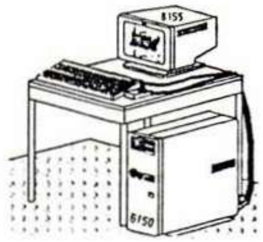
RT PC Distributed Services

Charles H. Sauer
Don W. Johnson
Larry K. Loucks
Amal A. Shaheen-Gouda
Todd A. Smith

IBM Industry Systems Products
Austin, Texas 78758



ABSTRACT

RT PC Distributed Services provides distributed operating system capabilities for the AIX[1] operating system. These include distributed files with local/remote transparency, a form of "single system image" and distributed interprocess communication. Applications, including data management/data base applications, can typically be used in the distributed environment without modification to existing *object* code. Distributed Services is architected for strong compatibility with AIX, the UNIX[2] operating system, IBM architectures such as SNA and some of the architectures of Sun Microsystems'[3] NFS. The Distributed Services implementation includes caching mechanisms and other algorithms designed for high performance without sacrificing strict functional transparency. This paper describes the key characteristics and decisions in the design of Distributed Services.

INTRODUCTION

The widespread availability of personal computers and workstations has lifted many of the limitations of the previous generation of time sharing systems, by giving individuals a great deal of dedicated computing capacity and previously unavailable capabilities, e.g., memory-mapped displays. However, having machines[4] dedicated to individuals means the loss of important, previously taken for granted, system characteristics. These include the ability to share files and peripherals, the ability to use any one of a pool of equivalent machines, and the ability of one person to act as administrator for a number of other users. File transfer and remote login facilities can alleviate these losses somewhat, but a better solution is to provide distributed operating services, such as access to remote files using the same mechanisms as access to local files.

There have been numerous research efforts and a number of products with goals and characteristics similar to Distributed Services. Perhaps the best known of these are LOCUS [Popek *et al* 1981, Popek and Walker 1985], NFS [Sandberg *et al* 1985, Sun 1986] and RFS [Rifkin *et al* 1986]. We studied each of these, and many other, previous designs. As we describe Distributed Services, we will contrast our design with some of its predecessors.

The remainder of the paper is organized as follows: First, we will summarize our view of the administrative environments anticipated for Distributed Services installations and our objectives for Distributed Services. Then we describe the overall characteristics of Distributed Services. Finally, we cover our definition and treatment of "single system image" in additional detail.

1. AIX is a trademark of International Business Machines Corporation
2. Developed and licensed by American Telphone and Telegraph. Unix is a registered trademark in the U.S.A. and other countries.
3. Sun Microsystems is a trademark of Sun Microsystems, Inc.
4. We use "machine" to indicate a generic computer — a personal computer, workstation or larger computer.

ADMINISTRATIVE ENVIRONMENTS

Assumptions about administrative environments are fundamental to design of distributed systems, yet administrative concerns are often omitted from primary consideration. Two primitive elements can be identified in the environments we anticipate: multi-machine clusters and independently administered machines.

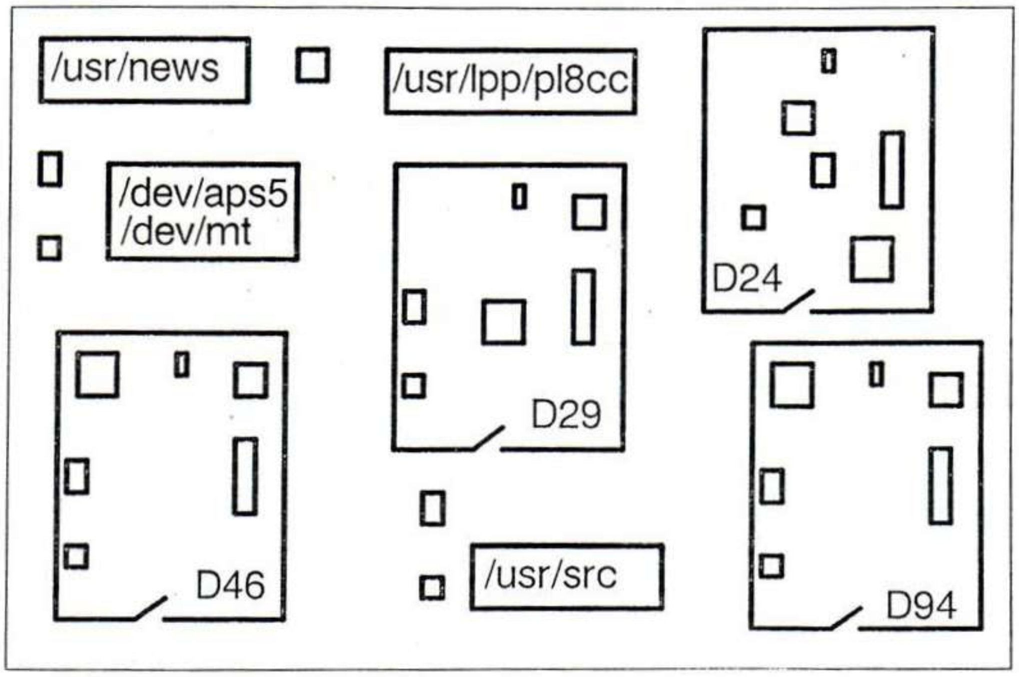


Figure 1. Multi-machine clusters and separately administered machines.

Multi-machine Clusters

All of the machines in a multi-machine cluster are administered uniformly, let us assume by the same person. The machines in the cluster are likely owned by the same department, and members of the department can use the machines equivalently, i.e., the multiple machines present a "single system image." Regardless of which machine in the cluster is used, login ids and passwords are the same, the same programs/data files/directories are accessible and authorization characteristics are the same. The large boxes marked D46, D29, ... in Figure 1 are meant to suggest multi-machine clusters, with each small box representing a machine. Of course, the machines may be dispersed geographically within the limitations of the networks used — they are shown together in a room for convenience in drawing the figure.

Independently Administered Machines

The other primitive element is the independently administerd machine. These machines fall into two subcategories: servers and private machines. Servers in this sense are not functionally different from other servers (e.g., file or device servers) which may be found within multi-machine clusters, but they are administered independently from other machines/clusters. The boxes with path names in Figure 1 are intended to suggest file/device servers administered independently. Other machines may be independently administered because the owners are unwilling to allow others to assume administrative responsibility. Both of these subcategories can be considered degenerate cases of the multi-machine cluster, but it is convenient to discuss them separately.

Networks of Multi-Machine Clusters/Independently Administered Machines

As organizations evolve toward connecting all machines with multimegabit per second networks, administrative configurations such as the one depicted in Figure 1 will inevitably occur. It will be required that all of the machines be able to communicate with one another, and a high degree of network transparency will be required. But administrative clustering of machines according to subgroups of the organization will be natural, and cooperation/transparency within these clusters will usually be a primary issue. Authorization characteristics will vary across the clusters/independent machines. Organizations will change, and correspondingly, machines will be added to/deleted from clusters, and clusters/machines will be added to/deleted from networks. Distributed system designs must be prepared to cope with these configurations and changes in configuration.

DISTRIBUTED SERVICES DESIGN GOALS

The primary design goals in our design of Distributed Services were

Local/Remote Transparency in the services distributed. From both the users' perspective and the application programmer's perspective, local and remote access appear the same.

Adherence to AIX Semantics and Unix Operating System Semantics. This is corollary to local/remote transparency: the distribution of services cannot change the semantics of the services. Existing object code should run without modification, including data base management and other code which is sensitive to file system semantics.

Remote Performance = Local Performance. This is also corollary to transparency: if remote access is noticably more expensive, then transparency is lost. Note that caching effects can make some distributed operations *faster* than a comparable single machine operation.

Network Media Transparency. The system should be able to run on different local and wide area networks.

Mixed Administrative Environments Supported. This was discussed in the previous section. Additionally, services must be designed to make the administrator's job reasonable.

Security and Authorization Comparable to a Single Multiuser Machine.

DISTRIBUTED SERVICES FILE SYSTEM

Remote Mounts

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[5]. 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. Directory granularity and file granularity mounts are now also allowed with AIX local mounts.

Distributed Services does not require a file system server to export/advertise a file system before it can be mounted. If a machine can name a directory/file to be mounted (naming it by node and path within that node), then the machine can mount the directory/file if it has the proper permissions. The essential permission constraints are

1. Superuser (root) can issue any mount.
2. System group[6] can issue local device mounts defined in the profile /etc/filesystems.
3. Other users/groups are allowed to perform remote directory/file mounts[7] if the process has search permission for the requested directory/file, owns the mounted upon object (directory/file) and has write permission in the parent directory of the mounted upon object.

The objectives of these constraints are to maintain system integrity but allowing users the flexibility to perform "casual" mounts.

File System Implementation Issues

Virtual File Systems. The Distributed Services remote mount design uses the Virtual File System approach used with NFS [Sun 1986]. 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.

Inherited mounts. It is desirable for one machine to be able to "inherit" mounts performed by other machines. For example, if a machine has mounted over /usr/src/icon and a second machine then mounts the first machine's /usr/src, it might be desired that the second machine see the mounted version of /usr/src/icon. This would not happen in the default case, but Distributed Services provides a query facility as part of a new mntctl() system call. The mount command supports a -i (inherited) flag which causes the query to be performed and the additional mounts to be made. By use of inherited mounts, clients of a file server need not know of restructuring of the server's mounts underneath the initial mount. For example, if a client always uses an inherited mount of /usr/src, it does not need to change it's configuration files when the server uses additional mounts to provide the subdirectories of /usr/src.

5. 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.
6. In AIX, we have given the system group (gid 0) most of the privileges traditionally restricted to the superuser. Only especially "dangerous" or "sensitive" operations are restricted to the superuser [Loucks 1986].
7. Remote device mounts are not supported, but the only practical effect is that a remote device that is not mounted at all at the owning machine can not be remote mounted. This is likely desirable, since this situation is only likely to occur during maintenance of the unmounted device.

lookup. 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. For file granularity mounts, the string form of the file name is used, 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.)

Statelessness and Statefulness. One of the key implementation issues is the approach to "statelessness" and "statefulness." Wherever it is practical to use a stateless approach, we have done so. For example, our remote mounts are stateless. However, in some areas where we believe a stateful approach is necessary, we maintain state between server and client and are prepared to clean up this state information when a client or server fails. In particular, we maintain state with regard to directory and data caching, so that cache consistency can be assured.

Directory Caching. Use of component by component lookup means, in the worst case, that there will be a lookup() remote procedure call for each component of the path. To avoid this overhead in typical path searches, the results of lookup() calls are cached in kernel memory, for directory components only. Cached results may become invalid because of directory changes in the server. We believe that state information must be maintained for purposes of cache validity. Whenever any directory in a server is changed, client directory caches are purged. Only clients performing a lookup() since the previous directory change are notified, and they, of course, only purge the entries for the server that had the directory change. This purpose of this strategy is to keep the directory cache entries correct, with little network traffic.

Data Caching. Distributed Services uses data caching in both client and server, to avoid unnecessary network traffic and associated delays. The caching achieves the traditional read ahead, write behind and reuse benefits associated with the kernel buffer cache, but with both client and server caches. As a result, read ahead(write behind) can be occuring in the client cache with regard to the network and in the server cache with regard to the disk. *As a result, disk to disk transfer rates to/from remote machines can be substantially greater than local rates.* In AIX we have carefully tuned the local disk subsystem, yet use of cp for remote files yields significantly higher disk to disk throughput than for local only files. Note that stateless designs may not support write behind, in order to guarantee that all data will be actually on the server's disk before the write rpc returns to the client.

Data Cache Consistency. In general, it is difficult to keep multiple cached data blocks consistent. We designed a general cache invalidation scheme, but chose to implement instead a state machine based on current opens of a given file. We emphasize that this mechanism is applied at a file granularity, and that it is strictly a performance optimization — the mechanism is designed to preserve the traditional multireader/multiwriter semantics of the Unix file system. Any particular file will be in one of the following states:

1. Not open.
2. Open only on one machine This may be a different machine than the server for the file. ("async mode")
3. Open only for reads on more than one machine. ("read only mode")

4. Open on multiple machines, with at least one open for writing. ("fullsync mode')

We believe that the read only and async modes are dominant in actual system operation, and our client caching applies to these modes only. In fullsync mode, there is no client caching for the given file, but the server caches as in a standalone system.

Close/Reopen Optimization. A frequent scenario is that a file is closed, say by an editor, and then immediately reopened, say by a compiler. Our data cache consistency mechanisms are extended to allow reuse of cached data blocks in the client data cache, if and only if the file is not modified elsewhere between the close and subsequent reopen.

Kernel Structured Using Sun "vnode" Definition. We have used the Sun vnode data structure [Kleinman 1986] to support multiple file system types in the AIX kernel. This allows a clean division between the local AIX filesystem code and the remote filesystem code.

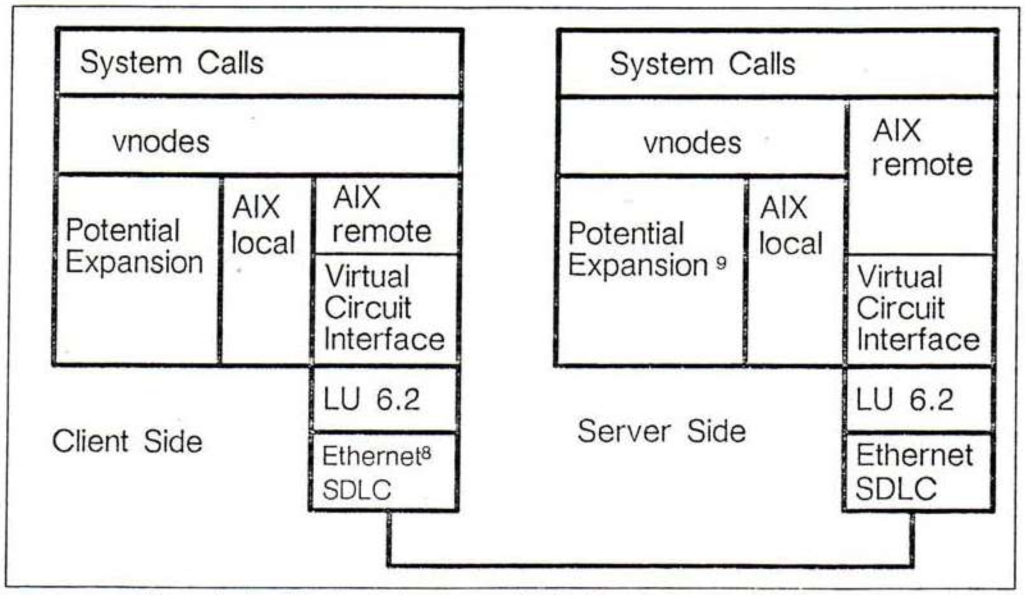


Figure 2. Architectural Structure of Distributed Services File System

8. Ethernet is a trademark of Xerox Corporation.
9. This is not intended as speculation of future products.

Virtual Circuit Interface. Distributed Services assumes virtual circuits are available for network traffic. One or more virtual circuits must remain in force between a client with a file open and the server for that file. (The mere existence of a remote mount does not require retention of a virtual circuit.) Execution of cleanup code, e.g., decrementing usage counts on open files, will be triggered by loss of a virtual circuit. The architecture of Distributed Services includes a *Virtual Circuit Interface* (VCI) layer to isolate the Distributed Services code from the supporting network code. Our current code uses the SNA LU 6.2 protocol to provide virtual circuit support, but, potentially, another connection oriented protocol, e.g., TCP, could be used. The basic primitives of the VCI are the `dsrpc()`, `dsrpc_got()` and `dsgetdata()` functions. `dsrpc()` acquires a connection with a specified machine and then issues `dsrpc_got()` to invoke a function on that machine. `dsrpc_got()` is called directly if the caller has a previously established connection available. Both of these calls return without waiting for the result of the remote function, allowing continued execution on the calling machine. dsgetdata() is used to request the result of a remote functions; it will wait until the result is available.

SNA LU 6.2 Usage. We chose to use LU 6.2 because of its popular position in IBM's networking products and because of its technical advantages. In particular, LU 6.2 allows for "conversations" within a session. Conversations have the capabilities of virtual circuits, yet with low overhead of the order typically associated with datagrams. Typically, one or two sessions are opened to support the flow between two machines, regardless of the number of virtual circuits required. We have carefully tuned the LU 6.2 implementation, exploiting the fully preemptive process model of the AIX Virtual Resource Manager [Lang, Greenberg and Sauer 1986]. By properly exploiting the basic architecture of LU 6.2 and careful tuning, we have been able to achieve high performance without using special private protocols [Popek and Walker 1985] or limiting ourselves to datagrams.

The AIX implementation of LU 6.2 supports both Ethernet and SDLC transport. The AIX LU 6.2 and TCP/IP implementations are designed to coexist on the same Ethernet — in our development environment, we use both protocols on a single Ethernet, e.g., TCP for Telnet and/or X Windows and LU 6.2 for Distributed Services.

DISTRIBUTED PROCESS SUPPORT

Approaches to Distributed Process Support

Unlike distributed file systems, where there seems to be emerging consensus in the technical community on basic concepts, e.g., use of remote mount approaches, there is no consensus on mechanisms for distributed process support. For examples, LOCUS has chosen to distribute the traditional fork and pipe mechanisms, NFS provides the Sun RPC interface for interprocess communication and System V.3 provides the streams interface for interprocess communication. We have chosen to provide a distributed version of the AIX message queues. In addition, base AIX provides facilities for less transparent network wide interprocess communication.

Distributed Message Queues

The base AIX message queue definition is a superset of the System V definition. The primary extension is the provision of the msgxrcv() call, which provides additional information about the sender of the message, e.g., the effective userid and groupid, so that the recipient can be more selective in acting upon the message received, and a time stamp.

We expect that distributed message queues will typically be used to communicate with a server process, e.g., a print spooler, without requiring the client and server to be aware of whether or not they are on the same machine. To provide a distributed version of message queues, we have done the following:

1. The system call msgget(), which takes a 32 bit key as an argument and returns a msqid for the corresponding queue, has been modified to do a table lookup to see if the key has been registered as remote. This is done within the kernel proper, so an existing object module which received a key as a parameter and invoked msgget() would invoke the distributed version. If the queue is remote, a request, including a remote key found in the lookup, is sent to the remote machine to create or find the message queue, as needed. The remote machine returns a remote msqid, the local kernel creates/finds an entry in its own tables to give the local msqid for the remote queue and returns the local msqid. (The local to local case is handled using these same tables and mechanisms.)

2. When the system calls msgctl(), msgsnd(), msgrcv() and msgxrcv() are invoked, the msqid table is searched to find the location of the message queue, and if the queue is remote, the operation is sent on to the remote machine.
3. A new system call, loadtbl(), is used to load the table which lists keys and ids of remote message queues. (loadtbl() is a general purpose mechanism which is also used by Distributed Services for loading the uid/gid translate tables discussed below, and which is used by base AIX for loading tables used for national language support.) loadtbl() is invoked at startup time to initialize the tables and is also invoked while the system is running, when the tables need to be updated.
4. ftok() has been modified to not return keys less than 0x1000000, and the remaining key space is used by new services/profiles. The new services, create_ipc_prof() and find_ipc_prof(), have been provided for creating/finding a profile entry which contains a symbolic name and both a local and remote key for the queue.
5. Additional commands and menus have been provided for creating/updating the tables used by the above services.

We have not provided corresponding distributed versions of shared memory and semaphores.

AIX Remote Process Support

In addition to the distributed message queues of Distributed Services, base AIX provides non-location transparent support mechanisms for remote processes. These mechanisms are enhanced by the distributed file system support. They include:

SNA LU 6.2. The previously discussed remote procedure call support of LU 6.2 is available directly to user level processes.

SNA Services System Resource Controller (SRC). The SRC provides mechanisms for starting and signalling remote processes. Menu interfaces are provided for managing these mechanisms.

AIX TCP/IP. Remote print, remote login, and remote execution facilities are provided, in addtion to the base TCP/UDP/IP protocols. (Other services/protocols, e.g., ftp and smtp, are also available.)

DISTRIBUTED SERVICES SECURITY AND AUTHORIZATION

Encrypted Node Identification

When considering networks of the sort suggested by Figure 1, it is clear that each machine needs to be suspicious of the other machines. If a machine is going to act as a server for another, it should have a mechanism to determine that the potential client is not masquerading. The AIX implementation of SNA LU 6.2 provides an option for encrypted node identification between a pair of communicating machines. The identification is by exchange of DES encrypted messages. The identification occurs at session establishment time and at random intervals thereafter. Once a client/server have each determined that the other is not masquerading, then they can take appropriate actions authorized according to (the translated) userid's/groupid's associated with each request.

Userid/Groupid Translation

There are a number of reasons why a common userid space and a common group id space are impractical in the environment of Figure 1:

1. An individual machine, whether a private machine or a server, should not be required to give superuser (root) authority to a request from a process

with root authority on another machine. Rather, it should be possible to reduce the authority of the remote process. The reduced authority may retain some administrative privileges, may be that of an ordinary user or may be no access at all, depending on the preferences of the administrator of the individual machine. Similar statements apply to the cluster of machines.

2. A user may have logins provided by several different administrators on several different machines/clusters, and these will typically have different numeric userids. When that user uses different machines, he/she should have access to his/her authorized resources on all machines in the network.
3. Previously operating machines may join a network or move to a new network, and existing networks may merge. When this happens, there may be different users/groups with the same numeric ids. Such reconfiguration should be possible without requiring users/groups to change numeric ids or changing userids/groupids in all of the inodes.

Our response to these requirements is to define a network wide ("wire") space of 32 bit userids and groupids. Each request leaving a machine has the userid translated to the wire userid and each request entering a machine has the wire userid translated to a local userid. The above requirements are met by proper management of the translations.

DISTRIBUTED SERVICES ADMINISTRATION

In addition to the normal system profiles, e.g., /etc/filesystems, there are profiles for both the SNA support and for Distributed Services. With these new profiles, we have taken care to organize the directories containing the profiles so that we can use remote mounts to administer remote machines, without use of remote login (or roller skates). For Distributed Services, there are three profiles, for machine ids and passwords, for userid/groupid translation and for registry of message queues.

Part of the AIX design is provision of a user interface architecture for a screen oriented ("menu") interface, to simplify system management and usage [Kilpatrick and Green 1986, Murphy and Verburg 1986]. Configuration of both SNA and Distributed Services, i.e., management of the SNA and Distributed Services profiles, is normally performed using menus conforming to this user interface architecture.

DISTRIBUTED SERVICES "SINGLE SYSTEM IMAGE"

Our definition of "Single System Image" is as follows: *Users of the given system, users of external systems which communicate with the given system and application programmers ARE NOT aware of differences between single and multiple machine implementation. System administrators and maintenance personnel ARE aware of distinctions amongst machines.*

User/Programmer View of Distributed Services Single System Image

Though there are inherent exceptions to this, e.g., the uname() system call is designed to return the machine name, we believe that Distributed Services largely meets this definition. The key mechanism is to be able to properly configure the several machines so that they share the files and directories which matter to the user and the application programmer. These include basic profiles such as /etc/passwd, home directories, and directories containing applications, commands and libraries. Figure 3 sketches one such possible configuation.

Once this is accomplished, most of the desired properties just fall in place. The login process will be the same because of the sharing of /etc/passwd related files. Normal file system manipulations and applications work in the shared directories. Administrative commands for ordinary users, e.g., passwd, also work properly if they follow reasonable conventions (we had to rework several commands such as passwd, as discussed below.)

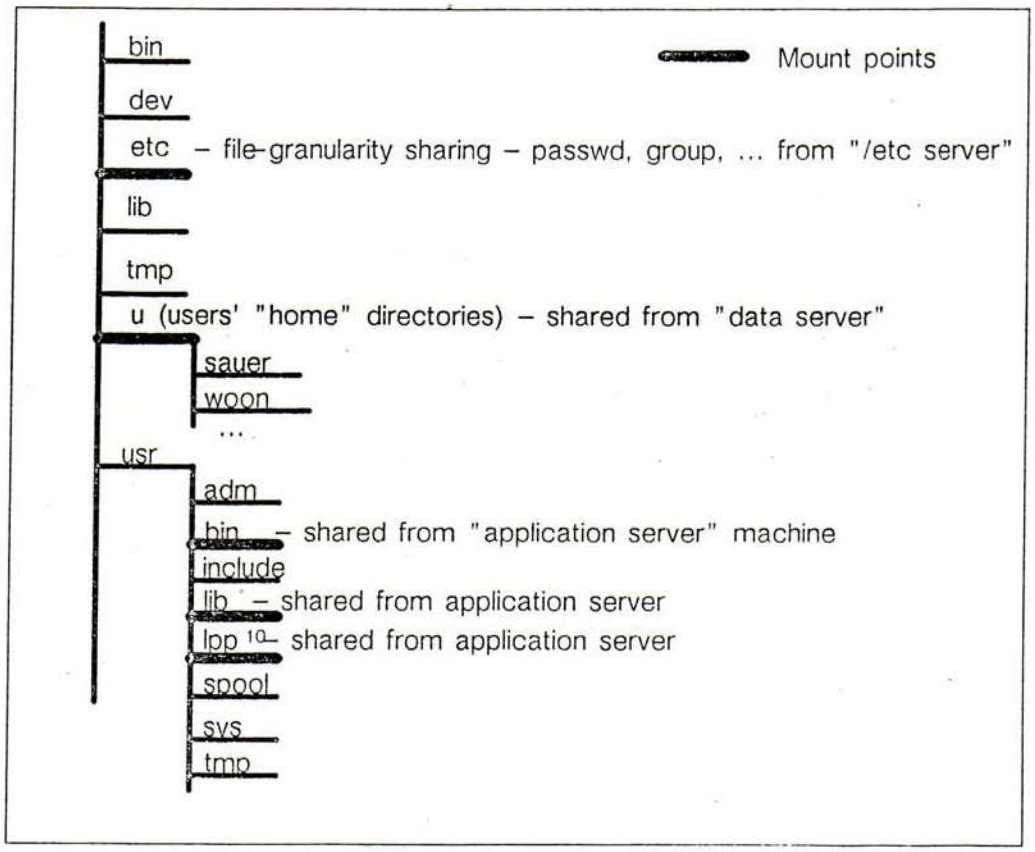

bin
Mount points
dev
etc – file-granularity sharing – passwd, group, ... from "/etc server"
lib
tmp
u (users' "home" directories) – shared from "data server"
sauer
woon
...
usr
adm
bin – shared from "application server" machine
include
lib – shared from application server
lpp[10] – shared from application server
spool
sys
tmp

Figure 3. Example Shared File System.

10. An AIX convention is to place most applications in subdirectories of /usr/lpp.

Administrator's View of Single System Image Configurations

Some of the administrator's tasks must be be performed for each machine individually. For example, the administrator must install and configure AIX and Distributed Services on each machine. Other tasks can be performed once for the entire single system image cluster. For example, installation of an application, in the usual case where the installp command retrieves files from diskette and places them in the appropriate subdirectory of /usr/lpp, need only be done once, assuming it is done after normal system startup. Similarly, the adduser command, which creates an entry in /etc/passwd, creates a home directory and copies standard files to the home directory, need only be applied once.

Routine maintenance, e.g., backing up and restoring files, can be done for the system as a whole while the system is in normal operation. Error logs are intentionally kept separately for each machine — otherwise, the first problem determination step would be to isolate the anomalous machine. Some maintenance operations, e.g., image backups of disks and hardware diagnostics, are necessarily performed on a machine by machine basis, while the machine is in maintenance mode.

Implementation Issues in Distributed Services Single System Image

There is an obvious question of ordering in starting the separate machines. We have added a number of options to the mount command and /etc/filesystems to allow simple retry mechanisms to be executed in the background when initial mount attempts fail. This is done to allow arbitrary ordering of the startup of machines.

Many of the interesting commands, e.g., passwd, use private locking mechanisms, e.g., based on creating/deleting dummy lock files. We have had to modify a number of these commands to use the lockf() system call.

A more subtle issue is the "copy/modify/unlink/relink" idiom used in a number of interesting programs such as editors. This idiom does not work in all cases of file granularity mounts, because a client may be attempting to violate the prohibition of linking across devices. In more detail, the idiom is as follows, for updating foo in the current directory:

```
1.   cp foo .foo.tmp
2.   modify .foo.tmp
3.   rm foo
4.   ln .foo.tmp foo
5.   rm .foo.tmp
```

If foo is a file mount from a different device, step 4 will fail. We have had to modify several programs to do a copy if the link step (4) fails. Note that this is not a problem with directory mounts, only file granularity mounts.

There is also a potential problem with routines such as mktemp() and tempnam(), which use process ids to generate unique file names. Since process ids are not unique across machines, we have modified these routines to use the machine id as well as the process id in deriving a file name. (The modified versions of these routines are packaged with AIX, so that object code does not have to be recompiled/relinked to run with Distributed Services.)

Separate Machine Operation

Clearly, it is desirable that a client machine of the servers in Figure 3 be able to operate if one or more of the servers is down. A critical aspect of this is having recent copies of the shared files from the "/etc server." As part of the mounting of these files, before the mount is actually performed, the file is copied from the server to the client. For example, before mounting the shared /etc/passwd over the client /etc/passwd, the shared version is mounted temporarily over another file and copied to /etc/passwd. For each user that is to be able to use a machine when the "home directory server" is not available, a home directory must be created and stocked with essential data files. Similarly, for a machine to be able to use an application when the "application server" is not available, that application must be installed in the client's /usr/lpp, when the server's /usr/lpp is not mounted. The resulting machine is certainly not as useful as when the servers are available, but it is usable, and much better than no machine at all.

SUMMARY

We believe we have done well in meeting our design goals:

1. Distributed Services provides local/remote transparency for ordinary files (both data and programs), for directories and for message queues.

2. Our implementation adheres closely to AIX semantics, except for the lack of support for remote mapped files.
3. We have achieved good remote performance in general, and some remote operations are actually faster than corresponding local operations.
4. Use of a popular network protocol, SNA LU 6.2, gives us synergy with other SNA development and independence of the underlying transport media.
5. We have been careful to provide for flexibility in configurations and administrative environments.
6. Our encrypted node identification and id translation mechanisms give us strong control over security and authorization.
7. Our use of architectures such as LU 6.2, the vnode concept, our Virtual Circuit Interface, etc. allows us substantial room for potential extension and growth in network media, file systems and network protocols, respectively.

Further, we believe we have advanced the state of the art with the following

1. Our simple, but effective approach to single system image.
2. Use of a standard virtual circuit protocol, SNA LU 6.2, while achieving high performance.
3. Our performance optimizations, especially our caching strategies.
4. Our extensions for administrative flexibility and control, e.g., file granularity mounts, inherited mounts, administration based on remote mounting of profiles, etc.

REFERENCES

1. IBM, *IBM RT Personal Computer AIX Operating System Technical Reference Manual*, SA23-0806, January 1986.
2. P.J. Kilpatrick and C. Greene, "Restructuring the AIX User Interface," IBM RT Personal Computer Technology, SA23-1057, January 1986.
3. S.R. Kleinman, "Vnodes: An Architecture for Multiple File System Types in Sun UNIX," *USENIX Conference Proceedings*, Atlanta, June 1986.
4. T.G. Lang, M.S. Greenberg and C.H. Sauer, "The Virtual Resource Manager," *IBM RT Personal Computer Technology*, SA23-1057, January 1986.
5. L.K. Loucks, "IBM RT PC AIX Kernel — Modifications and Extensions," *IBM RT Personal Computer Technology*, SA23-1057, January 1986.
6. T. Murphy and R. Verburg, "Extendable High-Level AIX User Interface," *IBM RT Personal Computer Technology*, SA23-1057, January 1986.
7. G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin and G. Thiel, "A Network Transparent, High Reliability Distributed System," *Proceedings of the 8th Symposium on Operating Systems Principles*, Pacific Grove, CA, 1981.
8. G. Popek and B. Walker, *The LOCUS Distributed Operating System*, MIT Press, 1985.
9. A.P. Rifkin, M.P. Forbes, Richard L. Hamilton, M. Sabrio, S. Shah and K. Yueh, "RFS Architectural Overview," *USENIX Conference Proceedings, Atlanta*, June 1986.
10. R. Sandberg, D. Goldberg, S. Kleinman, Dan Walsh and B. Lyon, "Design and Implementation of the Sun Network File System," *USENIX Conference Proceedings*, Portland, June 1985.
11. Sun Microsystems, Inc., *Networking on the Sun Workstation*, Feburary 1986.