The Effect of Client Caching on File Server Workloads

Kevin W. Froese

Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

Richard B. Bunt

Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

Abstract

A distributed file system provides file service from one or more shared file servers to a community of client workstations over a network. While the client-server paradigm has many advantages, it also presents new challenges to system designers concerning performance and reliability. As both client workstations and file servers become increasingly well-resourced, a number of system design decisions need to be re-examined. This research concerns the caching of disk blocks in a distributed client-server environment. Some recent research has suggested that various strategies for cache management may not be equally suited to the circumstances at both the client and the server. Since any caching strategy is based on assumptions concerning the characteristics of the demand, the performance of the strategy is only as good as the accuracy of this assumption. The performance of a caching strategy at a file server is strongly influenced by the presence of client caches since these caches alter the characteristics of the stream of requests that reaches the server. This paper presents the results of an investigation of the effect of client caching on the nature of the server workload as a step towards understanding the performance of caching strategies at the server. The results demonstrate that client caches alter workload characteristics in a way that will have a profound impact on server cache performance, and suggest worthwhile directions for future development of server caching strategies.

1 Introduction

In the early days of client-server environments, the typical workstation client was a diskless limited-memory machine with little or no capacity to store files. By pooling resources into a shared server it became possible to provide remote file service to a community of such clients, effectively and at low cost [1]. In addition, the server could be administered and managed centrally so that services such as back-ups could be provided for all users. As time has passed, workstations have become increasingly well-resourced, and the needs of the users and the demands placed on both the network and the file servers have changed as a result. Main memories on client workstations are now large enough to accomodate large disk block caches with low miss ratios, and continued technology advances should make it possible to achieve even lower miss ratios in the future. The presence of caches at both the client and the server, however, has created a two-level cache hierarchy, the characteristics of which may not be fully understood. In fact, the very presence of client caches significantly alters the nature of demand at the file server, with potentially profound performance consequences. The long term goal of this research is to investigate quantitatively the impact of various configuration parameters, including the amount of file cache at clients and servers, the cache management policies at each location, and the interaction semantics between the two levels of cache. This paper deals specifically with the effect of client caching.

On the surface it would seem that issues affecting caching performance should be well understood since caching has been with us for a long time. In a distributed file system with disk caches at both clients and servers, however, caching has its own unique characteristics. Makaroff and Eager [2] studied the performance of a two-level caching system and observed that although increasing the client cache size decreases the client cache miss ratio, it may increase the miss ratio at the server cache. The overall miss ratio (across both caches) may be unaffected or even increase! Willick, Eager and Bunt [3] investigated the performance of caching strategies in a client-server system, focussing on the suitability of various cache management strategies at the file server. They observed that locality-based polices such as LRU (least-recently-used) perform poorly at the server. Two factors would seem to contribute to these unusual observations: first, the presence of multiple clients means that the stream of requests presented to the server is an aggregation that comes from the interleaving of the streams from individual clients, and, second, the presence of client caches means that these individual request streams have many of the normal application references "filtered out". Individually and in combination these two effects result in a stream of requests for service that can no longer be expected to exhibit the properties needed for traditional caching strategies to perform effectively. The effect is a change in workload characteristics, and it is precisely these characteristics that should dictate the choice of replacement policy.

Since the performance of any system is a function of the workload presented to it, the focus of this paper is on the characterization of the file server workload. The goal of the project was to investigate experimentally the effect of these two factors (aggregation and filtering) on the workload presented to a file server, first to provide a quantitative explanation for this unusual performance and, second, to provide a basis for the design of more effective server caching strategies. The methods used to characterize the workloads were introduced in [3].

The rest of the paper is organized as follows. Section 2 provides a brief overview of the role of caching in distributed file systems and techniques to manage caches at the server and at clients. Section 3 considers the nature of file system requests and describes the particular traces used in these experiments. The experiments themselves are described in Section 4, first the experiments with a single client and then the experiments with multiple clients. Section 5 summarizes the results obtained and presents some conclusions from the work.

2 Caching in distributed file systems

Caching is employed to overcome the performance impact of speed discrepancies between different components of a computing system. Disk caches are used, for example, to address a difference of several orders of magnitude in access time between memories and disk devices. In a traditional centralized system there is typically one cache interposed between the application and the file system's disk (to simplify the discussion it is assumed that caches are always located in memory). Reads to blocks already in the cache (read hits) are resolved without requiring any disk activity, but if a reference is made to a block that is no longer in the cache (a cache miss), the block is fetched into the cache in anticipation of future requests for it. Replacement policies, such as LRU, implement models of block preference that try to predict future reference patterns to determine which of the cached blocks should be removed to make room for an incoming block resulting from a cache miss. LRU assumes that past activity is a good predictor of future activity and makes good decisions when the reference patterns of the application exhibit good temporal locality. Should a cached block be changed by an application (a write hit), the results of this change must always find their way back to the disk to maintain data consistency, and so some disk activity is required whether or not the requested block is in the cache. This presents further cache management challenges, some of which are explored in [4].

A distributed client-server environment introduces additional factors that also bear on performance. First, a network is interposed between the client workstation and the file server, thereby introducing yet another opportunity for delay. A second, and potentially more serious, complication is that caches may exist on both the client and the server. Although they have slightly different roles to play, the two caches work together, but rather than cooperating, they may interfere with each other. The resolution of a read request in the client's cache (a read hit at the client) eliminates the need for a network access to the file server, but a read miss at the client cache must be sent out over the network to be processed by the server. If it is possible to resolve this request in the server's cache (a server cache hit) then the need for a disk operation is eliminated, but if the request misses at the server's cache then it is necessary to go to the disk to retrieve the data requested. The data is then sent back over the network to the client and deposited in both the client's memory and the client's cache. Write operations are even further complicated by multi-level caching since the effects of changes must be reflected everywhere.

A number of distributed file systems provide for two levels of caching. Examples include Sun NFS [5], Echo [6], Sprite [7], Andrew [8], and CODA [9]. Typically the different levels are managed identically, using familiar tried-and-true cache management strategies such as LRU. While this is a common choice, Figures 1 and 2 suggest that it is often a poor one. Figure 1 shows the impact of the total cache allocation (client + server) on the overall miss ratio. The system simulated consists of eight clients and one server. All client caches are managed with the LRU replacement strategy, and one of four possible strategies is chosen for the server cache: LRU, LFU (least-frequently-used), FBR (frequency-based-replacement[1] [10]), and Belady's MIN [11]. As the size[2] of client caches increases, the relative effectiveness of LRU as a management strategy for the server cache diminishes. While the introduction of cache is clearly beneficial, each cache configuration reaches a point beyond which adding further cache provides little

[1]FBR is a "hybrid" policy that behaves like LRU under certain conditions and like LFU under others.

[2]All cache sizes in this paper are expressed in terms of 8 Kbyte blocks

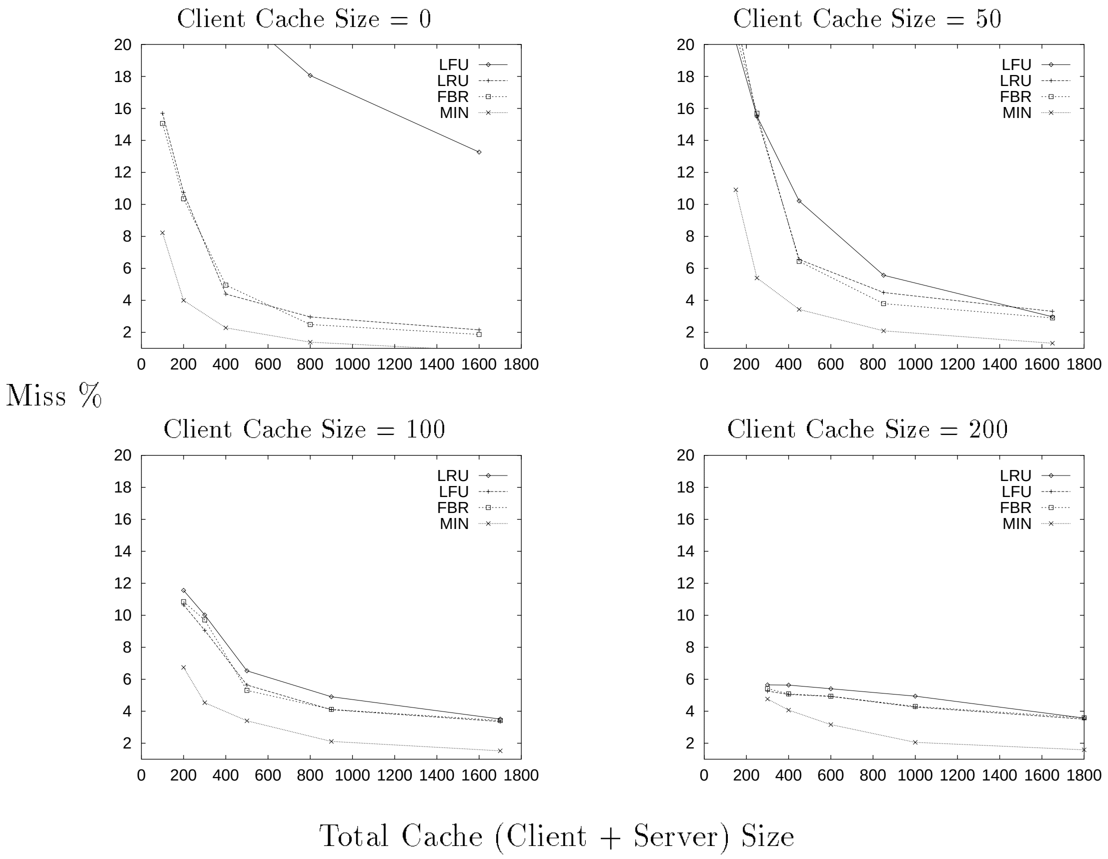

Figure 1. Overall cache miss ratios (LRU replacement at the 8 client caches, various policies at the server cache).

or no performance gain. The reason for this is seen in Figure 2, which shows that the miss ratio at the server often *increases* as more cache is added at the clients. This unfortunate behaviour is directly attributable to changes in workload characteristics, changes that are investigated more closely in the following sections.

3 The nature of requests for file service

In order to manage any resource effectively it helps to know as much as possible about the nature of requests for it. This provides motivation for detailed study of traces of requests for resources. Not only does such a study say something about workloads in typical systems (since the traces are normally collected from a "typical system") but the data collected can be used as input to controlled experimental investigations. For this work the focus is on requests for file service by applications running on client workstations.

Several previous studies have addressed the characterization of requests for file service. Results reported in [2], [10], [12], [13] and [14] suggest that file references exhibit properties of temporal and spatial locality similar to those found in memory reference traces. In other words, the same tendencies for references to information items to be clustered spatially (spatial locality) and to persist over substantial time periods (temporal locality) that are directly responsible for the success of any memory hierarchy apply to file system activity as well.

Locality can be measured directly (as in [15] for example) or inferred by its effects. For this work the analysis was based on an examination of stack depth frequencies, for both LRU (least-recently-used) and LFU (least-frequently-used) stacks. File blocks are arranged in two dynamic lists, one ordered by *recency* of use and the other by *frequency* of use, and the frequency of reference to each position in each list is computed. Plots of these frequencies are generated. The tendency for persistent block reuse (i.e. temporal locality) is reflected in the LRU stack depth frequency plot by a predominance of references to the most-recently-referenced end of this list, while the tendency for block reuse on a frequency basis is reflected in the LFU stack depth frequency plot by a predominance of references to the most-frequently-referenced end of this list. Both of these plots can tell us

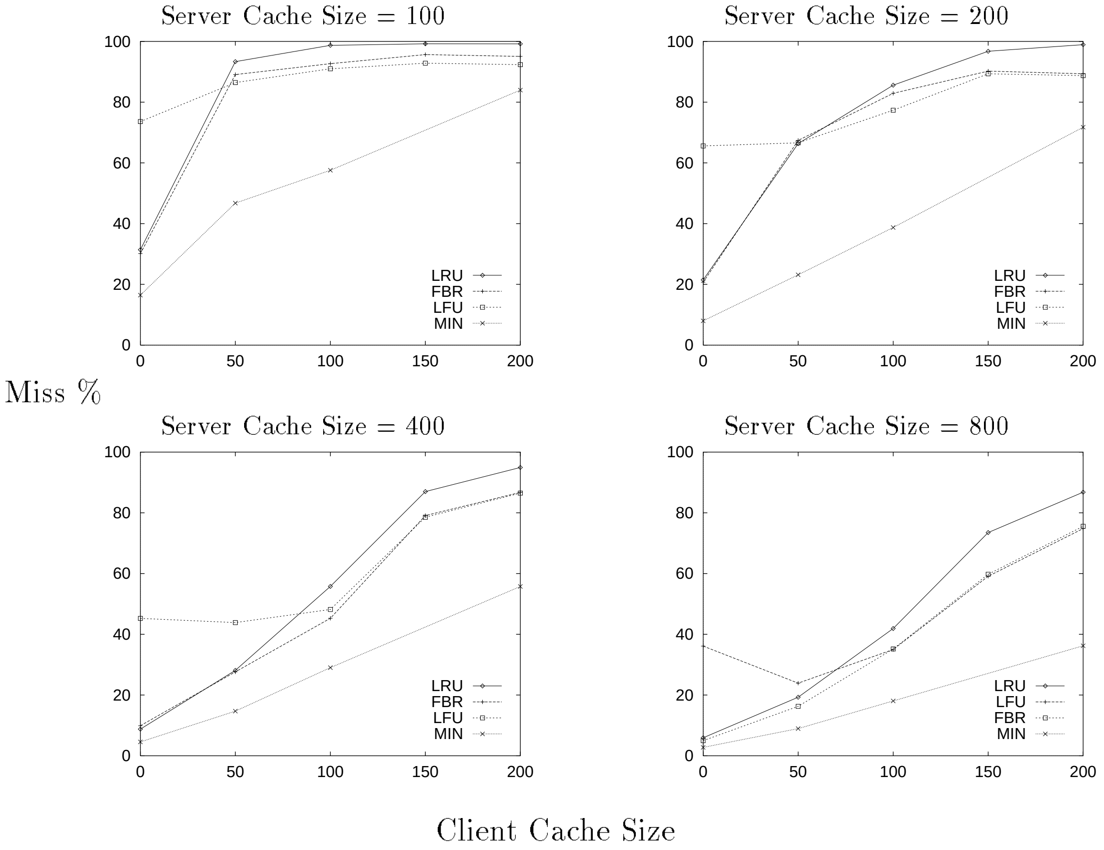

Figure 2. Cache miss ratios at the server (LRU replacement at the 8 client caches, various policies at the server cache).

something about the expected effectiveness of a cache management policy, as will be demonstrated.

3.1 Trace collection

The file reference traces that formed the basis for this work were collected from individual applications running on an HP 9000 series 400 workstation running as a client in a small client/server network in the Distributed Systems Performance Laboratory at the University of Saskatchewan. Trace collection was made possible through modifications made to the HP-UX operating system kernel to record file I/O events. In particular, each call to the internal kernel routine *getblk()* [16, 17] was recorded. Since *getblk* is used by almost all other file manipulation system calls this is sufficient for the desired purpose. Each trace record contains the device id, inode number and logical block number of the file referenced, as well as a time stamp and a read/write indicator. The (device id, inode number, logical block number) triples can be used to represent a unique disk block, with one exception. Since files may be removed and created during the span of a tracing period, some inode numbers may be reassigned to new files. If this situation were not properly accounted for, the temporal locality in a trace would be slightly exaggerated. To account properly for this effect, the gathered traces also contain "remove" events produced by tracing the *vnsremove()* system call. These events provide enough information to invalidate the appropriate blocks in the caches when the file the block belongs to is removed.

Eight different traces were collected from "representative" applications, ranging from compilation, to text editing/processing, to graphics/image processing. Table 1 summarizes the gross features of the eight traces. The traces range in duration from roughly half an hour to almost four hours, and in length from about 3000 file reference events to about 150,000 events. This trace suite was felt to represent the range of file access activity typical within a university research environment.

3.2 Characterization of the traces

The goal of this exercise is to determine the nature

Table 1. Trace Characteristics

Trace	Duration (hours)	Number of Reads, Writes	Unique Blocks	Dominant Activity
1	3.80	111947, 31661	5979	Kernel compiling from source code
2	1.57	5871, 1710	513	Program development (edit, compile, debug)
3	0.78	3570, 743	340	Program development (edit, compile, debug)
4	1.20	3734, 447	351	Document editing and processing
5	1.11	2595, 431	375	Document editing and processing
6	1.69	11304, 5498	2169	Program development (edit, compile, debug)
7	0.52	11301, 4372	2287	Graphics/Image processing
8	2.03	40805, 15654	7675	Graphics/Image processing

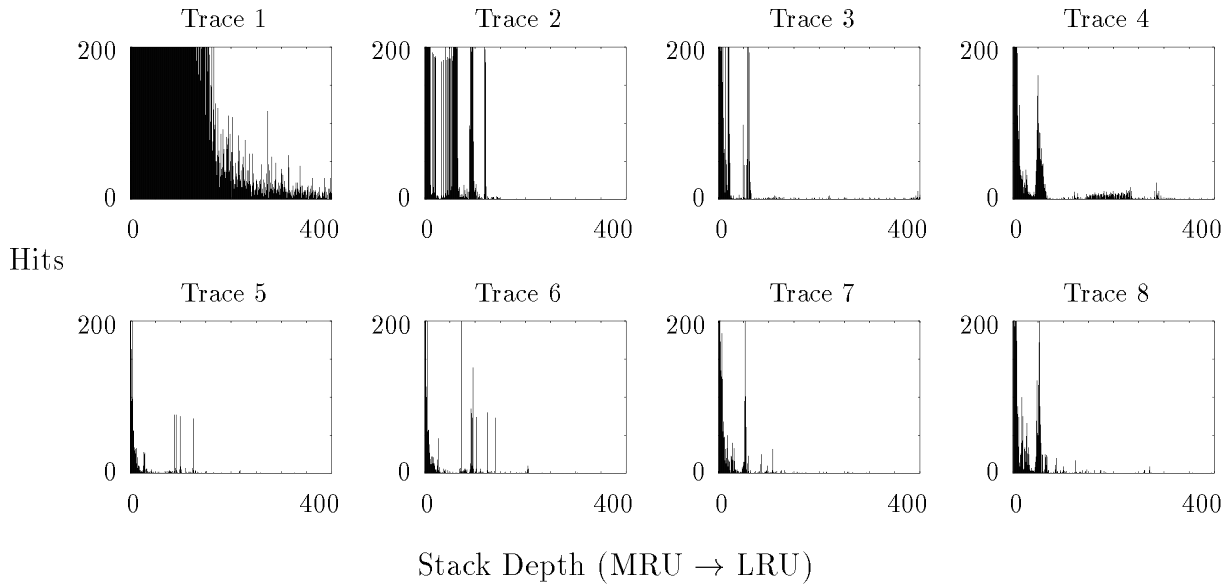

Figure 3. LRU stack depth frequencies.

of service demands represented in the input traces, for the purpose of assessing the suitability of various approaches to cache management. As described in the previous section, two plots form the basis for this characterization, and these are taken for various combinations of system configuration parameters: whether or not client cache is present, the size of each client's cache, and the number of clients accessing the server. The LRU stack depth frequency reflects the tendency for references to certain blocks to persist over time, i.e. the tendency for applications to re-reference blocks that have been referenced *recently* in the past. The LFU stack depth frequency plot reflects the tendency for applications to re-reference blocks that have been referenced *frequently* in the past.

Figure 3 shows the LRU stack depth plots for each of the eight traces (in their original unaltered form). The x-axis represents the first 400 blocks in the trace, ordered by recency of reference (block 0 is the most recently referenced block and block 400 is the 400th most recently referenced block) and the y-axis gives the frequency of reference (or number of "hits") to the block occupying each position in the stack (truncated at 200 references). In each trace there is clear evidence of a tendency to reference recently referenced blocks, i.e. blocks at the most recently referenced end of the stack. For most traces the frequency of reference diminishes rapidly as we move down the stack, towards the least recently referenced end. There are notable exceptions, however. Trace 4 shows a somewhat elongated tail while a number of traces show spikes in the distribution that are likely due to loops in the application code. In any event there appears to be clear evidence of temporal locality in all eight traces.

Figure 4 shows the LFU stack depth plots for each of the eight traces. Here the x-axis represents the first

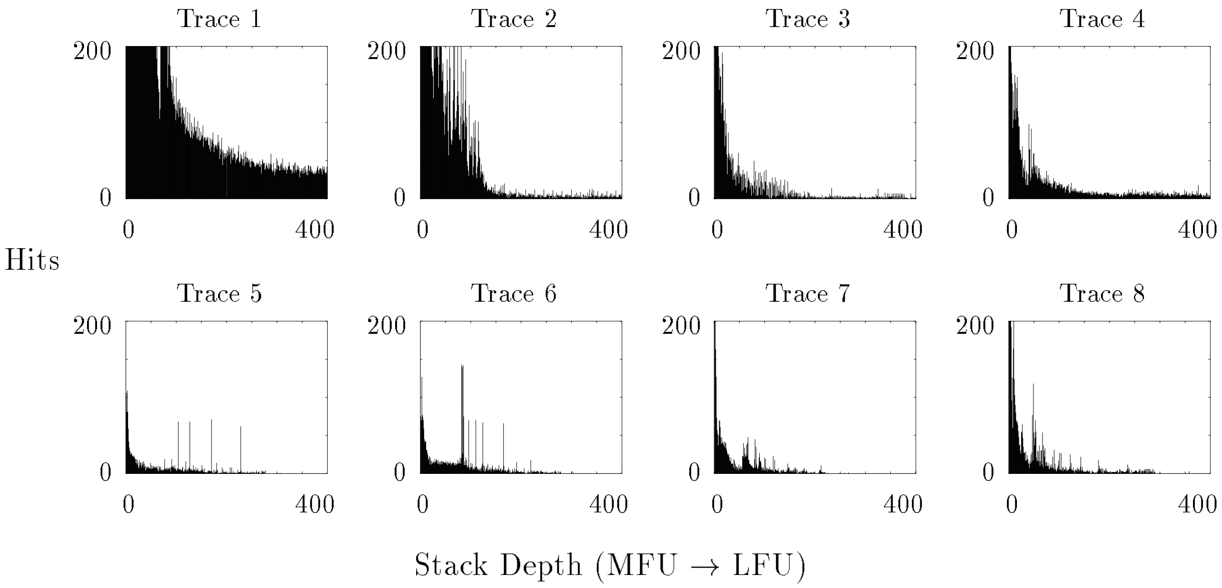

Figure 4. LFU stack depth frequencies.

400 blocks in the trace, ordered by frequency of reference (block 0 is the most frequently referenced block and block 400 is the 400th most frequently referenced block) and the y-axis gives the frequency of reference to the block occupying each position in the stack (again, truncated at 200 references). There is a clear tendency to reference the most frequently referenced blocks, but in every case the tails are longer than they were in the LRU stack depth plots. This suggests that a cache managed according to the LFU policy might experience more misses on reference streams such as these than might the same cache managed according to LRU, since references to infrequently referenced blocks are more common than references to old blocks, i.e. it is less often a bad choice to replace an old block than it is to replace an infrequently referenced one. An obvious example of this is a block that is referenced for the first time. The frequency count is low but it is probably unwise to replace this block since the likelihood of continued reference to it is high (as shown by the LRU plot). This is why cache designers rarely choose to manage their caches by the LFU policy.

This analysis reflects the sequence of requests emerging from the individual applications. In the client-server environment, this sequence is precisely what is presented to the cache on the client on which the application is running as these requests for file service are made. A cache at the server, however, will see a different sequence, for the reasons mentioned previously: the aggregation of individual request streams from multiple clients, and the removal of those references that are hits to the client cache. As a result, the workload characteristics that should dictate the choice of replacement policy are changed. The effect that each of these factors has on the characteristics of the reference traces is considered in the following sections.

4 Experiments in server workload characterization

The effect of client caching was studied through a series of trace-driven simulation experiments. The experiments are divided into two groups: first a set of experiments with a single client to investigate the effect of placing cache at the client, and second a set of experiments with multiple clients to investigate the effect of interleaving request streams. The single client results were published previously [3] but are presented here again so that the complete picture is provided.

4.1 Methodology

The system simulated consists of a set of diskless client workstatations receiving file service from a single file server equipped with a disk of unlimited capacity. The experimental variables include the number of clients and the size of the client caches (all client caches are the same size). Read caching is done as described previously; all write operations are written through to disk directly, updating each cached copy along the way. No other network activity is simulated.

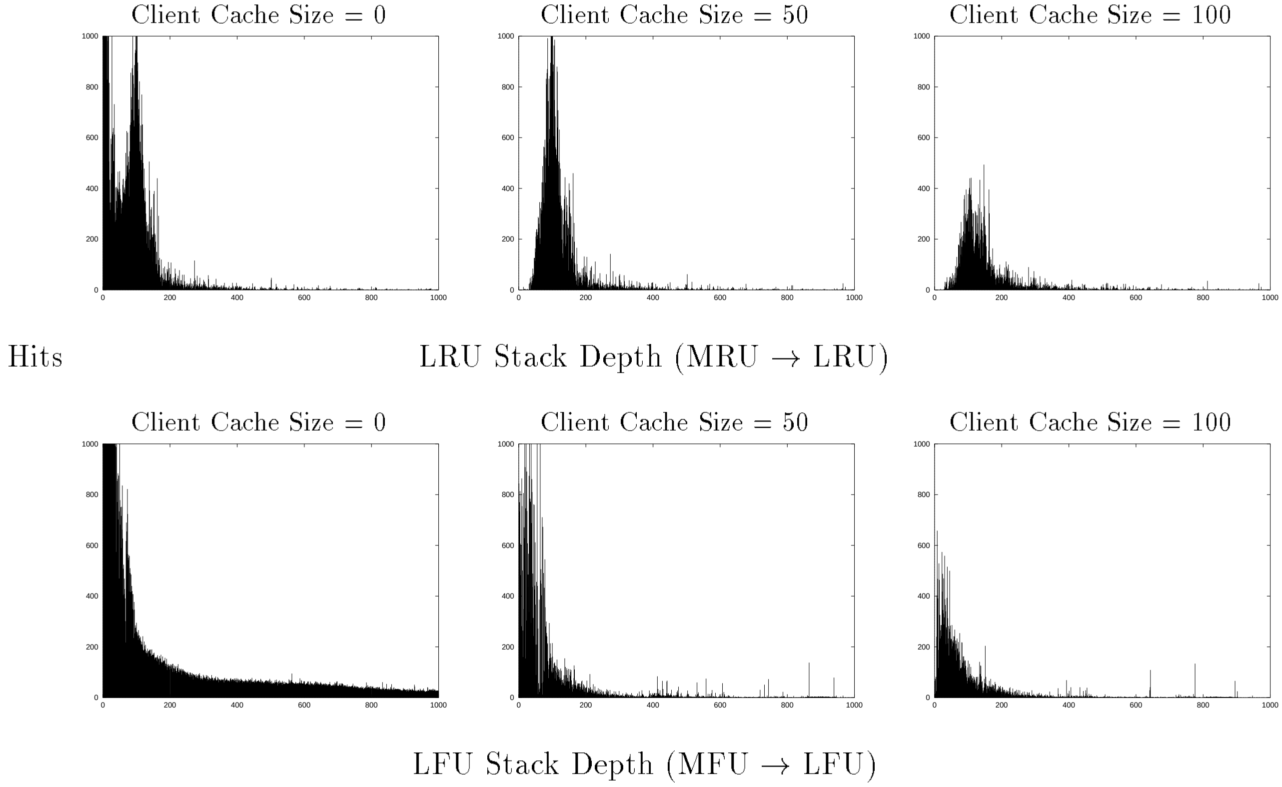

Figure 5. The effect of client caching (single client).

Since eight different traces are available, up to eight different applications can be simulated at once. If only a single client is simulated, the simulation ends when the chosen trace is exhausted. If multiple clients are simulated, each client is assigned one of the traces and the simulation runs for a specified length of time. To accommodate the different trace lengths, the individual traces are stitched end-to-end on a given client, in effect simulating on each client the restart of a terminated application. If more than eight clients are simulated (the number of different traces), the stitched traces are assigned to the clients in a cylic fashion, so that each client is running one trace, but the same application may be running on more than one client as a result.

4.2 The effect of client cache

A set of experiments was run with a single client to investigate the effect that the introduction of a client cache has on the workload presented to the file server. The results presented in this section are all based on experiments done using trace 1. A client cache of 0 blocks means there is no caching at the client, and thus these results provide the unaltered workload which is used as the basis for comparison.

It is expected that workload characteristics relating to temporal locality will be strongly affected by the presence of client caching. Repeated references to cached blocks, for example, will be resolved at the client cache and will not be present in the stream of requests passed on to the server. In other words, the server sees only those references that miss at the clients, along with any writes that must go through to the disk whether they were hits or not. This "filtering" of the references could have a profound impact on the performance of a locality-based cache management strategy at the server. This is confirmed by the results of performance experiments reported elsewhere ([3], [4]).

Figure 5 shows the effect of client caching on the reference stream seen at the server. These plots of LRU stack depth and LFU stack depth were made *after* the effects of client caching to show the reference behaviour in the "filtered" reference streams. With a client cache size of 0 (i.e. no client caching) there is a strong tendency towards references at the most-recently-used end of the LRU stack and a weaker tendency towards references at the most-frequently-used end of the LFU stack (as reflected in a much more prominent tail). The in-

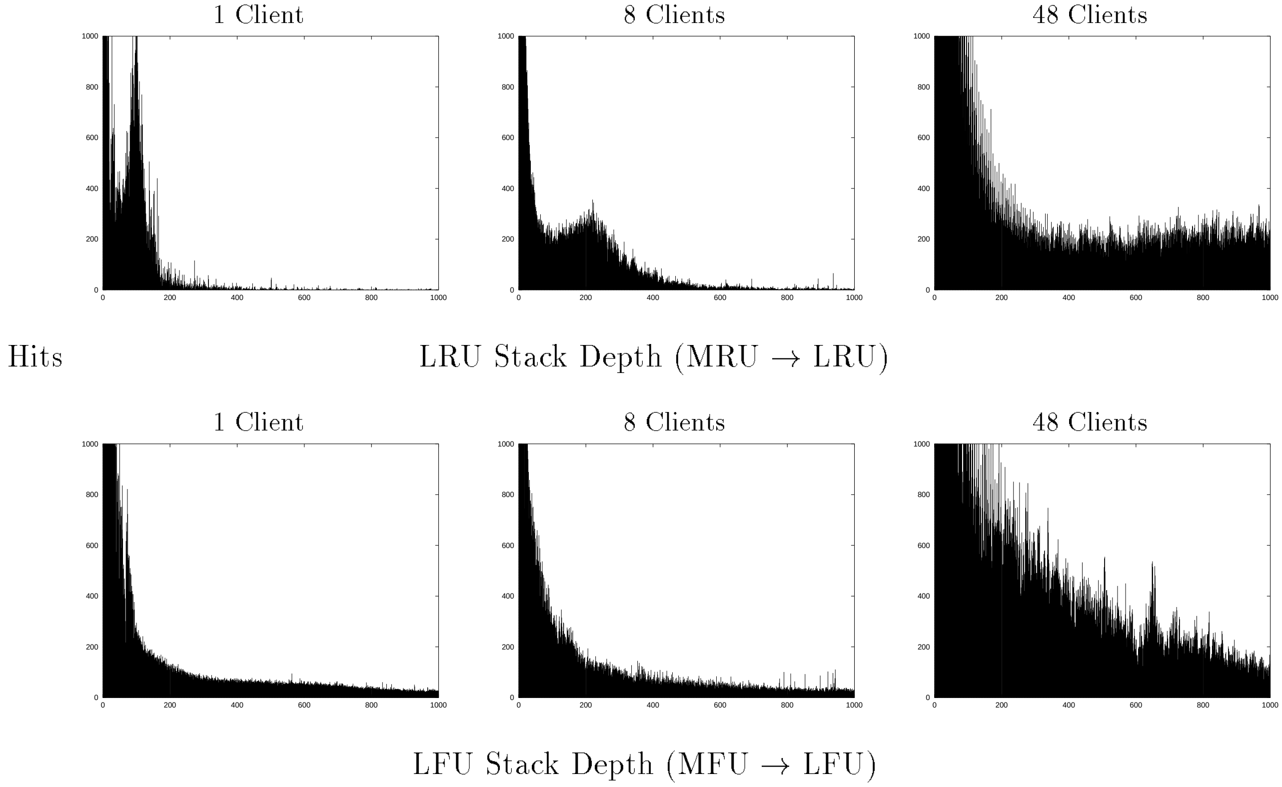

Figure 6. The effect of multiple clients (no client caching).

troduction of a client cache has a clear effect. In the LRU stack, it is clear that many of the references to blocks at the most-recently-referenced end have been removed from the reference stream. This effect is even more pronounced as the size of the client cache increases from 50 blocks to 100 blocks. The "flattening" of the LRU stack depth curve is a clear indication that temporal locality is removed by the client cache. Similarly, there is a tendency for references to blocks at the most-frequently-referenced end of the LFU stack to be removed as well, although this tendency is not nearly so strong. These plots suggest that the use of a recency-based caching strategy at the server is likely to be increasingly ineffective as client cache sizes increase. The LFU stack depth plots suggest more hope for success with frequency-based strategies at the server.

4.3 The effect of multiple clients

The presence of multiple clients means that the stream of requests seen at the server is an aggregation resulting from the interleaving of separate request streams from individual clients. The effect of this interleaving is a potential further disruption of locality in the request stream seen at the server.

Figure 6 shows the effect of multiple clients (with no client caching) on the reference stream seen at the server. With a single client, the plots are simply the baseline cases in Figure 5: there is a strong tendency towards references at the most-recently-referenced end of the LRU stack and a weaker tendency towards references at the most-frequently-referenced end of the LFU stack. The introduction of additional clients once again has an effect shown by the emergence of more pronounced tails as the number of clients increases, reflecting a weakening of preference for recently or frequently referenced blocks.

Comparing Figures 5 and 6, it appears that the presence of client caches will be a more significant factor in server cache performance than will be the presence of multiple clients. Futher research is needed toward the design of server cache management strategies that can respond appropriately to the workloads presented in realistic client-server environments. Some hint of what might be expected is given in Figure 7, which is a repeat of the experiment leading to Figure 6, but with caches of 100 blocks at each of the clients. These plots suggest very little hope for recency-based caching at the

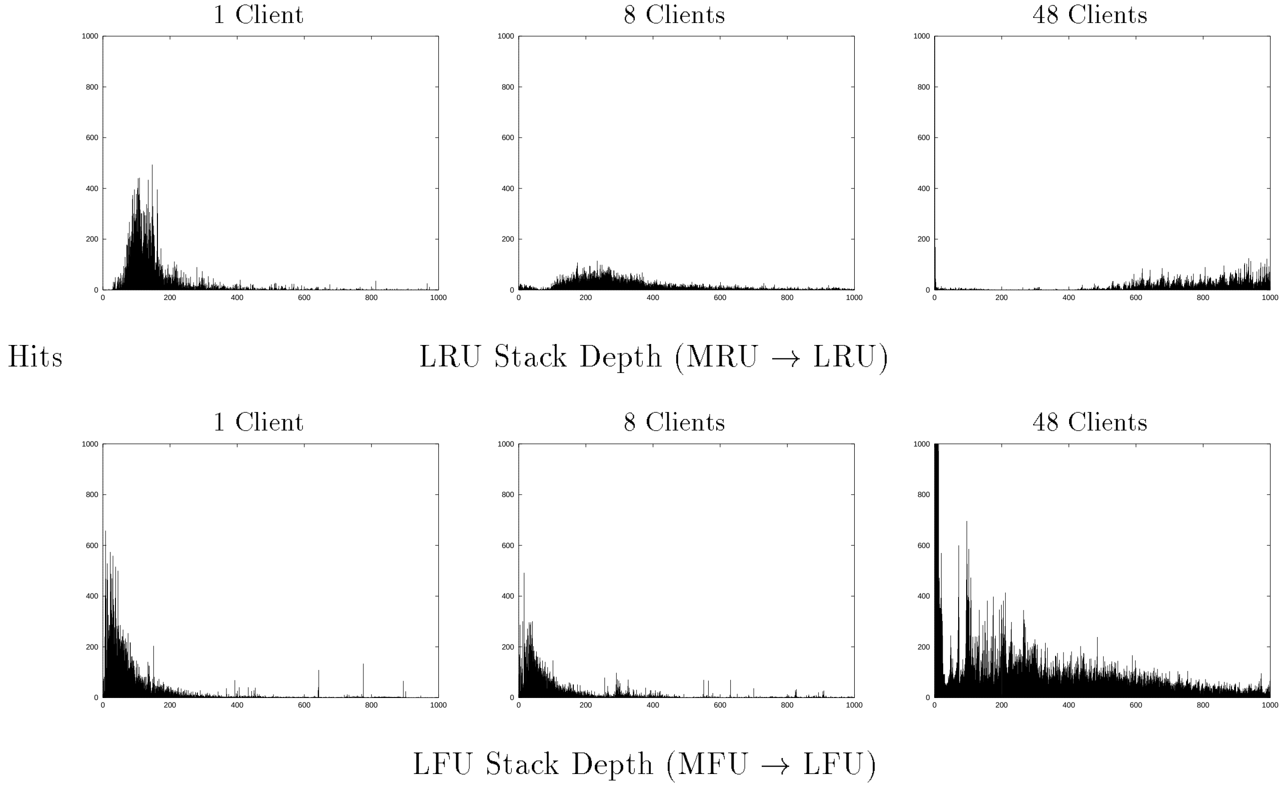

Figure 7. The combined effect (100 block caches at clients).

server since there is virtually no remaining preference for recently-referenced blocks. There is some remaining preference for frequently-referenced blocks, however, suggesting that frequency-based caching might be more effective than recency-based caching at the server.

5 Summary and conclusions

The goal of this research was to investigate the effect of client caching on the workload presented to a file server in a distributed file system. This can have a profound influence on the performance of strategies selected to manage a cache at the file server. Custom instrumentation of the HP-UX operating system kernel made it possible to gather disk block reference traces of real workload. These were used as the basis for this investigation.

The results presented in this paper confirm our belief that cache management at the server is a different problem than the traditional problem of managing a single level cache since the presence of lower level caches fundamentally alters the characteristics of the workload presented to the higher levels. Our analysis of the stack depth plots for a two-level caching system shows that as the size of the first level (client) cache increases, the reference stream is altered so that temporal locality may no longer be present. This means that the reuse of recently referenced blocks at the second level (the server) is less likely than it is at the first level. This explains earlier observations relating to the ineffectiveness of recency-based replacement policies at a server cache. While recency-based policies are very effective in managing a single level cache or in managing a client cache in a multilevel caching system, they would seem to be inappropriate choices for higher levels because the characteristics of the reference stream at that level are quite different. Workload characteristics observed at the higher levels suggest that there remains some tendency to reuse of frequently referenced blocks, and thus policies based on frequency of use (eg., LFU, FBR) may be better choices. It is clear that the management of a multilevel caching system is a complex issue warranting further study.

Acknowledgements

The authors would like to thank Michael Molloy, from Hewlett-Packard, for providing technical assistance and several tools used to instrument the HP-UX 8.0 operating system. The tracing tools were installed by Darryl Willick, who also collected the original traces. Greg Oster helped put together the camera-ready copy. Funding was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), through Research Grant OGP0003707, Strategic Grant STR0045250, and the Summer Undergraduate Awards Program.

References

[1] Lazowska, E.D. and Zahorjan, J., "File Access Performance of Diskless Workstations", *ACM Transactions on Computer Systems*, Vol. 4, No. 3, August 1986, 238-268.

[2] Makaroff, D.J. and Eager, D.L., "Disk Cache Performance for Distributed Systems", *Proc. 10th International Conference on Distributed Computing Systems*, Paris, France, May 1990, 212-219.

[3] Willick, D.L., Eager, D.L. and Bunt, R.B., "Disk Cache Replacement Policies for Network Fileservers", *Proc. 13th International Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, May 1993, 2-11.

[4] Chen, K., Bunt, R.B. and Eager, D.L., "Write Caching in Distributed File Systems", *Proc. 15th International Conference on Distributed Computing Systems*, Vancouver, Canada, May 1995, 457-466.

[5] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B., "Design and Implementation of the Sun Network File System", *Proc. 1985 Summer USENIX Conference*, Portland, Oregon, June 1985, 161-203.

[6] Birrell, A.D., Hisgen, A., Jerian, C., Mann, T., and Swart, G., *The Echo Distributed File System*, SRC Research Report 111, Digitial Equipment Corporation, September 1993.

[7] Nelson, M.N., Welch, B.B. and Ousterhout, J.K, "Caching in the Sprite Network File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, 134-154.

[8] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., and West, M.J., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, 51-81.

[9] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, W.H., and Streere, D.C., "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990, 447-459.

[10] Robinson, J.T. and Devarakonda, M.V., "Data Cache Management Using Frequency-Based Replacement", *Proc. 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Boulder, Colorado, May 1990, 134-142.

[11] Belady, L.A., "A Study of Replacement Algorithms for a Virtual Storage Computer", *IBM Systems Journal*, Vol. 5, No. 2, 1966, 78-101.

[12] Smith, A.J., "Disk Cache Miss Ratio Analysis and Design Considerations", *ACM Transactions on Computer Systems*, Vol. 3, No. 3, August 1985, 161-203.

[13] Majumdar, S. and Bunt, R.B., "Measurement and Analysis of Locality Phases in File Referencing Behaviour", *Proc. Performance 86 and ACM SIGMETRICS 86 Joint Conference on Computer Performance Modelling, Measurement and Evaluation*, Raleigh, NC, May 1986, 180-192.

[14] Ramakrishnan, K.K., Biswas, P. and Karedla, R., "Analysis of File I/O Traces in Commercial Computing Environments", *Proc. 1992 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Newport, Rhode Island, June 1992, 78-90.

[15] Bunt, R.B. and Murphy, J.M., "The Measurement of Locality and the Behaviour of Programs", *The Computer Journal*, Vol. 27, No. 2, August 1984, 238-245.

[16] Bach, M.J., *The Design of the UNIX Operating System*, Prentice-Hall, 1986.

[17] Leffler, S.J., McKusick, M.K., Karels, M.J. and Quarterman, J.S., *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison Wesley, 1989.