EXTENDED ABSTRACT

Scale and Performance in a Distributed File System

John H. Howard
Michael L. Kazar
Sherri G. Menees
David A. Nichols
M. Satyanarayanan
Robert N. Sidebotham
Michael J. West

Information Technology Center
Carnegie Mellon University
Pittsburgh, PA 15213

Andrew is a distributed computing environment being developed in a joint project by Carnegie Mellon University and IBM. One of the major components of Andrew is a distributed file system which constitutes underlying mechanism for sharing information. The goals of the Andrew file system are to support growth up to at least 7000 workstations (one for each student, faculty member, and staff at Carnegie Mellon) while providing users, application programs, and system administrators with the amenities of a shared file system.

A fundamental result of our concern with scale is the design decision to transfer whole files between servers and workstations rather than some smaller unit such as records or blocks, as almost all other distributed file systems do. This paper examines the consequences of this and other design decisions and features that bear on the scalability of Andrew.

Large scale affects a distributed system in two ways: it degrades performance and it complicates administration and day-to-day operation. This paper addresses both concerns and shows that the mechanisms we have incorporated cope with them successfully. We start the initial prototype of the system, what we learned from it, and how we changed the system to improve performance. We compare its performance with that of a block-oriented file system, Sun Microsystems' NFS, in order to evaluate the whole file transfer strategy. We then turn to operability, and finish with issues related peripherally to scale and with the ways the present design could be enhanced.

The Prototype

Using a set of dedicated servers, collectively called *Vice*, the Andrew File System presents a homogeneous, location-transparent file name space to all its client workstations. Clients and servers run the 4.2 Berkeley Software Distribution (4.2BSD) of the Unix operating system[1]. The operating system on each workstation intercepts file system calls and forwards them to a user-level process on that workstation. This process, called *Venus*, caches files from Vice and stores modified copies back on the servers they came from. Venus contacts Vice only when a file is opened or closed; reading and writing individual bytes of a file are performed directly on the cached copy, bypassing Venus. In general, the design performs operations directly in the workstation wherever possible, minimizing interactions with Vice.

[1]Unix is a trademark of AT&T. To avoid any possible ambiguity, we use the name "4.2BSD" throughout for the specific version of Unix used in our system.

The prototype of the Andrew file system was intended to validate the basic architecture and to obtain design feedback as rapidly as possible, while being large and usable enough to make that feedback meaningful. At its peak the prototype had about 400 users sharing 100 workstations and was implemented with six servers.

Almost every application program on workstations was able to use files in Vice without being recompiled or relinked, demonstrating that it is possible to emulate 4.2BSD file system semantics using caching and whole file transfer. Command execution from Vice was noticeably slower than from local files on workstations, but overall performance was so much better than that of the heavily-loaded timesharing systems used by the general user community that our users suffered the Vice delays willingly. We found that performance was usually acceptable up to a limit of about 20 active users per server. However, there were occasions when even a few users could cause performance to degrade intolerably.

Performance Improvements

In addition to subjective reports of file system speed, we measured file system performance with a benchmark operating on a set of files. The operations performed by the benchmark are intended to be a sample of the kinds of actions a user might perform. We used this benchmark to compare different versions of the Andrew file system with each other and with alternatives including a non-distributed system. We also derived curves showing the response of the file systems compared to increasing loads, and extracted file system specific statistics, such as hit and read/write ratios, from workstations in production use.

One of the first surprises we got about the prototype was the relatively high frequency of "stat" operations, which ask about file status. The prototype Venus re-validated cache entries every time the application program opened files or asked for status information. About 62% of the operations were re-validations of existing cache entries. Another 27% where requests for status about files not in the cache. Only 6% of the Venus-Vice calls involved actual file transfers, with approximately two Fetch calls for every Store. This led us eventually to modify the design so that file servers notify Venus when file status information changes (a relatively infrequent event) rather than Venus so frequently re-validating it. Although this *Callback* mechanism violated an earlier design principle that the servers should never initiate operations, it resulted in a very significant reduction in Venus-Vice traffic and the resultant network and CPU loads.

CPU utilizations were high in both in the benchmark and in production measurements. Average CPU utilizations of the two busiest servers were about 40%, with 5-minute peaks around 75%. Profiling revealed that the two factors chiefly responsible for CPU loading were the process switches and pathname resolution. Server loads were often unbalanced, suggesting that some sort of load balancing would be desirable.

In order to deal with the functional complexity and CPU load of symbolic pathname resolution, we moved it to workstations. We introduced a low-level internal file name, called a *Fid* (File Identifier.) A Fid is a fixed-length unique ID for a specific Vice file. Venus and Vice communicate using Fids rather than symbolic names. When Venus fetches a directory from Vice, it gets a mapping from symbolic pathname components to Fids. (This is in principle similar to the way Unix directories map pathname components into inodes, so a Fid resembles an

inumber.) Venus caches these directories very much as it caches files, and resolves pathnames directly without involving the server except to fetch missing directories. This approach depends on the callback mechanism to avoid revalidating all the directories. It eliminated server CPU load for filename resolution with no observable cost in the workstation.

The other heavy consumer of server CPU time in the prototype was process switching. We introduced a lightweight process mechanism to deal with this, and to eliminate the problem of running into arbitrary and sometimes undocumented resource limitations in the kernel.

The overall effect of these changes on performance was dramatic. Where the prototype peaked out at approximately 20 users, the current system handles 50 with capacity to spare. Production measurements show 50 to 70 active users per server during peak hours. (However, it should be kept in mind that a single user may have active connections to more than one server.) At present, all servers are connected to the campus backbone Ethernet which is showing peak utilizations of around 5%; this may well become our next bottleneck.

Comparison With Remote Open File Systems

The comparison with NFS is intended primarily to evaluate the effect of whole file transfer and caching in a distributed file system. The Andrew strategy is not without its drawbacks: it requires a local disk for the file cache, it has trouble with very large files, and it moves the entire file even if only one byte is read or changed. Strict emulation of 4.2BSD concurrent read and write, which permit byte by byte interleaving, is impossible since read and write operations are not intercepted. On the other hand, it greatly reduces the number of interactions with the file servers, it simplifies cache management since only files, not individual pages, must be tracked, and makes it possible to retain cache contents across reboots, a beneficial bonus.

We chose NFS for this comparison because it is a typical and successful example of a distributed file system which open files remotely and reads and writes blocks, and because it runs on the same hardware as Vice does. The comparison addresses only performance given an identical hardware configuration, not such other interesting questions as overall system cost, which is meaningful only if the alternative configurations are individually tuned to the needs of their respective systems.

The comparison's results are summarized in Figure 3 of the paper (reproduced below), showing the time to complete the benchmark for various loads. At very low loads, NFS is faster, but the curves cross at 3 to 4 load units (corresponding to about 15 to 20 Andrew users). At the highest loads measured, NFS was taking nearly twice as long to complete the benchmark as Andrew, and client processes were beginning to fail under NFS due to lost packets. Both CPU and disk utilizations in NFS exceeded 50% beyond 5 load units, while the highest measured values for Andrew were 42% CPU and 28% disk. It is clear that the Andrew File System is far less load sensitive than NFS. Since NFS is a mature and well-tuned system, the most probable cause for this is the difference in their designs.

Operability

Among the features introduced for operability's sake were simple migration of files from one server to another, space quotas for users, replication of seldom-changing system files, and on-the-fly backup. All of these were made much easier by introduction of the notion of a *Volume*, which is a collection of files forming a partial subtree of the Vice name space. Typically, a user is given a single volume to hold his or her personal files; project groups may share a volume, and system administrators set up new volumes to hold new versions of the system or other sets of closely related files.

Volumes are glued together at *Mount Points* to form the complete name space. Mount points are not visible in pathnames; Venus transparently recognizes and crosses them during name resolution, much as the standard 4.2BSD mount mechanism does.

One of the components of a Vice File Identifier (Fid) is a Volume number. Knowing the volume number, you locate the server(s) which have a file by reference to a *Volume Location Database* which is replicated on all the servers using a periodic broadcast from a centralized administrative source. (All administrative data is handled this way, so it is possible a Vice server to continue running even if all the other servers are down.) The server locates the file using the remainder of the Fid to identify a particular file within a volume.

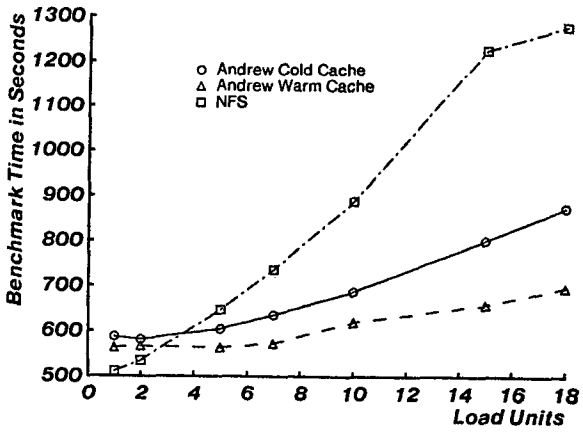
The volume mechanism permits a *Clone* operation, which constructs a read-only snapshot of a volume by duplicating the volume's index (but not the individual files.) Clones are cheap, so they are used for several purposes. They are the basis of replicated system volumes, which help balance server utilizations and increase availability. Migrating a volume to a new server uses a temporary clone, so the volume can continue to be used (and updated) while it is being migrated. Every user volume is cloned daily and the clone inserted into the user's home directory under the name "OldFiles", thus making yesterday's files available in case of an accidental deletion or other blunder. The "OldFiles" clone is also used by the backup system, which can thus make a consistent copy of a volume without taking it offline.

The volume system is also the basis for administrative actions such as adding and removing users, and for disk quotas. We are convinced that something like the volume abstraction is indispensable in a large distributed file system.

Conclusions

At writing (summer of 1987) there are about 400 Andrew workstations shared by about 4000 registered users, of whom some 1000 are regular users. Approximately one fifth of the workstations are in public clusters. There are 16 servers storing approximately 6 gigabytes of data. Present performance is generally satisfactory although there are instances of noticeable but tolerable sluggishness. We feel confident that we can nearly double the number of workstations with the current design.

There are many other areas which will need attention in the future. Moving Venus and the server code into the kernel would improve performance significantly. It would also be desirable to convert the kernel intercept mechanism to an industry standard. Network topology and clustering may be needed to reduce backbone loading. Some form of replication of writable files will eventually be necessary. Monitoring, fault isolation, and diagnostic tools that span all levels of the hardware and software will also become increasingly important. Finally, decentralized administration and physical dispersal of servers will will be necessary in a very large system.



This figure compares the benchmark times of NFS and the Andrew file system as a function of load. The clients were Sun3/50s with 4 Mbytes of real memory and a 70 Mbyte local disk. The server was a Sun3/160 with 8Mbytes of real memory and two 450 Mbyte disks. In the NFS experiments, at loads of 10 or more, some of the clients failed to complete the final phase of the benchmark. Refer to the paper for more details.

Figure 3: NFS and Andrew Benchmark Times