Low Latency File Access in a High Bandwidth Environment

Stephen Pink, Swedish Institute of Computer Science
Anders Klemets, Royal Institute of Technology
steve@sics.se, klemets@it.kth.se

Abstract

Workstations will soon have hundreds of megabytes of main memory and will be attached to high speed networks. Many distributed applications will have to be redesigned to take advantage of these new features. Today's distributed file systems are designed for workstations with relatively small amounts of memory that are attached to medium-speed networks (e.g., Ethernets). File access latency will not decrease when today's distributed file systems are run on the newer platforms. The JetFile distributed file system is designed to take advantage of workstations and servers with large memories attached to networks where user-to-user bandwidth approaches one gigabit per second. The design of JetFile allows the high bandwidth of the network to provide low latency file access. This is accomplished by caching large data objects, i.e., directory subtrees, on reference to limit client/server communication. In addition, JetFile uses multicast extensively in its client cache consistency mechanism to help avoid costly network round trips between client and server. Reliable multicast is also used to support server replication of files in JetFile.

1 Introduction

In this position paper we briefly describe JetFile, a distributed file system designed to take advantage of computers with large memories attached to high speed networks in order to provide low latency file access. Current distributed file systems were designed for workstations with relatively small amounts of memory, i.e., less than 64 Megabytes in the normal case, and that are attached to "medium speed" networks such as Ethernets. Because of the caching strategy of these file systems, it is likely that users of future workstations will find current distributed file systems to be a bottleneck. JetFile is designed to cache very large data objects in main memory and to minimize the number of round trips on the network necessary to keep the clients' cached data consistent with the servers' store. Thus, JetFile takes advantage of the ability of high speed networks, i.e, networks of 100

Megabits per second or more, to transfer large amounts of data quickly, as well as making use of the large amount of main memory that will be available in future workstations.

2 Caching in JetFile

A user is likely to use many files in his home directory tree, even in one session. Searching through mail files and compiling programs with multiple modules are just two examples of such behavior. It makes sense, then, as a first approximation to a working set, to cache the files in one's home directory tree in main memory, if there is room. Since the speed of the network is increasing faster than the speed of the disk, the fastest path between file system client and server is often between their memories. It is likely, in the future, that servers will have enough main memory to hold the home directory trees of their clients' users, and clients will have main memories large enough to hold the home directory trees of their users. Thus, the general caching strategy of JetFile is to transfer whole directory trees on reference from servers to clients. On this strategy, most users will have their home directory trees cached at the beginning of their work session.

There will, of course, be some important exceptions to this rule of caching directory trees. For example, file system root directories will not be cached and symbolic links to directory trees will not normally be cached. Also, it makes no sense to cache files that are abnormally large in size, such as files that contain large databases as well as files which contain pre-recorded digital audio or video, especially since these files do not exhibit much locality of reference. In the latter case, for example, a video file of a film is usually played back linearly so that the early parts of the file are never accessed again once they are played back. These exceptions mean that JetFile will have file types such that the sorts of files mentioned above will be marked uncacheable. It will be part of our experiment to determine exactly how to determine these uncacheable types of files.

2.1 High bandwidth and low latency

High speed networks transfer large amounts of data in a short period of time. They do not very much, in comparison, decrease the latency of short request/response communication such as occur in remote procedure calls. To use an illustration, a gigabit per second network and a slow-speed serial link are just as slow/fast as each other when it comes to the first bit of a message. It is the last bit of the message that arrives faster in the case of the gigabit network. Thus, distributed applications which make efficient use of high speed networks should avoid costly round trips of small messages and anticipate future use by transferring large data objects. And since memory is becoming cheaper, it is reasonable to store a large working set in the client's main memory.

A simple TCP bulk data file transfer, in JetFile, will be the only network activity necessary, in many cases, for a user to read his files, since files in home directories are almost exclusively written by their owners. No remote procedure calls are used to transfer files, as in NFS[SGK+85]. Also, JetFile uses a stateful server and multicast updates to ensure the consistency of files among multiple users. Our multicast update scheme avoids using costly remote procedure calls.

Upon login, a user's home directory is transferred from the server to the client's main memory cache. If the user's process changes directory, the contents of the new directory are transferred to the client's cache. Memory is made available for the new objects in the cache by swapping old files out to the swap area based on an LRU algorithm. If another user writes to one of the first user's cached files (not very usual for a user's home directory), the server detects this and multicasts the updated file to all of its clients. If two users open a file for writing, client caching is turned off, as in Sprite[NWO88].

2.2 Virtual memory and the file system

We buy memory for our workstations primarily to run large programs. We don't always want to use all of our memory to store cached files. JetFile shares memory dynamically with the the operating system's virtual memory as in the Sprite operating system. When a user's program needs memory, the cached files are swapped to backing store (JetFile uses a single-level store) according to an LRU algorithm, allowing dynamic sharing of main memory between the file system and virtual memory.

Dynamic memory sharing between JetFile and the virtual memory system means that there is no waste of memory even if few of the files in a directory are ever used after they are transferred. And since the the workstation is attached to a high speed network, the latency incurred by transferring more files than are ever used is low. The point is that large memories and high bandwidth networks make reasonable our strategy of anticipation in order to avoid future latencies.

2.3 Large client and server main memories

It is likely that file servers of the future will have more physical memory than their clients, since that is the case today. It is important for an NFS server, for example, to have as much of the client's file in its main memory as possible as this greatly speeds up reads from the server, and the server's disk is a bottleneck even today in NFS. We assume that a JetFile server will have much more memory than any of its clients. If a client workstation has a main memory size of 300 megabytes (here one assumes that a reasonable home directory size for a user is 300 megabytes), a server should have three gigabytes of memory to serve 10 clients. If the server should have to read the client's home directory tree from disk, it would add significantly to the latency suffered by the client when first caching his home directory tree.

3 Consistency

Most distributed file systems offer some consistency guarantees to their clients. NFS puts the onus of this guarantee solely on the client which must perform periodic remote procedure calls to validate its cached file blocks. As we have said, frequent remote procedure calls of small messages does not make the best use of high speed networks. As CPU cycles increase and networks grow longer, the delay caused by the propagation time alone for a remote procedure call will become intolerable as more and more instruction cycles are wasted waiting for the remote procedure call to return.

AFS[MSC+86], on the other hand, has a stateful server that eliminates the need for client RPCs to guarantee consistency. AFS's server detects inconsistencies between servers and clients and issues a callback to its clients to flush their caches and re-transfer the new version down to their local disk. We think that this is an improvement on the heavily used RPC mechanism in NFS. But we also believe that this callback RPC from the server could be eliminated as well by having the clients "snoop" on multicast addresses that are being used by other clients that are updating the server with a reliable multicast transfer protocol. When clients write to the server in JetFile, they use reliable multicast. At the same time, other clients "snoop" on the multicast address, updating their caches without always needing to contact the servers.

3.1 Snooping for consistency

Many of the new gigabit networks are designed with support for multicast[BLRS94, BCF+92, WBC+94]. Multicast allows the clients of a server to form a group, where sending to one member means sending to them all. Not only does this lessen the load on the network, it lessens the load on the server which would otherwise be burdened with serving up the same file many times. In order to provide this multicast update service, we are developing a reliable multicast transport protocol, for use over multicast IP, that minimizes the latency of file transfers to groups of receivers[HW]. This reliable multicast protocol provides much of the support for server replication as described below.

It would be costly from the point of view of performance, however, to depend on reliable multicast between clients to provide cache consistency. A client should not have to keep track of the caches of all its peers. One feature of many high speed networks is reliability; very little data is actually lost on such networks. Thus, we propose that, other clients "snoop" on a group multicast address for a file or set of files, using unreliable multicast, gathering blocks of the updated file, while the file is being reliably multicast to the servers from writing clients. Since the network promises to be fairly reliable, in most cases clients will successfully snoop the whole file correctly.

Of course there will always be errors on the network, and sometimes clients will miss all or parts of the updated file. In case a client has successfully snooped some blocks of the file but not others, negative acknowledgments can be sent to the server and the missing blocks retransmitted to the clients that need them. In case a client has missed all the blocks of a file update, so that the client is not capable even of sending these negative acknowledgments, the servers will periodically publish, via reliable multicasts, lists of versions and lengths of files that have recently been updated from other clients. These reliable multicast advertisements can then be used by clients to request missing files from the servers. The number of reliable multicasts that have to be made in this way will be a key factor in the overall scaling and performance of JetFile.

Thus, we use both reliable and unreliable multicast in JetFile to provide consistency of client caches with a minimum of delay. What is crucial to achieving low delay is limiting the number of round trip times between client and server. In those cases where the network does not corrupt or reorder data and when hosts are functioning normally, clients can update their caches by snooping on a multicast group address without any further requests from the server. This implies that the network interface to each client and server be capable of supporting numerous multicast addresses. This hardware support for multicast filtering is essential to prevent host operating system kernels from being interrupted needlessly for updates to files that are not cached in their memory. A high speed network that we plan to use for JetFile supports thousands of multicast addresses in its host interface, thus providing the possibility to map identifiers of cached files into multicast addresses. In contrast, many current Ethernet implementations support only 16 multicast addresses in hardware.

3.2 Replication

Another use of multicast in JetFile is to help achieve server replication. Most of today's distributed file systems limit the number of servers per file to one. This is unfortunate since it limits file availability and doesn't provide backup for the user of a file in case of server failure. Current versions of files can be made available efficiently on multiple servers if clients update their servers via multicast. Using multicast, as in Coda[KS91], to achieve server replication is easy on the network and easy on the servers as well. Most of the burden of replication is on the client in this scheme not on the server, which is a busy shared resource.[1] Thus, when a client writes a file, the file is multicast to a group of servers. The reliable multicast protocol used in updating clients, mentioned above, will also be used to provide file server replication.

4 Conclusion

JetFile is a new distributed file system designed to take advantage of workstations of the future that will have large memories and will be attached to high speed networks.

[1]There must, of course, be a transfer protocol that operates between servers to make them consistent in case one or more file servers are down when the client performs a write.

JetFile caches directories trees in clients' main memories to avoid the latency caused by by multiple remote procedure calls across the network. Multicast network support in the form of "snooping" is used in JetFile to provide efficient consistency guarantees among clients and servers. Reliable multicast file transfer is used to help provide server replication. A prototype of JetFile is planned for HP workstations and the experimental HP JetStream network. [2]

References

[BCF+92] Ernst Biersack, Charles Cotton, David C. Feldmeier, Anthony J. McAuley, and David W. Sincoskie. Gigabit Networking Research at Bellcore. *IEEE Network*, 6(3), March 1992.

[BLRS94] Christer Bohm, Per Lindgren, Lars Ramfelt, and Peter Sjödin. The DTM Gigabit Network. *Journal of High Speed Networks*, 3(2), 1994.

[HW] Olof Hagsand and Assar Westerlund. Sid: A Reliable Multicast Protocol for High Speed Networks. Forthcoming.

[KS91] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 213–25. Association for Computing Machinery SIGOPS, October 1991.

[MSC+86] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal, and F. Donelson Smith. Andrew: A Distributed Personal Computing Environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[NWO88] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions of Computer Systems*, 6(1), February 1988.

[SGK+85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and Inplementation of the Sun Network File System. In *USENIX Conference proceedings*, pages 119–130, June 1985.

[WBC+94] Greg Watson, David Banks, Costas Calamvokis, Chris Dalton, Aled Edwards, and John Lumley. AAL5 at a Gigabit for a Kilobuck. *Journal of High Speed Networks*, 3(2), 1994.

[2] The authors wish to thank M. Satyanarayanan, P. Kumar, D. Johnson, and the other members of the seminar at Carnegie Mellon University where an earlier version of this design was presented. We benefited greatly from their careful criticism, but we take sole responsibility for the views expressed in this paper.