Presenting a Single System Image with Fine Granularity Mounts

Charles H. Sauer

IBM Advanced Engineering Systems
Austin, Texas 78758

ABSTRACT

In distributed system environments, a variety of administrative environments (system images) can be presented, reflecting different user requirements and administrative objectives. One of the most important system images is the so called "single system image." This paper provides a context and definition for single system image. It describes an effective approach to collecting multiple *UNIX*™ systems into a single system image, based on simple use of remote mounts at fine granularities, including individual files. The approach is designed to allow for replication of administrative files, e.g., /etc/passwd, and graceful reconfiguration of the system to accomodate planned outages and respond to unplanned outages. Experiences with this approach and *AIX*™ Distributed Services are summarized.

INTRODUCTION

In a distributed system environment, individual machines usually perform roles as servers (file, print, name, ...) and/or clients. Subsets of machines may be associated into administrative groups or the associations between machines may remain primarily pairwise and *ad hoc*. Figure 1 illustrates a typical software development environment. Some machines provide services to all other machines in the organization, e.g., network news, source control, special devices, etc. Some machines are administered directly by their owners and have only loose associations with other machines, e.g., the organization wide servers. Many of the other machines are collected into *single system images*, based on suborganizations. These machines are administered as a group, with the intent that users can use any of the machines equivalently. There will be inherent exceptions to this, e.g., some machines will have color displays and others will have monochrome displays. And even where the hardware configurations are the same, the end users will usually be able to distinguish one machine from another, e.g., by querying a machine readable serial number. But a successful approach will give users the illusion that all the machines are the same under most circumstances:

user accounts/passwords. A user can login to any of the machines using the same login name and the same password. Regardless of which machine is used, the user has the same home directory and execution environment. When the user changes his/her password, using the standard *passwd* command, the change is effective immediately on all machines in the single system image. When an administrator adds a new account, this is done once for all the machines.

availability. Even though one or more of the machines is unavailable, the rest of the machines are still able to function together and present the same system image, except for resources which exist only on unavailable machines. A machine which cannot connect to other machines is still usable.

administered services. System wide functions, e.g., print and mail service, function the same from machine to machine. If mail is sent to a particular user, it can be seen/handled on any of the machines.

This is not an exhaustive list, but is intended to be indicative. Wherever possible, the administrator should view the collection of machines as if it were one machine and use the same procedures that would be used on a single machine. We will use the above characteristics as an operational definition of "single system image" and discuss an approach which we believe is effective in meeting the definition.

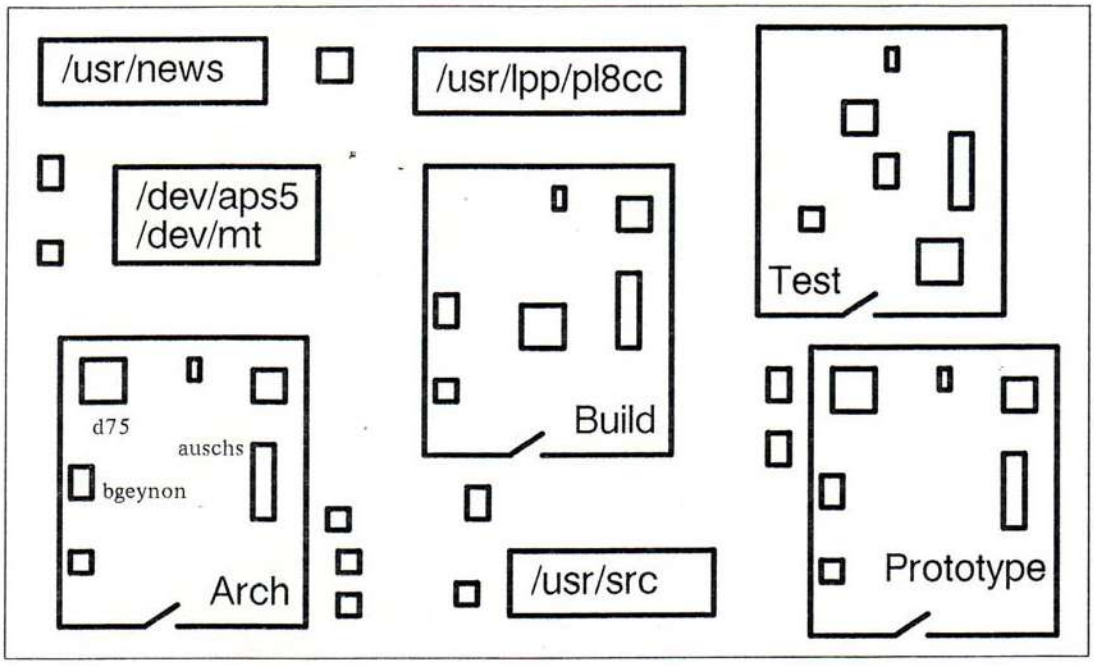

Figure 1 – Associations of machines.

Distributed Services (DS) provides distributed operating system capabilities for the AIX operating system. These include distributed file services with local/remote transparency, distributed interprocess communication and a number of administrative services. For background information on DS, see Sauer *et al* [1,2,3] and Levitt [4]. One of the design goals of DS was to provide support for mixed administrative environments, such as the one depicted in Figure 1, using the same protocols and conventions across the administrative environment. One of the cornerstones of this administrative flexibility is a general remote mount model. The focus of this paper is to show how the features of this remote mount model can be used to simply and effectively present a single system image. We first describe some of the characteristics of the DS mount model, then describe the approach to single system image, and finally discuss some additional related topics.

DISTRIBUTED SERVICES MOUNT MODEL

Distributed Services uses "remote mounts" to achieve local/remote transparency. A remote mount is much like a conventional mount in the Unix operating system, but the mounted filesystem is on a different machine than the mounted on directory. Once the remote mount is established, local and remote files appear in the same directory hierarchy, and, with minor exceptions, file system calls

have the same effect regardless of whether files(directories) are local or remote[1]. Mounts, both conventional and remote, are typically made as part of system startup, and thus are established before users login. Additional remote mounts can be established during normal system operation, if desired.

Conventional mounts require that an entire file system be mounted. Distributed Services remote mounts allow mounts of subdirectories and individual files of a remote filesystem over a local directory or file, respectively. File granularity mounts are useful in configuring a single system image. For example, a shared copy of `/etc/passwd` may be mounted over a local `/etc/passwd` without hiding other, machine specific, files in the `/etc` directory. Use of mounts at a fine granularity is key to this approach to single system image.

Virtual File Systems

The Distributed Services remote mount design is based on the Virtual File System approach used with NFS [5,6]. This approach allows construction of essentially arbitrary mount hierarchies, including mounting a local object over a remote object, mounting a remote object over a remote object, mounting an object more than once within the same hierarchy, mount hierarchies spanning more than one machine, etc. The main constraint is that mounts are only effective in the machine performing the mount.

In conjunction with using the Virtual File System concept, we necessarily have replaced the traditional `namei()` kernel function, which translated a full path name to an i-number, with a component by component `lookup()` function. `lookup()` is used both for local and remote path name resolution. The arguments to `lookup()` are a file *handle* representing a directory and the name of a component to be found in that directory. `lookup()` returns a handle for the component, if found. A handle is effectively a pointer to the on disk inode for the corresponding object and a generation number for that inode. The generation number is used for subsequent validity tests.

When a client successfully requests a mount from a server, it receives an *handle* for the object it is mounting and stores it in its mount table. When the client is parsing a file pathname, e.g., for `open()`, and encounters the mounted object, the handle is given to the server as an argument in the `lookup()` remote procedure call. Typically, the mounted object is a directory, and the server will lookup an object within that directory.

For example, let us suppose that a client mounts server's `/B` over `/a/b`. The client then opens `/a/b/c`. When the client gets to `b/c`, it passes the handle for `b` and the component `c` to the server, requesting the server to lookup and return a handle for `c` that can be used in the actual `open()` call. The server will return a handle for `/B/c`.

For file granularity mounts, the string form of the file name component is returned, along with the file handle of the (real) parent directory. This alternative to using the file handle for the mounted file allows replacement of the mounted file with a new version without loss of access to the file (with that name). (For example, when `/etc/passwd` is mounted and the `passwd` command is used, the old file is renamed `opasswd` and a new `passwd` file is produced. If we used a file handle for the file granulariity mount, then the client would continue to access the old version of the file. Our approach gives the, presumably intended, effect that the client sees the new version of the file.)

1. The traditional prohibition of links across devices applies to remote mounts. In addition, Distributed Services does not support direct access to remote special files (devices) and the remote mapping of data files using the AIX extensions to the shmat() system call. Note that program licenses may not allow execution of a remotely stored copy of a program.

There are several points to notice here. First, this approach is stateless in that the server can be recycled (e.g., powered off and on) and the handle(s) given to the client(s) performing a mount(s) is still valid, so the mount need not be repeated. This is true because the handle refers to an on disk structure, not an in memory structure. Second, the path resolution process must necessarily ignore mounts on the server, since these are not reflected in the on disk structures and are not necessarily repeated when the server is recycled. Third, as an immediate consequence, the client must explicitly perform all mounts "for itself," since it does not "see" mounts performed by the server.

Inherited Mounts

In constructing a single system image of Unix systems, it is desirable, if not necessary, to preserve the traditional directory hierarchy and conventions. All the machines in the single system image must see the same instances of /etc/passwd, /etc/hosts, ..., home directories, spool directories for mail, and so forth. However, it is also desirable/necessary to be able to access local equivalents of these files/directories so that they may be kept up to date with the shared copies. For example, /etc/passwd refers to a shared copy of the file, and /native/etc/passwd refers to the unshared local version. In general /native/a/b/... is established as the path to the local instance of /a/b/....

Without the concept of inherited mounts, discussed below, this implies that each machine would have to be doubly configured for it's local (device) mounts. E.g., if / (root), /u and /usr are on partitions /dev/hd0, /dev/hd1 and /dev/hd2, then the desired mounts could be achieved by the commands:

```
mount /dev/hd1 /u
mount /dev/hd2 /usr
mount / /native
mount /u /native/u
mount /usr /native/usr
```

Alternatively, the mount profile (`/etc/filesystems` in AIX) would contain an entry for each of these mounts. If another disk was added to hold /usr/src, then two profile entries would be needed, one for mount /usr/src and one for mounting /native/usr/src.

Distributed Services implements inherited mounts on top of virtual file systems. There is a `mntctl()` system call and corresponding remote procedure call. One of the options of `mntctl()` is to query and return a list of all mounts currently in effect on a given server. The `mount` command in AIX supports a `-i` (inherited) flag which causes the query to be performed and the additional mounts to be made. For the above example,

```
mount -i / /native
```

would have the same net effect as the three separate mount commands for the /native subtree. When additional device mounts are configured, this single mount command still provides the desired effect of an aliased naming path for the local instance of the file hierarchy. Additional examples of motivation for inherited mounts are given in [3].

PRESENTATION OF SINGLE SYSTEM IMAGE

Objectives

The configuration is managed by a few simple profiles. It should be easy to add/delete machines and users, and to make other configuration changes.

All of the machines in the single system image cluster should use exactly the same configuration files, i.e., there is no distinction between the profiles on the server for /etc/passwd and related files and the ones on the clients.

As a result of the above, it should be simple to reconfigure to use a different server for the /etc files, either because of planned outages of the existing server or because of failure of the existing server.

Client machines should recognize when the server is unavailable, and switch to alternate copies of administrative files and other shared files.

Local replicated copies of the administrative files should be periodically updated, so that if there is an unplanned outage of the server, the other machines have up to date copies.

If a machine providing some of the home directories is unavailable, a user should discover this immediately at login time, and and be able to either use an alternate home directory or wait until the machine becomes available again.

General Approach

The discussion will focus on administrative files, e.g., /etc/passwd, and data files, e.g., home directory subhierarchies and mail spooling areas. Assuming, for the moment, a homogeneous processor architecture, executable files may be viewed between two extremes: (1) all of the machines in the cluster have full copies of the executable code, and so there is no sharing of executables, or (2) there is a single shared copy of each executable file. The first extreme has the potential for inconsistency amongst the multiple copies, administrative burden to ensure that inconsistent copies are not present, and wastage of disk space for the redundant copies. The second extreme has the limitation that executables will be unusable if the shared copy is not accessible. An administrator will typically choose a policy in between the extremes, e.g., that the kernel and the files in /etc and /bin are replicated, but that other executables are shared. The approach discussed below supports and allows flexibility in determining such policies. However, AIX and DS have other administrative mechanisms, known as "code serving" which address these policies in detail, so we focus on the administrative and data files.

Where heterogeneous processor architectures are involved, the motivation for a separate mechanism for code serving is stronger, since the mechanisms below should not be used for sharing binary executables across different processor architectures. The files explicitly shared in the mechanisms described below are in ASCII format, and are suitable for sharing across heterogeneous processor architectures. Thus the mechanisms themselves will work across heterogeneous processor architectures. However, in the heterogeneous environment, machine boundaries are much more likely to be visible, e.g., due to byte order considerations in application data. More stringent requirements must be placed on application code in such an environement, if the illusion of a single system is to be preserved.

Configuration Files

`/etc/adminserver`. One machine is designated as the "administrative server" and is the machine that has the disk copies of the shared administrative files such as /etc/passwd. The ad-

ministrative server can be changed while machines are in operation, as discussed below. The name of the administrative server is stored in /etc/adminserver.

/etc/SSImachines. This file lists the names of all machines in the single system image (including the administration server).

/etc/server.files. This file lists individual files that will be shared based on the administrative server's copy. For example, this list might include

```
/etc/passwd
/etc/group
/etc/motd
/etc/qconfig              (AIX printer configuration file)
/etc/hosts
/etc/hosts.equiv
/etc/adminserver
/etc/SSImachines
/usr/adm/user.cfile       (AIX adduser defaults)
/etc/server.files
/etc/server.dirs          (see below)
/etc/remounts.list        (see below)
/etc/ug.SSI               (for id translates – see below)
/etc/oug.SSI
/etc/opasswd
/etc/ogroup
/etc/umountd.c            (source for umountd – see below)
/usr/adm/newuser.sys      (AIX adduser defaults)
/usr/adm/newuser.usr
```

Though the list could be longer or shorter, roughly this set of files has been appropriate in our experience.

/etc/server.dirs. This file lists directories, other than home directories, that will be shared based on the administrative server's copy. Assuming that code serving is handled separately, this list might include

```
/usr/mail
/usr/lib/news
/usr/spool/news
/usr/man
...
```

If code serving is not handled separately, then /usr/bin, /usr/lib, ... might be added to this list.

/etc/remounts.list. This file lists files and directories that will be unmounted when it is detected that the server is inaccessible and remounted when the server becomes accessible again. This is handled by the umountd daemon, discussed below. This list will be a subset of the combined lists in server.files and server.dirs, e.g.,

```
/etc/passwd
/etc/group
/etc/motd
/etc/qconfig                    (AIX printer configuration file)
/etc/hosts
/etc/hosts.equiv
/etc/adminserver
/etc/remounts.list
/usr/mail
```

This is a subset of the combined list oriented toward normal operation when the administrative server is inaccessible. It is a subset because some operations, e.g., changing passwords, presumably will be deferred when the administrative server is inaccessible, and some directories, e.g., /usr/man and /usr/spool/news, are likely to be empty, except on the administrative server, and thus uninteresting when that machine is unavailable.

`/etc/ug.SSI`, `/etc/oug.SSI`. DS provides general translation mechanisms for user and group id translation [1,2]. For machines within the cluster, there should be one to one translations, so that numeric id's are the same on all machines in the cluster, but machines in the cluster may also need translates to other machines outside the cluster. ug.SSI is used for a cluster wide definition of the translates. For brevity, we will not discuss the content of these files.

Home Directories

For sake of simplicity, it is assumed that home directories' pathnames have the form .../machine/user, where "machine" is the name of the machine where the home directory is actually stored. Even though paths are of this form, users will see the same actual home directory on each machine of the cluster, e.g., in our environment, when user sauer is logged into machine d75, his home directory is still /u/auschs/sauer, since his home directory is stored on auschs. This is a minor sacrifice of transparency, since users usually do not use rooted paths to get to their home directories — their home directory is listed in /etc/passwd, so that is where they start, cd takes them back there, and the c shell "~" notation is often used to get to other users home directories, e.g., cd ~dale. Shipley has proposed a similar convention for sharing home directories [7].

Having paths of this form allows each machine to simply mount the home directories stored on other machines, e.g., mount -n auschs /u/auschs /u/auschs, or, in general,

```
for i in 'cat /etc/SSImachines'
do
   if [ $i != $myname ]
   then
      mount -i -n $i /u/$i /u/$i
   fi
done
```

This is a slightly simplified fragment from /etc/SSImounts, discussed below.

Machine Initialization

As init processing, after normal standalone initialization, e.g., fsck and device mounts, /etc/rc.DS starts DS and then runs /etc/SSImounts. SSImounts runs in the background so that local operations can begin without server availability. SSImounts runs on all machines, including the ad-

minserver. Following are simplified sketches taken from SSImounts. Error checks, touches/mkdirs for mount points, precautionary umounts, etc. are omitted:

Initial mounts from adminserver, updating local copies of files:

```
if [ $myname != $adminserver ]
then
        until mount -i -n $adminserver /native /$adminserver
        do
                sleep $delay
        done
        for i in 'cat /etc/server.files'
        do
                cp -p /$adminserver$i $i
                mount -n $adminserver /native$i $i
        done
        for i in 'cat /etc/server.dirs'
        do
                mount -i -n $adminserver /native$i $i
                if [ $i = '/usr/mail' ]
                then
                        /etc/movemail & # see below
                fi
        done
fi
```

Start /etc/umountd

```
make umountd
/etc/umountd &
```

Update user/group ids.

```
cmp /etc/ug.SSI /etc/oug.SSI
if [ $? -ne 0 ]
then
        dsldxprof -a -f /etc/ug.SSI
        if [ $? -eq 0 ]
        then
                cp -p /etc/ug.SSI /etc/oug.SSI
        fi
fi
```

Mount home directories. This is as indicated in the previous fragment, except that the mounts are retried asynchronously in the background so that availability of any given machine doesn't delay availability of home directories from other machines.

Once these steps have been performed, then the machine has joined the single system image.

/etc/movemail. Mail received while the administrative server is not available will be kept in the native spool directory, /usr/mail. movemail moves mail from the native spool directory to the shared

spool directory whenever the shared directory is mounted, either by SSImounts or remounts (see below).

umountd

The key remaining topic is the daemon umountd. umountd uses a polling loop, performing the following functions and then sleeping until repeating the functions. The default sleep time is 60 seconds.

Detection of server inaccessibility. umountd attempts to open each of the files listed in /etc/server.files. If an open fails, umountd assumes the server is inaccessible and executes /etc/remounts. /etc/remounts unmounts all of the files in /etc/remounts.list and then attempts to remount them. remounts will execute movemail after successfully remounting /usr/mail. (umountd runs on adminserver, but skips these steps.)

Updating modified files. umountd determines whether any of the files in /etc/server.files have been updated. If so, umountd locks the server copy and updates the native copy. (umountd running on adminserver skips these steps.)

Detection of configuration changes. If key configuration files, e.g., /etc/adminserver or /etc/server.files, have been updated, umountd exec's SSImounts. SSImounts applies the changes and then starts a fresh version of umountd, as indicated above.

There are subtleties of locking and timing which we omit for brevity. Starting with DS 1.2, the source for umountd.c is included in the DS samples directory, along with the installation documentation, installation command and so forth.

Note the power of the above mechanisms. By simply changing the name of the server in /etc/adminserver, e.g., for a scheduled outage of the adminserver, the machines in the cluster will shift to the new adminserver in a couple of minutes, without rebooting any of the machines or otherwise significantly disrupting users. For an unscheduled server outage of significant duration, a switchover to a different server can be accomplished by changing the adminserver file on each of the other machines and rebooting them. In either case, scheduled or unscheduled, the original server can rejoin the cluster as a client when it is ready to rejoin the cluster, and then assume the server role again when its configuration files have been updated.

SUMMARY

We believe this approach effectively meets the operational definition given in the introduction, in regard to user accounts, data, availability, and administered services. The mechanisms are designed to be simple to apply and administer, yet highly effective in presenting the image of a single system. These mechanisms are complementary to the underlying file system mechanisms of Distributed Services, and orthogonal to other enhancements such as the code server mechanisms. Thus the same underlying mechanisms are applied across the distributed environment, for example the abstraction of our software development environment depicted in Figure 1. In addition, significant administrative flexibility is present, as suggested in the preceding paragraph. These concepts could be applied to other distributed file systems supporting fine granularity mounts.

REFERENCES

1. Charles H. Sauer, Don W. Johnson, Larry Loucks, Amal A. Shaheen-Gouda, Todd A. Smith, "RT PC Distributed Services: Overview," *Operating Systems Review 21*, 3 (July 1987) pp. 18–29.
2. Charles H. Sauer, Don W. Johnson, Larry Loucks, Amal A. Shaheen-Gouda, Todd A. Smith, "RT PC Distributed Services: File System," *;login: 12*, 5 (September/October 1987) pp. 12–22.

3. Charles H. Sauer, Don W. Johnson, Larry Loucks, Amal A. Shaheen-Gouda, Todd A. Smith, "Statelessness and Statefulness in Distributed Services," *UniForum 1988*, Dallas, Texas, February 1988.
4. Jason Levitt, "The IBM RT Gets Connected," *BYTE 12*, 12 (1987) pp. 133-138
5. R. Sandberg, D. Goldberg, S. Kleiman, Dan Walsh and B. Lyon, "Design and Implementation of the Sun Network File System," *USENIX Conference Proceedings*, Portland, June 1985.
6. Sun Microsystems, Inc., *Networking on the Sun Workstation*, Feburary 1986.
7. M. Shipley, "The Virtual Home Environment," *UniForum 1988*, Dallas, Texas, February 1988.