# Adverse Filtering Effects and the Resilience of Aggregating Caches *

Ahmed Amer and Darrell D. E. Long
Computer Systems Laboratory
Jack Baskin School of Engineering
University of California, Santa Cruz
a.amer@acm.org, darrell@soe.ucsc.edu

## ABSTRACT

Cache hit rates can be dramatically affected by the pre-filtering effects of successive cache stages. Where the intervening cache is of comparable size to the target cache, this effect can render traditional replacement policies totally ineffective. We propose that such scenarios are increasingly common in distributed and mobile environments. We present a study of the effects of intervening cache filtering on hit rates, demonstrating the rapid degradation of performance for traditional cache replacement schemes. We also compare the performance of a mechanism we have recently developed, the aggregating cache. The aggregating cache exhibits tremendously improved performance degradation under even the most problematic scenarios.

## Keywords

aggregating cache, prefetching, implicit prefetching, cache filtering effects

## 1. INTRODUCTION AND MOTIVATION

Caching algorithms are most often tested as a management system of an intermediate store of limited capacity, lying between a data client and a storage server. Indeed, this is the most common scenario for a cache, where it is often used as an intermediate store, faster than a larger, slower, main store. This is true of caching between levels of a memory hierarchy, or caching of data retrieved through slow, or high-latency, interfaces such as a distant network server. Such a model we will refer to as a single-stage caching model (Figure 1(a)). With the growing complexity and scale of distributed data storage environments, we can now see examples of caches that do not follow this single-stage model. The important distinction of a multi-stage cache is the location of the high-cost (high latency, and/or low throughput) interface. Figure 1(b) shows a simple two-stage model. The data client is considered a source of data access requests, which are then filtered through two caches before being passed on to the main store (request sink).

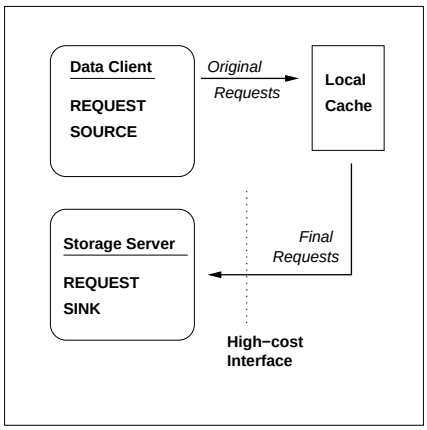
*Supported in part by the National Science Foundation award CCR-9972212, and the Usenix Association.

The filter cache is logically near to the data client, and the second cache is located logically closer to the storage server. Our main concern is the effectiveness of such a second cache in light of the filtering effects of the first stage cache. Throughout this paper we will refer to the first stage cache of the two-stage model as the "filter," while the second stage will simply be described as the "cache." This scenario, two comparable caches with a limited cost for inter-cache access, can be found in many distributed computing environments. Examples include mobile file hoards when on a LAN, where the access requests to a server cache have already been pre-filtered through an increasingly large intervening cache – the mobile computer's local storage. Another example includes distributed storage systems having large client-side caches, and high-speed network interfaces to a high performance storage server. In such a system, client accesses to local storage are comparable to remote data retrievals, and so we are questioning the usefulness of server-side caching. In a system like Sprite [14], with a high performance network, we would therefore be considering the effects of a client cache on a server cache's hit rate.

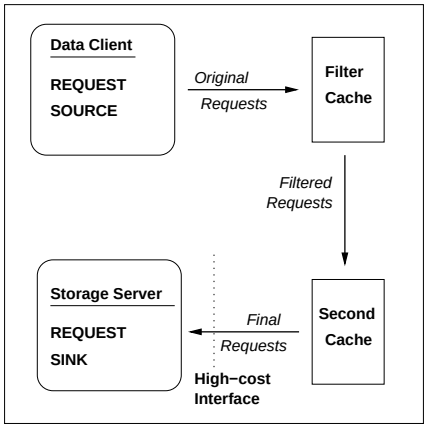
We will demonstrate how an LRU filter, when comparable in size to the cache, can render straightforward LRU and LFU caching useless, with hit rates rapidly approaching zero. We go on to describe how a caching mechanism based on predictive grouping of related access events, the aggregating cache, is much more resilient to this filtering effect.

## 2. THE AGGREGATING CACHE

The most prominent feature of the aggregating cache is the retrieval of groups of related files from the remote storage server. Although originally intended to reduce the impact of high latency data retrieval requests, the aggregating cache was found to increase cache hit rates in addition to its intended use of reducing the aggregate number of demand fetches [1]. Figure 2 presents a conceptual view of this scheme. File access requests go through the local file system interface and, if not satisfied locally, are forwarded through a local cache manager to the file store managed at a remote server. The major difference from prior art in distributed caches is the mechanism of maintaining server-side relationship information, and its use to retrieve multiple related




(a) *Simple one-stage caching model.*




(b) *Two stage cache configuration.*

**Figure 1: Single and two stage caching models.**

files per request. This allows the client and server to transparently utilize available bandwidth to fetch groups of files based on observed access patterns.

The server component maintains per-file relationship information, keeping track of a strictly limited group of related files. When a client is forced to perform a high-latency remote request, the server and client components of the aggregating cache can cooperate to opportunistically "transmit" a group of related files. This offers the opportunity to fully utilize the transport medium, while avoiding traditional problems with predictive techniques at reducing latency effects. For example, with file prefetching there is a risk of increasing perceived latencies by initiating incorrect prefetches. Recent experience with file prefetching implementations [8] has demonstrated the negative impact of user-level prefetchers, contending with user-generated requests, and the need for system-level preemptible mechanisms.

We build groups by tracking a fixed number of successors

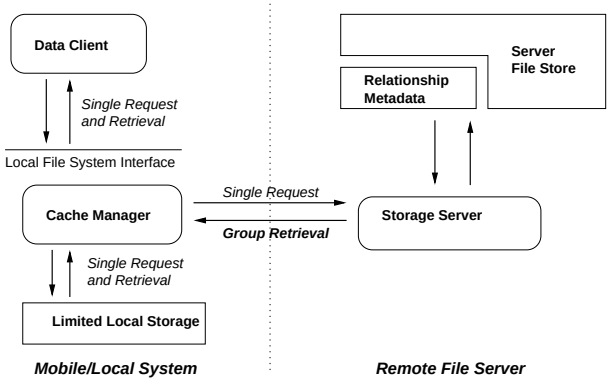



**Figure 2: General aggregating cache model.**

for all files accessed. A successor is simply a file accessed immediately following the current file. This information can easily be maintained as file metadata.

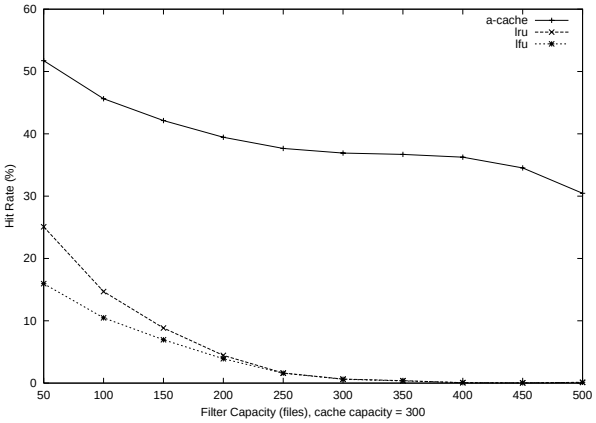
# 3. EXPERIMENTAL RESULTS

Simulations were run on file system traces gathered using Carnegie Mellon Univeristy's DFSTrace system [13]. The tests covered five systems, for durations ranging from a single day to over a year. The traces represent varied workloads, particularly *mozart* a personal workstation, and *ives*, a system with the largest number of users. These traces provide information at the system-call level, and represent the original stream of access events, not filtered through a cache. For these CMU traces we are measuring the hit-rate for a whole file cache calculated based on file open requests. This assumes a coarse granularity for the analysis, focusing on patterns of file requests, more representative of file hoarding scenarios.

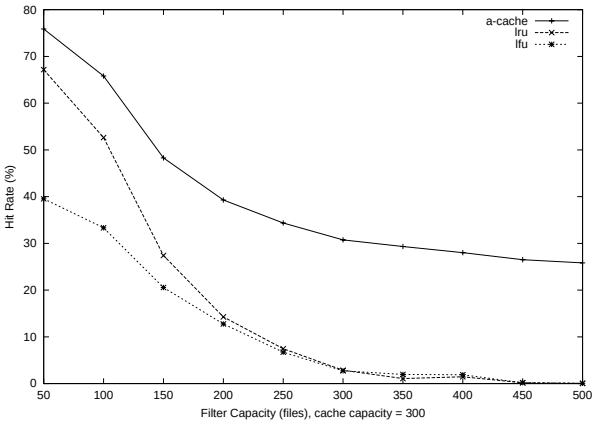
For a finer level of analysis we also considered individual requests for file data, drawn from the traces provided by Drew Roselli at the University of California, Berkeley [15]. To eliminate any interleaving issues, these UCB traces were processed to represent the workloads of individual workstations, and simulations were run against both instructional and research machines.

The most interesting results occurred when the filter capacity grows comparable to the cache size, with the usefulness of a traditional cache disappearing as the filter capacity exceeds the cache size. The critical point for relative cache and filter sizes appears to be the equality point. For one set of graphs we fix an arbitrary filter size, and observe the performance of the cache as we increase its capacity. For the other set of results we will fix an arbitrary cache size, and observe the effect of increasing the intermediate filter capacity on cache performance. For the following graphs we chose to fix cache capacity at 300 items, and to fix filter capacity at 100 items. These capacities were chosen arbitrarily, but the critical nature of the equality point was observed at all capacities tested.

Figure 3 shows the effects of varying the capacity of the filter on the hit rate of our cache. We compare three cache management schemes: LRU replacement, LFU replacement, and

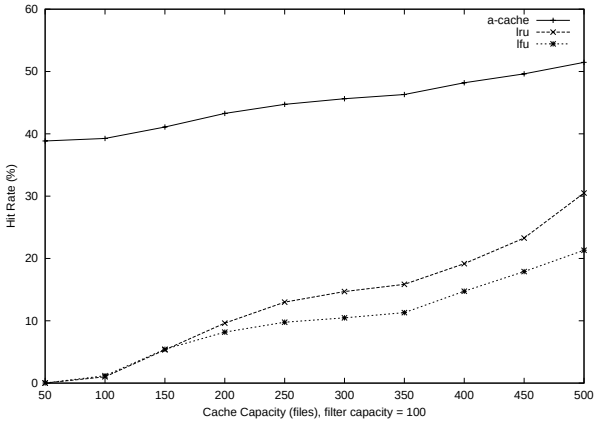



(a) *mozart*

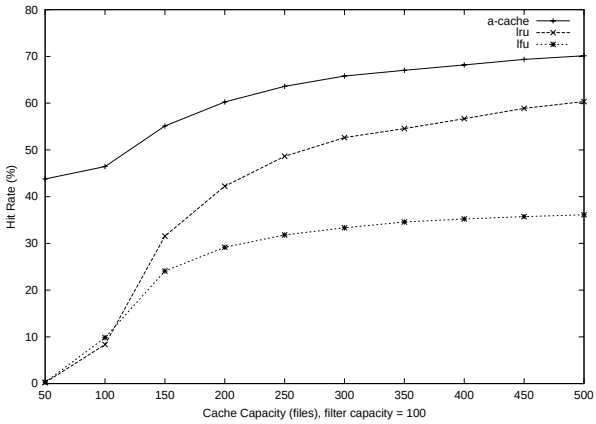



(b) *ives*

**Figure 3: CMU trace cache hit rates for varying filter sizes.**




(a) *mozart*



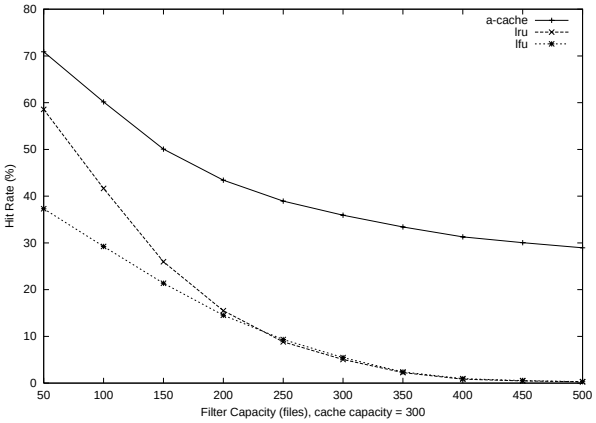

(b) *ives*

**Figure 4: CMU trace cache hit rates for varying cache sizes.**

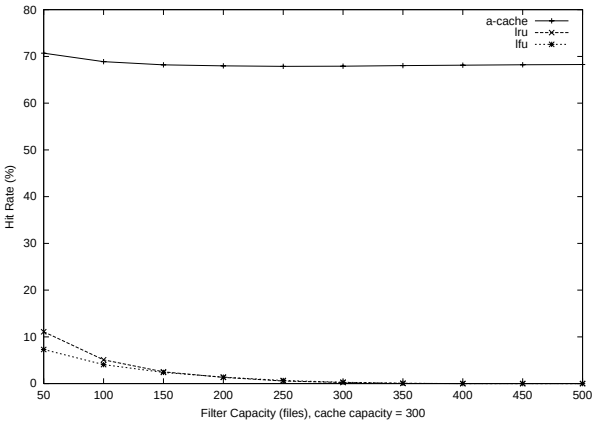
a basic aggregating cache (that tracks and retrieves groups of up to five related files). It is no surprise that LRU outperforms LFU replacement, but the most important observation from the figure is how rapidly the performance of the cache degrades. As the filter size approaches the fixed cache size, we see a dramatic drop in the hit rate for our cache. This is consistent both for the dedicated workstation *mozart*, and the more populous system *ives*. Regardless of the nature of the request source (multi-user or dedicated system) this degradation appears very rapidly, and both LRU and LFU caching quickly become useless. In contrast, the aggregating cache maintains consistent performance, and shows a much milder degradation in hit rate. All independent locality of reference is quickly masked by the intervening cache, rendering straightforward LRU caching useless while the aggregating cache manages to maintain a higher hit rate in spite of this. This is thanks to the ability of this scheme to capture inter-file relationships. Although the intervening cache masked all observable locality for LRU and LFU, these schemes assume an independence of access. Fortunately, the interdependence among file access events is not completely masked by filtering, allowing the aggregating cache to sustain a hit rate even when requests are filtered through a filter larger than the cache.

It should be noted that the aggregating cache is being supplied with exactly the same information as the LRU and LFU schemes. The design of the aggregating cache allows for the forwarding of relationship information, gathered at the data client (request source), to the storage server. If this scheme were used we would expect higher hit rates (above 90% for these workloads), equivalent to an aggregating cache free of the filtering effects of an intervening cache. These results represent the performance of an aggregating cache when confronted with the same problematic access behavior as presented to the LRU and LFU schemes.

Figure 4 shows the improvement in cache hit rates as cache size is increased relative to a fixed filter size. Again we see a rapid recovery of LRU and LFU performance as the

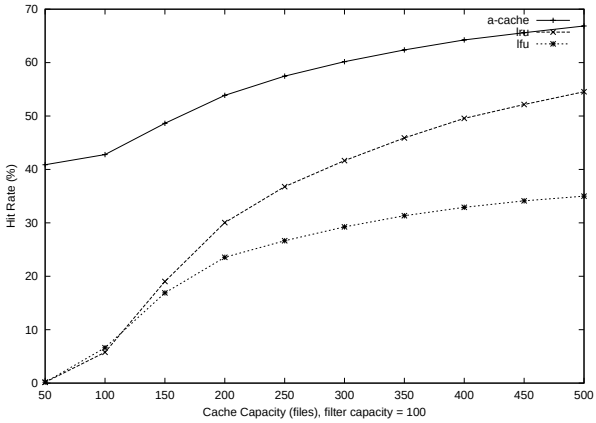



(a) *Instructional Worsktation*

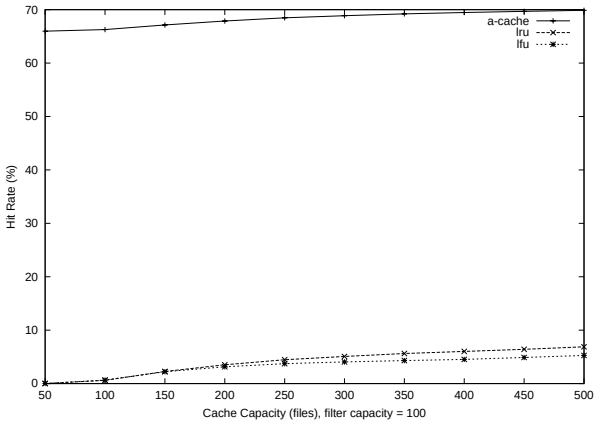



(b) *Research Workstation*

**Figure 5: UCB trace cache hit rates for varying filter sizes.**




(a) *Instructional Workstation*




(b) *Research Workstation*

**Figure 6: UCB trace cache hit rates for varying cache sizes.**

cache size increases. The aggregating cache remains ahead of both, and will remain so, as the limit is a simple one-stage cache scenario for which the aggregating cache is known to outperform the other two schemes.

A trend observable in both Figures 3 and 4 is the rate of decline for the LRU/LFU caches *vs.* the aggregating cache. Other experiments have shown us that the *mozart* workload tends to be more predictable than *ives*. The *mozart* trace, with fewer users and subsequently fewer independent sources of requests, degrades more rapidly due to filtering, and yet the aggregating cache is better able to maintain hit rate. While with *ives* we see a more gradual degradation in LRU and LFU, while the aggregating cache is slightly less adept at tolerating the filtering. This supports the reasoning that the aggregating cache's performance is due to its ability to capture higher-level dependencies between file access events. More independent accesses result in reduced filtering impact and reduced improvement through use of the aggregating cache.

The CMU traces are most useful for their extensiveness, as they cover well over a year of recorded workloads, but may be too old for to be fully representative of current workloads. Also, they often lack recordings of more detailed access events, *e.g.* specific read and write system calls. For these reasons we repeated our experiments on the UCB traces [15]. Specifically we will present results of a typical research workstation and an instructional system. Figures 5 and 6 illustrate the same results as Figures 3 and 4 respectively.

Both figures demonstrate the same trends we observed for the CMU traces, with a notably higher performance for the aggregating cache, especially for the more predictable research workload. In fact, the aggregating cache could be considered to have been only slightly affected by the growing size of the filter, while the LRU and LFU are even more seriously affected. Our hypotheses appear to hold more strongly for the more recently recorded workloads.

## 4. RELATED WORK

Our work has drawn from work in distributed file systems, predictive prefetching, and working set identification. In particular, our model is based on ideas of cache management introduced with such systems as Sprite [14], AFS and later Coda [7]. These systems have undergone analysis of their caching behavior, with Sprite the subject of a particularly detailed analysis [2]. In this context our work has looked to server-side cache hit rates, an area not directly considered by these studies.

Griffioen and Appleton presented a file prefetching scheme based on graph-based relationships [6]. Their probability graphs are very similar in nature to our relationship model, but are limited to tracking frequency of access within a particular "lookahead" window size. Our model, on the other hand, is primarily based on immediate recency (succession), and requires no minimum probability to initiate a prefetch, but opportunistically fetches related files. Our aggregating caches are also independent of any concept of lookahead window size. Later work by Kroeger and Long [9] compared the predictive performance of the last successor model to Griffioen and Appleton's scheme, and more effective schemes based on context modeling and data compression. The use of the last successor model for file prediction, and more elaborate techniques based on pattern matching, were first presented by Lei and Duchamp [12]. The first proposed application of data compression techniques to file access prediction was presented by Vitter and Krishnan [18]. Earlier work on the automatic detection of working sets includes the work of Tait and Duchamp [17]. "Dynamic Sets" presented another model for using file groups, but instead of automatic detection, dynamic sets provides mechanisms for applications to specify groups of files in which they are interested [16]. The *Seer* project also attempted to use file groups for mobile file hoarding [11]. Seer used the notion of a semantic distance coupled with shared-neighbors clustering to build file hoards.

Our study has considered the general case of requests being filtered through a client/intervening cache of comparable capacity to the server cache. In the Web domain, especially cooperative caching and web proxies, Wolman *et al* have addressed similar issues [20, 19]. In that context, the authors were specifically interested in the usefulness of cooperative caching schemes at different system scales. Other recent work in this area includes the Hummingbird file system, which is very effective at improving the performance of caching web proxies [5]. The prefetching nature of the aggregating cache is similar to Bestavros' work on the use of speculation [3] to reduce server loads and improve service times, and later work by Duchamp on "Prefetching Hyperlinks" [4]. Specifically, the similarity lies in the non-volatile maintenance of relationship information at the server, and its use to reduce server loads and service times. In contrast, our study targets general file system workloads, and is based on a more general scheme for relationship tracking. We should also point out that we do not make any assumptions about the access information used to build file groups, and the results presented in this study were based solely on the "filtered" file accesses, with detailed information about a client's access behavior being fully masked from the server. In a WWW environment, the server (or proxy) cache has the advantage of being able to receive more detailed client access information, and the additional luxury of embedded relationship hints (the hyperlinks found in most HTML documents).

## 5. CONCLUSIONS AND FUTURE WORK

We have confirmed the intuitive result that multiple stages of comparable caches can have negative interactions, resulting in the uselessness of the latter stages. With the aggregating cache we have seen considerable resilience to such effects, implying its suitability when an effective caching strategy is required closer to the server. This result has also shown that the aggregating cache can remain effective in terms of hit rates, regardless of the placement of the data gathering components. Client forwarding of relationship information is not essential.

It is likely that other predictive prefetching techniques may exhibit similar graceful degradation in performance, though we doubt that any system can guarantee avoiding filtering penalties completely. Even with prefect prediction systems there are strict performance limits, imposed by the general caching architecture under which you operate. This was made clear in an earlier study on the bounds of latency reduction in web proxy caches [10]. The performance of other predictive approaches, and alternative filtering mechanisms are subjects of future study. Further investigation of alternative performance metrics, especially aggregate remote fetches, and workloads incorporating higher levels of interleaving are still required.

## 6. ACKNOWLEDGMENTS

We are especially grateful to Tom Kroeger and Randal Burns for valuable feedback, reviews and discussions. We also wish to thank the anonymous reviewers for their helpful feedback and valuable insights. We are grateful to all the members of the Computer Systems Laboratory, for their continuous feedback, support and valuable discussions. Our lengthiest traces were kindly made available by M. Satyanaryanan of Carnegie Mellon University, through the greatly appreciated efforts of Tom Kroeger in processing and conversion. We are also grateful to Drew Roselli for providing the UCB traces used in this study.

## 7. REFERENCES

[1] A. Amer and D. D. E. Long, "Aggregating caches: A mechanism for implicit file prefetching," in *Proceedings of the Ninth International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001)*, IEEE, Aug. 2001. (to appear).

[2] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "Measurements of a distributed file system," in *Proceedings of 13th ACM Symposium on Operating Systems Principles*, (Asilomar, Pacific Grove, CA), pp. 198–212, Association for Computing Machinery SIGOPS, Oct. 1991.

[3] A. Bestavros, "Using speculation to reduce server load and service time on the WWW," in *Proceedings of CIKM '95: Conference on Information and Knowledge*

*Management*, (Baltimore, MD), pp. 403–10, ACM, Dec. 1995.

[4] D. Duchamp, "Prefetching hyperlinks," in *Proceedings of the Second Usenix Symposium on Internet Technologies and Systems*, (Boulder, CO), pp. 127–38, USENIX Association, Oct. 1999.

[5] E. Gabber and E. Shriver, "Let's put NetApp and CacheFlow out of business!," in *9th ACM SIGOPS European Workshop*, (Kolding, Denmark), ACM, Sept. 2000. Proceedings are yet to be published.

[6] J. Griffioen and R. Appleton, "Reducing file system latency using a predictive approach," in *USENIX Summer Technical Conference*, pp. 197–207, June 1994.

[7] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the Coda file system," in *13th ACM Symposium on Operating Systems Principles (SOSP)*, (Pacific Grove, CA, USA), Oct. 1991.

[8] T. M. Kroeger, *Modeling File Access Patterns to Improve Caching Performance*. PhD thesis, University of California, Santa Cruz, Mar. 2000.

[9] T. M. Kroeger and D. D. E. Long, "The case for efficient file access pattern modeling," in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII)*, (Rio Rico, Arizona), IEEE, Mar. 1999.

[10] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the bounds of web latency reduction from caching and prefetching," in *Proceedings of the First Usenix Symposium on Internet Technologies and Systems*, (Monterey, CA), pp. 13–22, USENIX Association, Dec. 1997.

[11] G. H. Kuenning and G. J. Popek, "Automated hoarding for mobile computers," in *Sixteenth ACM Symposium on Operating Systems Principles*, (Saint Malo, France), pp. 264–75, Oct. 1997.

[12] H. Lei and D. Duchamp, "An analytical approach to file prefetching," in *1997 USENIX Annual Technical Conference*, Jan. 1997.

[13] L. Mummert and M. Satyanarayanan, "Long term distributed file reference tracing: Implementation and experience," *Software - Practice and Experience (SPE)*, vol. 26, pp. 705–736, June 1996.

[14] M. N. Nelson, B. B. Welch, and J. K. Ousterhout, "Caching in the Sprite network file system," *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 134–154, 1988.

[15] D. Roselli, "Characteristics of file system workloads," Technical Report CSD-98-1029, University of California, Berkeley, Dec. 23, 1998.

[16] D. C. Steere, *Using Dynamic Sets to Reduce the Aggregate Latency of Data Access*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Jan. 1997.

[17] C. D. Tait and D. Duchamp, "Detection and exploitation of file working sets," Tech. Rep. CUCS-050-90, Computer Science Department, Columbia University, New York, NY 10027, 1990.

[18] J. S. Vitter and P. Krishnan, "Optimal prefetching via data compression," *Journal of the ACM*, vol. 43, pp. 771–93, Sept. 1996.

[19] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy, "Organization-based analysis of web-object sharing and caching," in *Proceedings of the Second Usenix Symposium on Internet Technologies and Systems*, (Boulder, CO), pp. 25–36, USENIX Association, Oct. 1999.

[20] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative Web proxy caching," in *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, (Charleston, SC), pp. 16–31, Dec. 1999.