Multi-level Caching in Distributed File Systems
— or —
Your cache ain't nuthin' but trash

D. Muntz and P. Honeyman

Center for Information Technology Integration
The University of Michigan
Ann Arbor

Abstract

We are investigating the potential for a hierarchy of intermediate file servers to address scaling problems in increasingly large distributed file systems. To this end, we have run trace-driven simulations based on data from DEC-SRC and our own data collection to determine the potential of caching-only intermediate servers.

The degree of sharing among clients is central to the effectiveness of an intermediate server. This turns out to be quite low in the traces available to us. All told, fewer than 10% of block accesses are to files shared by more than one file system client.

Our simulations show that even with an infinite cache at an intermediate server, cache hit rates are disappointingly low. For client caches as small as 20M, we observe hit rates under 19%. As client cache sizes increase, the hit rate at an intermediate server approaches the degree of sharing among all clients. On the other hand, the intermediate server does appear to be effective in boosting the performance and scalability of upstream file servers by substantially reducing the request rate presented to them.

1. Introduction

As distributed file systems grow, so does the need to increase scalability. At the Institutional File System Project, we are investigating tools and techniques for offering file service to a huge client base, perhaps as many as 30,000 end systems. We elected to deploy AFS [1] as the principal distributed file system protocol, because it has proven to scale well to environments with large numbers of users and files [2]. AFS clients cache copies of recently used files on their local disks. This allows most file system access requests to be serviced by the local cache manager, without any mediation by file servers.

To reach the broad base of users at our campus, we need to service clients supporting a variety of file system protocols, *e.g.*, AFS, NFS [3], and AFP [4], among others. Our principal file servers all run AFS, so the first of these is not a problem. For other file system protocols, we have built intermediate servers that act as AFS clients of the principal file servers and as NFS or AFP servers for clients requiring foreign protocols.

We also considered the case where the intermediate server uses AFS for both its client and server roles. This architecture extends to one in which there are multiple levels of intermediate AFS (or *iAFS*) servers, each caching files it fetches from the upstream servers, and serving files out of its cache to downstream clients.

One reason for considering multi-level cache hierarchies is that they have shown great success in improving CPU performance when used in processor memories [5]. In the context of file systems, caching-only intermediate servers potentially reduce the load presented to the principal file servers by satisfying client requests directly. Furthermore, iAFS servers offer the potential to concentrate state information† that might otherwise overload the principal servers. Resources thus freed can then be used to serve a larger client base.

† Namely, AFS connections and callbacks.

The goal of this study is to assess the degree to which iAFS servers can increase the performance and scalability of large-scale distributed file systems. Our principal tool is a trace-driven simulator that analyzes file system trace data taken from ‘‘real-world’’ networks.

2. Trace-driven simulation

To explore the potential of multi-level caching in distributed file systems, we ran trace-driven simulations to predict the hit rates that we might see at an iAFS server. The subject of these simulations is data caching. (Directory caching may be studied in future work.) The traces fed to the simulator were derived both from data collected in a network of Firefly workstations [6] at the Digital Equipment Corporation’s Systems Research Center, and from file server trace data collected here at CITI.

2.1. Firefly trace data

The Firefly data was collected over a four day period in February, 1990 from 115 Firefly workstations supporting the Topaz environment, which includes a (proprietary) distributed file system protocol. During the trace period, each client produced a log record for every system call related to file system operations. Each record contained the following information:

- the name of the system call
- the process id of the invoking process
- the arguments to the call
- the time at which the call was entered
- the time at which the call was exited
- the success or failure status of the call

We preprocessed the data to convert file descriptors into path names, to eliminate irrelevant log records, and to normalize the name space.

The cache simulator needs pathnames for its hit rate accounting. However, some system calls, *e.g.*, `read`, use a file descriptor instead of a pathname. To convert fd’s to pathnames, we implemented a process simulator that builds a table for each process which associates the pathname used in, say, `open` calls with the file descriptor returned. This table is copied across `fork` and `exec` calls. Relative pathnames, such as those starting with ‘‘`.`’’ and ‘‘`..`’’, were also converted to the appropriate pathnames at this stage.

In this study, accesses to the local file system were not of interest and were eliminated in preprocessing. In addition, system calls that failed were elided. Failures can arise, *e.g.*, when attempting to create a file in a write-protected directory.

The name space was normalized by converting names of the form `host:path` to a flat name space of unique integers. In all, 68,413 different pathnames are referenced in 2,807,003 trace records.

2.2. IFS trace data

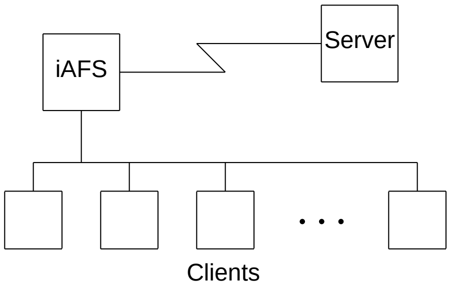
The IFS trace data was collected from four AFS servers running on IBM RT computers during a 4.8 day period in early November, 1990. The data records all file server requests from 49 clients. 31,538 different files are referenced in 92,571 trace records.

Data collected at the IFS Project was tailored more directly to our needs and required less preprocessing. AFS clients exchange file data with the server via `FETCHDATA` and `STOREDATA` requests, whose functions follow directly from their names. Each `FETCHDATA` and `STOREDATA` request contains a timestamp, the client’s network address, the file’s `FID` (the unique identifier for a file) and the offset and length of the data being requested.

In the IFS traces the return status of the requests is not recorded. We have observed that almost all fetch and store requests succeed, so we don’t believe that this limitation invalidates the results reported by the simulator.

3. The simulator

We performed experiments simulating distributed environments with a two-level cache design, using file system activity traces provided by DEC-SRC as well as traces collected locally. In the simulated environments of the experiments, the client machines are connected to an intermediate server which is in turn connected to a principal file server.



Topology of the simulated environment

For the experiments discussed in this paper, the intermediate server has a potentially infinite cache. This is obviously impractical. Because an iAFS server with a finite cache would be forced to flush its contents on occasion, the hit rates reported here are larger than can be achieved in reality.

The operation of the simulator is straightforward. When a system call requesting a file from the server machine appears in the trace data, the simulator checks the local cache on the requesting machine to see if the request can be satisfied there. If the requested block is found in the local cache, a ‘‘hit’’ is logged for that client and the next trace record is processed. Otherwise, a ‘‘miss’’ is recorded for the client, and the cache on the iAFS server is checked for the requested block.

If the block is found in the iAFS server’s cache, a hit is recorded for the iAFS server, and the block is placed in the client’s cache. Otherwise, a miss is recorded for the iAFS server, the block is installed in both the iAFS server’s cache and the client’s cache, and the next trace record is processed.

In this way, the input of trace records is processed until exhausted. All read and write requests are guaranteed to succeed at the server, and the cache replacement policy is LRU. When a file is written by a client, the simulator invalidates that file in the cache of any other client holding a copy. Write operations are counted as cache misses on the iAFS server.

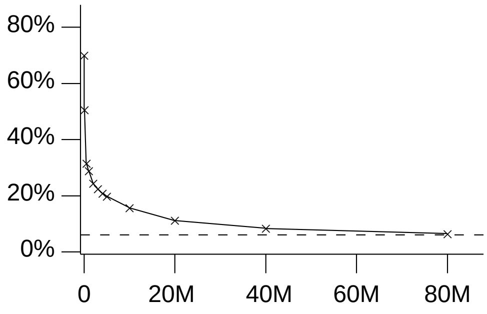
4. Hit rate simulations

In the first set of experiments, we examine the hit rates that can be expected for the iAFS server cache. We first simulate an iAFS server with an unbounded cache using the trace data from the Firefly clients. We then restrict our attention to 20 Firefly clients that appear to exhibit a high degree of data sharing — these 20 clients are responsible for over half of the iAFS server cache hits. We then use the trace data collected from the 49 IFS clients. Again we simulate an iAFS server with an unbounded cache.

We used 64K as our cache block size, because this is the size used by AFS. Simulations were also run using block sizes of 4K, 8K, 16K, and 32K; those results are not substantially different from the ones presented here.

4.1. Firefly clients

The first experiment with the Firefly data simulates an environment in which all 115 machines are clients to an iAFS server. The iAFS server is given an ‘‘infinite’’ cache size, so that if a given block is ever sought twice, each request after the first causes a hit at the iAFS server. In practice, the iAFS server cache would have to be 7,880M to achieve this hit rate. The size of the client caches is varied in each simulation to generate a graph of client cache size *vs*. iAFS server cache hit rate.



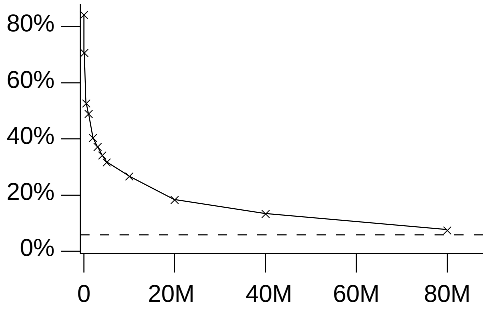
Client cache size *vs*. iAFS hit rate

Recent studies utilizing the Firefly data have shown that client caching is desirable [7]: when client caching is disabled, the iAFS has a 70% hit rate, which falls off rapidly as clients are able to resolve requests from a local cache. With just 1M of client cache, the iAFS hit rate falls below 30%. In our (typical AFS) environment, clients have 20M caches containing roughly 1,500 cached files when hot. For such an environment the simulation predicts an 11% hit rate at the iAFS server. Client cache sizes can be expected to grow as disk density continues to increase. The simulation predicts a corresponding decrease in iAFS server hit rates: an 80M client cache produces a 7% iAFS hit rate.

As the client cache size approaches infinity, the hit rate at the iAFS server asymptotically approaches the degree of sharing, *i.e.*, the fraction of files that are accessed by more than one client system. This asymptote is represented in the graphs by a dashed line. The degree of sharing seen among the Firefly clients is 6.1%.

4.2. Partial Firefly clients

Among the Firefly clients, there is a subset whose file reference patterns are more tightly woven: 20 clients are responsible for over half of the overlap in file references among all 115 Fireflies. We simulated an iAFS server for these 20 clients, as in the previous section. The degree of sharing among them is 5.8%.



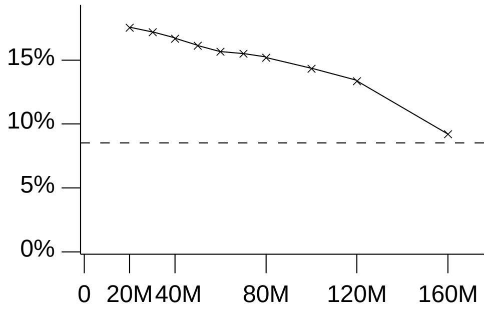
Client cache size *vs*. iAFS hit rate

Again, we see that even meager client cache sizes result in low iAFS hit rates. In this experiment when the client caches are 20M, the iAFS hit rate is about 18%. When client caches are 80M, the iAFS hit rate drops to about 7.6%

4.3. IFS clients

In the third experiment, data collected on IFS project file servers was used to drive the simulations. The four servers on which data was collected contain all home directories, system binaries, and project-related data and programs for the several dozen IFS project staff.

This data reflects FETCHDATA and STOREDATA requests from 49 AFS clients. Again, the simulation involves all 49 clients connected to one iAFS with infinite cache. Clients in the IFS project have 20M caches on their local disks, so read requests satisfied by the local cache are invisible to the server. Because the trace data was collected on the server, simulation is possible only for client caches of at least 20M. The degree of sharing among the 49 IFS project clients is 8.5%.

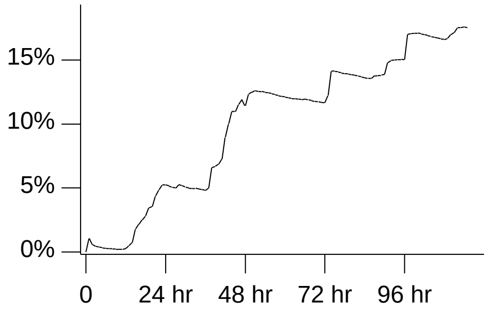


Client cache size *vs*. iAFS hit rate

For typical AFS client cache sizes, 20–80M, the simulation predicts the iAFS hit rate to be 15%–18%.

5. Hot and cold cache experiments

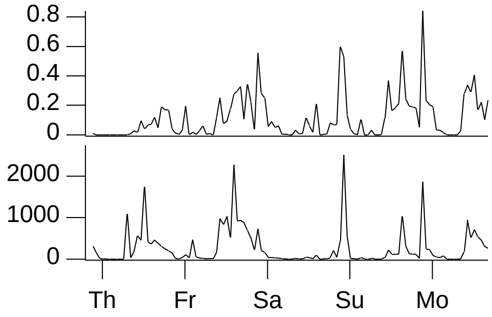
In this section, we focus on the IFS trace data with 20M client caches, reflecting the environment in which the data was collected. In the simulations described so far, the iAFS server cache is initially empty, or ‘‘cold.’’ Consequently, in the early hours of the simulation there are very few cache hits. Examining the iAFS server hit rate over time, the aggregate hit rate increases through the simulated 116 hour duration, and is apparently still rising at the end of the simulated period.



Time *vs*. iAFS server cache hit rate

As the iAFS server cache ‘‘warms up,’’ the hit rate becomes more respectable. Clearly there must be times when the hit rate is higher than the final 18%. Further data collection covering a larger period of time will give a better indication of the shape of this curve.

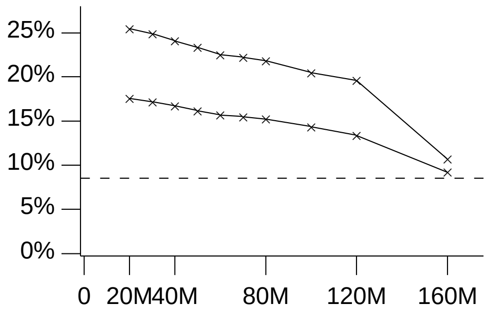
In the next experiment, we collect hourly hit rate statistics and plot them as the ‘‘instantaneous’’ hit rate at the iAFS server cache. We show this in the next graph, along with the hourly request rate presented to the iAFS server.



Upper graph: instantaneous hit rate
Lower graph: request rate

The upper graph shows a highly variable instantaneous hit rate with several distinct peaks. The lower graph shows a similar pattern in request rates. Note that after a ''warm-up'' interval, the peaks in the request rate coincide with peaks in the hit rate. This is due to bursts of activity on individual workstations that would be serviced locally if client caches were larger.

To gauge the effect of a ''hot'' cache in the iAFS server, we re-ran the simulation of the previous section on a hot-cache iAFS server. We treat the first half of the simulation period, 58 hours, as the warm-up interval and gather statistics for various client cache sizes in the last half of the period. The following graph shows the results of this experiment, superimposed with the graph from the preceding section, where warm-up is not taken into consideration.



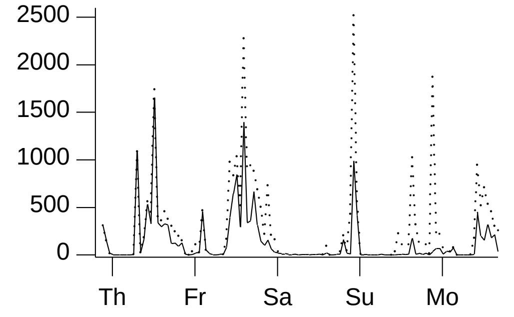
Client cache size *vs*. iAFS hit rate

The upper graph shows an improvement in the iAFS server cache hit rate when it is pre-heated in the first 58 hours of the trace interval. Even here, though, the iAFS hit rate is less than 30% for standard 20M client caches.

Another set of experiments involves clearing the client caches periodically while maintaining accounting throughout the simulation. This models an environment in which a machine is used sequentially by different people. The results are more pessimistic, however, as there would likely be some overlap among the users' data requests *e.g.*, `/bin/csh`. Results from these tests are similar to the other warm cache experiments: simulation predicts an improved hit rate at the iAFS server, but the improvement is not dramatic.

6. Effect on upstream server load

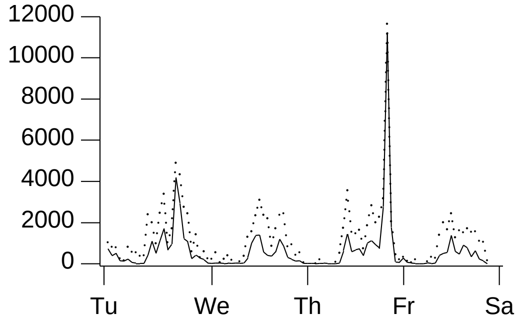
In the previous section, we saw that high request rates appear to coincide with high instantaneous hit rates on the iAFS server. This suggests that the iAFS server may be effective in moderating the peak traffic presented to the upstream server(s). To test this hypothesis, we ran a set of simulation experiments with the IFS data to measure the read request rate seen by upstream servers when an iAFS server is present and when it is absent. Client writes are always presented to the upstream servers, whether or not an iAFS server is employed. Since the iAFS server can have no effect on upstream server performance for writes, we ignore them in the next set of experiments and concentrate on read operations alone.



Request rate seen by iAFS server (dotted)
and by upstream server (solid)

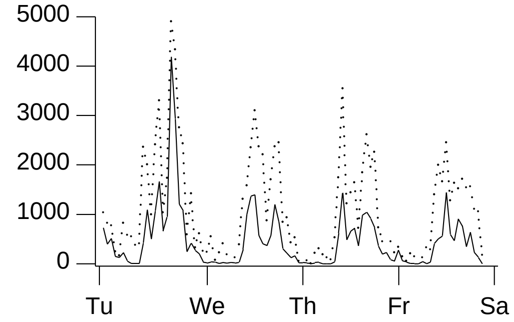
The graph shows the request rate, in requests per hour, presented to upstream server(s) when the iAFS server is present (solid line) and when it is absent (dotted line). After an interval during which the iAFS server cache warms up, the effect is striking: the peak load is reduced from over 2,500 requests per hour to fewer than 1,400 requests per hour.

The Firefly data also shows a correlation between request rates and iAFS server cache hit rates. Our simulations predict that with those file system traces as well, the iAFS is effective in clipping the peak load presented to the upstream server(s).



Request rate seen by iAFS server (dotted)
and by upstream server (solid)

The spike occurring late Thursday night is caused by a system task running on a single machine. This task maintains a database of cross-references between pieces of software at DEC-SRC, *e.g.*, which components use which other components. This task accesses a significant portion of the file system. Eliminating this process' activity from the traces, which accounts for about 3% of the Firefly trace data, makes it easier to see the beneficial effect of the iAFS on the upstream server.



Request rate seen by iAFS server (dotted)
and by upstream server (solid)

As with the IFS traces, the simulation indicates that an iAFS server would substantially lower the peak request rates at the principal file servers.

7. Discussion

The simulated hit rates on the iAFS server do not lend much encouragement for its role in enhancing client performance. Our simulations indicate that most of the requests presented to an iAFS server must be forwarded to an upstream server to be satisfied; from a client's perspective, the iAFS can be viewed as a ''delay server.''

Simulations using the Firefly data show that an iAFS server cache suffers hit rates below 19% when client caches are 20M or more. Simulations using the IFS trace data also predict iAFS server cache hit rates below 18%. This is largely due to a low degree of sharing among clients, less than 9% in each set of trace data.

We also simulated several ''warm cache'' scenarios, in which hit and miss accounting is delayed during a warm-up period. These warm cache simulations predict some improvement, but not much, for the iAFS hit rate.

Our simulations indicate that an iAFS server does help server performance, by clipping the peak request load presented by file system clients. We plan further experiments to investigate this and other ways to exploit multi-level caching in distributed file systems.

8. Future work

Simulation using realistic intermediate server cache sizes is needed to study the potential positive results in server load reduction. Improved data collection at the IFS Project should allow more complete results. We would like future data to span a larger interval (perhaps 2–4 weeks), include error return codes, and track file system accesses from AFS clients that hit the client cache. Directory caching is also of interest.

Acknowledgements

The Firefly traces were gathered by Andy Hisgen, who kindly made them available to us. Susan Owicki, B. Kumar, Jim Gettys, and Deborah Hwang contributed to the file system tracing facility.

We thank Bill Tetzlaff of IBM Research for suggesting some interesting experiments.

We thank Edna Brenner for her careful reading of this manuscript and for her many suggestions that led to improvement.

This work was partially supported by IBM.

References

1. J.H. Howard, ''An Overview of the Andrew File System,'' pp. 23–26 in *Winter 1988 USENIX Conference Proceedings*, Dallas (February, 1988).
2. J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M. West, ''Scale and Performance in Distributed File Systems,'' *ACM TOCS* **6**(1), pp. 51–81 (February, 1988).
3. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, ''Design and Implementation of the Sun Network Filesystem,'' pp. 119–130 in *Summer 1985 USENIX Conference Proceedings*, Portland (June, 1985).
4. G.S. Sidhu, R.F. Andrews, and A.B. Oppenheimer, *Inside AppleTalk*, Addison-Wesley, Reading (1989).
5. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., Palo Alto (1990).
6. Charles P. Thacker, Lawrence C. Stewart, and Edwin H. Satterthwaite, Jr., ''Firefly: A Multiprocessor Workstation,'' *IEEE Transactions on Computers* **37**(8), pp. 909–920 (August, 1988).
7. Matt Blaze and Rafael Alonso, ''Long-Term Caching Strategies for Very Large Distributed File Systems,'' pp. 3–15 in *Summer 1991 USENIX Conference Proceedings*, Nashville (June, 1991).

About the authors

Dan Muntz is a Ph.D. precandidate in Electrical Engineering and Computer Science at the University of Michigan. He received the B.S (with honors) and M.S. from Michigan in 1989 and 1991, respectively. His research interests include very large distributed systems while his personal interests revolve around very small distributed systems and things that make you go hmmmm. Send him mail at `dmuntz@citi.umich.edu`.

After completing undergraduate studies at the University of Michigan, **Peter Honeyman** was awarded the Ph.D. by Princeton University for research in relational database theory. He has been a Member of Technical Staff at Bell Labs and Assistant Professor of Computer Science at Princeton University. He is currently Associate Research Scientist at the University of Michigan's Center for Information Technology Integration. Honeyman has been instrumental in several significant software projects, including Honey DanBer UUCP, Pathalias, MacNFS, and the Telebit UUCP spoof. His current research efforts are focused on distributed file systems, with an emphasis on mobile computing, security, and performance. He can be contacted at `honey@citi.umich.edu`.