS, collaborations could distribute a CHOS OS along with the other packages. Using this approach the groups could be confident that the OS exactly matched the reference system and potential mismatches could be avoided.

CURRENT STATUS AND FUTURE DIRECTIONS

The version of CHOS which we have described in this paper is the current version, 0.4. Previous versions of CHOS have been used in production on PDSF for nearly the past year. While we have discovered a few initial flaws in our design, it has run almost problem free. Users often do not even realize that they are running on a system that does have there selected OS natively installed. All known issues with CHOS have been addressed or have a designed fix. In addition to these minor fixes, future releases will focus on simplifying installation and deployment.

CHOS has been tested with 2.4 and 2.6 kernels. Base operating systems under which CHOS has been successfully employed include RedHat, SuSE, Fedora, and Scientific Linux. CHOS OSs include RedHat, Fedora, Scientific Linux. In addition, CHOS has been tested with multiple versions of most of these distributions. A custom job starter has been used with Platform's LSF and the Sun GridEngine batch scheduling systems. CHOS is distributed as both a tar image and RPM format and is licensed under a modified BSD license. These packages can be obtained from the CHOS web site [3].

CONCLUSION

We have described a typical problem encountered on large shared clusters and presented our solution. As was stated earlier, CHOS has been in production on PDSF for nearly a year. It has enabled us to satisfy varying OS requirements presented by different groups. Furthermore, CHOS has given the administrators greater flexibility in upgrade schedules for the base OS. However, the real elegance of CHOS is that users often do not even realize that they are using it.

ACKNOWLEDGMENTS

This work was funded by the Department of Energy Office of Science under contract number DE-AC03-76SF00098.

REFERENCES

[1] Daniel P. Bovet and Marco Cesati, Understanding the Linux Kernel, 2nd Edition (2004).
[2] http://www.kernel.org/pub/linux/libs/pam/.
[3] http://www.nersc.gov/nusers/resources/PDSF/chos/