In Proceedings of the Second Intl. Workshop on Services in Distributed and Networked Environments (SDNE '95)

Application-Level Document Caching in the Internet*

Azer Bestavros Robert L. Carter Mark E. Crovella
Carlos R. Cunha Abdelsalam Heddaya Sulaiman A. Mirdad

Computer Science Department
Boston University
111 Cummington St, Boston, MA 02215

{best,carter,crovella,carro,heddaya,mirdad}@cs.bu.edu

Abstract

With the increasing demand for document transfer services such as the World Wide Web comes a need for better resource management to reduce the latency of documents in these systems. To address this need, we analyze the potential for document caching at the application level in document transfer services. We have collected traces of actual executions of Mosaic, reflecting over half a million user requests for WWW documents. Using those traces, we study the tradeoffs between caching at three levels in the system, and the potential for use of application-level information in the caching system. Our traces show that while a high hit rate in terms of URLs is achievable, a much lower hit rate is possible in terms of bytes, because most profitably-cached documents are small. We consider the performance of caching when applied at the level of individual user sessions, at the level of individual hosts, and at the level of a collection of hosts on a single LAN. We show that the performance gain achievable by caching at the session level (which is straightforward to implement) is nearly all of that achievable at the LAN level (where caching is more difficult to implement). However, when resource requirements are considered, LAN level caching becomes much more desirable, since it can achieve a given level of caching performance using a much smaller amount of cache space. Finally, we consider the use of organizational boundary information as an example of the potential for use of application-level information in caching. Our results suggest that distinguishing between documents produced locally and those produced remotely can provide useful leverage in designing caching policies, because of differences in the potential for sharing these two document types among multiple users.

1 Introduction

Some of the most popular services currently provided by the Internet are the distributed information systems such as the World Wide Web (WWW), the Anonymous FTP transfer system, the Wide Area Information System (WAIS), and the Gopher system. These services are characterized by a many-to-many pattern of file transfer — most hosts in the system are potentially capable of serving files as well as requesting them. We refer to these systems as *document transfer* systems and to the files involved as *documents* since each file has essentially been electronically "published."

An increasingly large fraction of available bandwidth on the Internet is being used to transfer documents [9]. Strategies for reducing the latency of document access, the network bandwidth demand of document transfers, and the demand on document servers are becoming increasingly important. Techniques that could reduce document latency, network bandwidth demand, and server demand include data caching and replication. However, in contrast to most distributed file systems, document transfer services usually incorporate simple caching strategies, if any, and do not typically provide location transparency.

While techniques based on distributed file systems could be used to improve significantly the performance of document transfer systems, there are a number of advantages to considering caching and replication at the application level, rather than at the filesystem level. First, application-level caching does not require all users to agree on a common filesystem; it enables heterogeneous systems to participate easily. Second, and more important, application-level caching allows cache strategies to make use of the higher semantic content available at the application level to exploit such information as document type, user profile, user past history, document content, and organizational boundaries.

This paper describes initial investigations into application-level strategies for document caching and replication on wide area networks. While we are in general concerned with all three aspects of the problem (document latency, network demand, and server demand) we focus in this paper on minimizing document latency as our primary goal. As a result, we concentrate on caching strategies rather than document replication, which is mainly a technique for reducing server load.

We employed a trace-driven simulation approach to studying the document caching problem. First, we collected logs of users accessing the World Wide Web. We instrumented a version of NCSA Mosaic [6] to keep

*This work has been partially supported by NSF (grant CCR-9308344).

a record of all documents (named by their Uniform Resource Locators — URLs) accessed by the user during an execution of Mosaic. (We refer to each execution of Mosaic as a *session*, and we call the log of each session a *trace*.) The results in this paper are based on 4,700 traces.

Next, we used the traces as input to an event-driven simulation that determined how various caching strategies and cache sizes affected the performance of the system. The simulation outputs a set of statistics that describes the effectiveness of caching in terms of bytes transferred and document latency.

This paper discusses cache policies that operate at three levels: 1) the *session* level, in which caches for separate sessions are managed independently; 2) the *host* level, in which caches for separate hosts are managed independently; and 3) the *LAN* level, in which caches for separate LANs are managed independently.

Session caches are similar to the policies used in current versions of NCSA Mosaic. Host caches consist of a single host's buffers allocated to document caching that persist across invocations of the client. Host caches could be implemented by a local server, or by periodically synchronizing each application's memory-based cache with a disk buffer. LAN caches consist of a cache managed by the clients on a single LAN, as in [4]. LAN caches require cooperation among the participating clients; host and session caches do not.

Our work is unique in a number of ways. First, we base it on the large amount of user trace data we have collected. Second, we consider caching policies that can be implemented without client cooperation as well as policies that require client cooperation. Finally, we use application-level information in analyzing our trace data and in formulating cache policies.

Our results show that caching strategies that are nearly as effective as a cooperative strategy can be implemented at the application level without cooperation; in fact, session level strategies yield nearly all the gains of host level and LAN level strategies. In addition, while session level caching is nearly as effective as the others, it consumes much more system resources. For a given level of performance, less system resources are consumed by host level caching, and even less are consumed by LAN level caching. Thus, if a fixed amount of system resources is to be allocated to caching, they are best allocated to LAN level caching.

Finally, our data suggest that the use of application-level information can significantly improve some aspects of system performance; in particular, identifying documents that originate outside of the local organizational boundary (in our case, the Boston University community) is useful in understanding and tuning cache performance. We discuss cache policies that favor or discourage retention of local documents. We show that documents originating outside the local organization show markedly different sharing patterns from those that are served locally.

The remainder of the paper consists of: first, a description of our trace data and the collection process; next, the results of our simulations for various caching policies using that data; next, a comparison of our work with related research; and, finally, our conclusions.

Sessions	4,700
Users	591
Documents Requested	575,775
Unique Documents Requested	46,830
Bytes Requested	2713 MB
Unique Bytes Requested	1088 MB

Table 1: Summary Statistics of Trace Data

2 Reference Patterns

2.1 Data Collection Methods

Prior studies of WWW traffic have been based on logs from proxies [7, 16], or logs from the HTTP server daemon [13]. Our study required knowledge of individual user's access patterns, and we did not wish our data to be influenced by the caching behavior built in to the client application (Mosaic). For these reasons, we instrumented Mosaic directly and captured logs of *all* accesses performed by the user. Thus the entries in our traces consist of each URL requested by the user, whether it was served from Mosaic's cache or from the network.

Each entry in a trace consists of the client host name, the time stamp when the request was made, the URL, the size of the document (including the overhead of the protocol) and the round trip retrieval time.

The computing environment considered in this study consists of 37 SparcStation 2 workstations; these workstations comprised the set of client hosts. The LANs used are part of a larger, subnetted domain (`bu.edu`) which consists of many hundreds of workstations, many of which act as WWW servers. Five of the workstations support graduate students in BU's Computer Science Department, and the other 32 support a general population of computer science students.

The traces used in this study were collected over a period of 3.5 months, from middle of November 1994 to end of February 1995. In this period a total of 4,700 sessions were traced, representing 591 different users. User names were mapped to numeric IDs so that researchers performing data analysis were not aware of user identities.

2.2 Summary of Data Collected

Descriptive statistics summarizing our data are given in Table 1. In the table, *Documents Requested* means the total number of URLs whose contents were retrieved for the user, either from the network or from Mosaic's cache. Note that many of the URLs may refer to small items, such as icon bitmaps. *Unique Documents Requested* is the number of unique URLs in the traces. *Bytes Requested* means the sum over all document requests of the file sizes requested; *Unique Bytes Requested* represents the same sum over only the unique URLs.

The table shows that, on average, users engaged in multiple sessions (8 sessions per user), and that the average number of documents requested per session was high (122 documents per session). It also shows that there is a high potential payoff for document caching, since the difference between Bytes Requested and Unique Bytes Requested is large. This difference represents the best-case number of bytes that could be obtained from a demand-driven cache without prefetching.

The distribution of the documents and references by size is shown in Figure 1.[1] In the figure, the complete distributions are shown in the left-hand histograms; the right-hand histograms show the distributions for small documents only (less than 6.4 k bytes). The upper histograms plot the distribution of documents, while the lower histograms plot the distribution of references to documents.

Previous studies of user filesystem requests have shown a strong preference for small files [11]. Our data shows a similar pattern; the most popular document size is between 256 and 768 bytes.

The strong preference for small files is somewhat surprising due to the potential for multimedia content of WWW documents and the large amounts of data needed to transfer images, video, and sound. However a number of factors may tend to increase the proportion of small file sizes. First, many of the images (small icon bitmaps) are actually fairly small in size; Mosaic typically caches these images so users are not often aware of how many small images are employed in a WWW document. Second, users may tend to interrupt document transfers that take too long; our data does not include any documents whose transfer was interrupted. Finally, we feel that these data indicate that despite the great potential for large, multimedia document transfers, such transfers do not as of yet constitute the predominant use of the WWW.

The differences between the upper and lower plots in Figure 1 indicate that, while most documents are small, the effect of user reference patterns is to *increase* the preference for small documents. That is, smaller documents are more likely to be requested than are large documents. In fact, there is a significant correlation between the size of a document and the number of times it is requested. Figure 2 shows a plot of the average number of requests to a document given its size. The solid line is the plot of a statistically significant (99.9% level) inversely proportional relationship between the number of requests to a document and its size. However the data is quite noisy and the least squares estimator (references $= 9.67\ \text{size}^{-0.33}$) only explains a small part of the total variation ($R^2 = 0.14$).

[1] These histograms are clipped; the maximum observed for the unique documents was 32,262, in the 0 to 10 K slot; the maximum total number of accesses was 536,826, in the 0 to 10 K slot. The counts for the 128-byte scale were 2,716 in the 256-384 slot and 2,460 in the 384-512 slots for the unique documents, and 112,119 in the 128-256 slot, 90,461 in the 256-384 slot, 49,372 in the 384-512 slot, and 56,051 in the 640-768 slot.

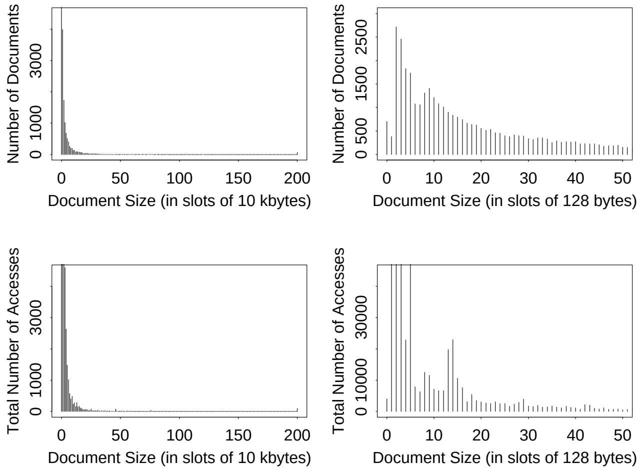

Figure 1: Distribution of User Requests by File Size; Unique (Top) and Total (Bottom).

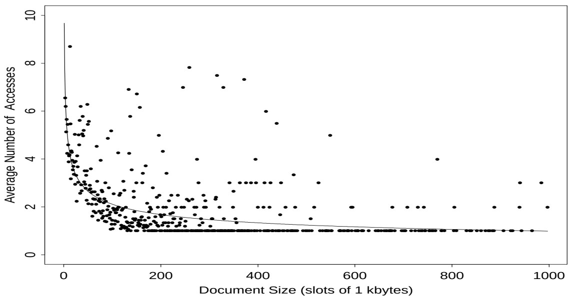

Figure 2: Distribution of Average Number of Requests by File Size.

3 Caching Effectiveness

3.1 Experimental Setup and Metrics

In this section we present results of trace-driven simulation of various caching schemes. The traces used consist of all references that a user makes during a session with Mosaic and were described in Section 2.2. The individual session traces are combined to allow for the simulation of the 3 granularities of caching: session, host and LAN.

We show results for both finite and infinite cache sizes. The measurements obtained for an infinite cache provide an upper bound on the effectiveness of caching at each granularity. We also use the traces to drive simulations of finite caching with variable size caches and least frequently used (LFU) replacement. Since we are dealing only with "published" documents we do not consider invalidation of cache entries. In the analysis of our results it is also useful to distinguish between documents that are local and those which are stored at remote locations. We consider a document to be local if was served from a host within the Boston

University organization, which can be detected based on the server's name.

For each caching granularity and cache size, we measure three quantities: hit rate, byte-hit rate and latency savings.

The first measure is H, the hit rate. This is defined as the ratio of the number of references satisfied by the cache to the total number of references.

Hit rate is a good measure only if the documents are of equal sizes. However, our traces reference documents of widely varying sizes. In order to get a better idea of the effectiveness of the caching schemes we weight each reference by the document size to calculate the byte-hit rate B.

Comparisons based on B alone presume that all bytes cost the same. Just as there is a wide variation in document size there are varying distances from which documents must be fetched introducing variation in delay. A measure of the fraction of worst-case latency saved by caching, which captures the variation in distance, can be defined by weighting each reference by the round-trip delay time for the document. We denote this measure of latency savings by C.

3.2 Simulation Results

3.2.1 Infinite Caches

For the case of an infinite cache we define H, B, and C in terms of the following variables. Each trace entry represents a reference to a document i from the set of all documents $\{1, 2, \ldots, n\}$. For each document i we denote the size of the document by s_i, the round-trip delay for document retrieval by d_i and the number of references to the document by r_i.

Given an infinite cache all but the first reference to any document will be cache hits:

$$H = \frac{\sum_{i=1}^{n}(r_i - 1)}{\sum_{i=1}^{n} r_i}.$$

Calculation of the byte-hit rate weights each document reference r_i by the document size s_i:

$$B = \frac{\sum_{i=1}^{n} s_i(r_i - 1)}{\sum_{i=1}^{n} s_i r_i}.$$

By weighting each reference r_i by the round-trip delay d_i we can define the fraction of worst-case latency saved by caching as:

$$C = \frac{\sum_{i=1}^{n} d_i(r_i - 1)}{\sum_{i=1}^{n} d_i r_i}.$$

The percentages H, B and C can be calculated directly for the single cache at the LAN. For host and session granularities the values are averaged over all caches.

In Table 2 we present the results we have obtained by running the trace-driven simulation on data compiled from all 4,700 sessions. Each row gives values of H, B and C as a percentage for one of the cache granularities studied. Table 3 gives values when only remote references are included in the simulation and

Cache Granularity	H	B	C
session	79.10	37.65	63.64
host	86.21	46.06	68.36
LAN	91.97	60.18	77.17

Table 2: All references (575,775 references).

Cache Granularity	H	B	C
session	82.50	36.83	63.44
host	84.95	41.31	66.77
LAN	90.35	52.73	75.67

Table 3: Remote references only (452,864 references).

Cache Granularity	H	B	C
session	66.58	40.92	66.36
host	90.85	64.84	89.37
LAN	97.94	90.69	97.16

Table 4: Local references only (122,911 references).

Table 4 gives values when only local references are considered. These tables show a steady increase in hit rate as the granularity of the cache is increased. This is to be expected as repeated references are increasingly captured when more references are passed through an infinite cache.

Table 2 shows that a relatively high hit rate (H = 91%) is possible for an infinite cache at the LAN level. However, the corresponding byte-hit rate is only 60%. This is in fact the best possible byte-hit rate since Table 1 shows that of 2713 MB requested, 1088 MB (40%) are unique. These data reflect the fact that the large documents in our traces are not as profitably cached — they are referenced relatively few times each. This agrees with the trend shown in Figure 2.

Turning to our measure of latency savings (C), we see a relatively small effect as cache granularity is increased. That is, for the combined data in Table 2 it appears that the advantage of a LAN cache is that it caches a greater percentage of the bytes referenced but does not save that much time compared to session or host caches. This is due to the extra sharing being composed primarily of local (inexpensive) documents as seen from Table 4. For local documents, the byte-hit rate increases from 41% at the session granularity to 91% at the LAN. Thus, the local proportion of the shared bytes increases as the cache granularity is increased. In addition, the cost of retrieving remote documents is so much higher than for local ones (the ratio of the total cost for remote document retrieval to the total cost for local document retrieval is approximately 18 for our traces) that the C values in Table 2 reflect the diminishing returns of caching remote documents at the LAN granularity. For B values, remote documents contribute 4 times as many bytes as lo-

Metric	Ideal	Mosaic
H	79.10	77.34
B	37.65	32.56
C	63.64	61.49

Table 5: Comparison of ideal session hit rates and Mosaic's hit rates.

cal documents and so the combined figures also more closely resemble those of the remote documents.

Since the results presented in this section were obtained using infinite caches, it might be the case that in practice actual hit rates would not approach these ideal rates. To test this possibility we measured the hit rates obtained by Mosaic itself, which demonstrates a real session caching algorithm operating with limited cache space. The results are shown in Table 5. The table shows the session-level hit rates, byte-hit rates, and latency reduction rates obtained in our ideal simulation and obtained by Mosaic. These data show that the hit rates obtained in our simulation are very close to those obtainable in practice.

3.2.2 Application-Level Caching Policies

Tables 3 and 4 also imply that there is a higher degree of sharing among local documents than remote documents. Notice that the hit rate and the byte-hit rate both increase for local documents as the cache granularity increases. This relationship also holds, though to a much smaller degree, for remote documents. For byte-hit rate B, an additional 16% of the requested remote bytes are satisfied by a LAN cache than a session cache. But, an additional 50% of the requested local bytes are satisfied by a LAN cache versus a session cache.

Turning to C, session caching of remote documents saves 63% of the total time that would have been spent retrieving documents if no caching were done. Caching at the LAN increases the savings rate to 75%. However, for local documents, LAN caching provides a 97% savings versus no caching while session caching provides only a 66% savings.

More detailed statistics summarizing the differences between local and remote documents are given in Table 6. The first four lines in the table are breakdowns of the corresponding lines in Table 1. The lower three lines are derived from the first four.

Table 6 shows that local and remote documents do not differ in size on average. However, these data suggest that accesses to local and remote documents exhibit significantly different sharing patterns. To explore the utility of distinguishing between local and remote documents, we studied the performance of caching policies that used document location information.

We first characterized the differences in sharing patterns between remote and local documents by studying caching policies that cached documents of only one of the two types. The Remote policy caches only

	Local	Remote
Documents Requested	122,911	452,864
Unique Documents Requested	2,579	44,251
Bytes Requested	551 MB	2163 MB
Unique Bytes Requested	51 MB	1037 MB
Average Document Size Requested	4,696	5,008
Average Requests Per Document	47.7	10.2
Average Requests Per Byte	10.6	2.1

Table 6: Summary Statistics of Local and Remote Documents

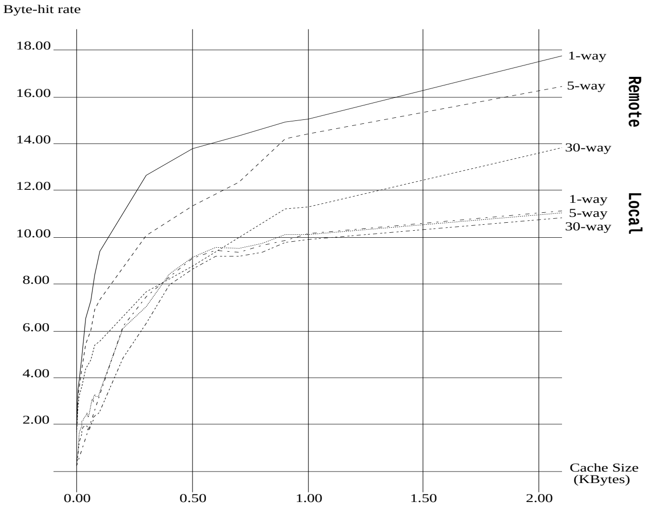

Figure 3: Multi-way byte-hit rate vs LAN cache size

remote documents; the Local policy caches only local documents. To study sharing patterns, we simulated multiple users sharing a single cache (essentially a LAN cache). To do this for N users, we divided the traces into N sets; in each set, traces were concatenated, and the sets were then interleaved. This strategy ensured a constant level of multi-way sharing of the cache.[2]

Figure 3 shows the relationship between the cache size and the achievable byte-hit rate, for different levels of multi-way sharing (namely for 1, 5, and 30 users). There are two sets of curves. The first illustrates the behavior of the Remote policy, whereas the second illustrates the behavior of the Local policy. Figure 3 shows that users are more likely to share local documents than remote ones. In order to quantify this level of sharing, we define the *Sharing Index* (SI) as a function of both the number of users N and a fixed byte-hit rate $B = \alpha$.

$$SI(\alpha, N) = 1 - \frac{1}{N-1} \cdot \frac{\mathcal{L}(\alpha, N) - \mathcal{L}(\alpha, 1)}{\mathcal{L}(\alpha, 1)}$$

[2]Due to the repeated numbers of simulations required these results and those in the next section were obtained from a subset of our total reference data.

In the above definition, $\mathcal{L}(\alpha, N)$ is the size of the LAN cache necessary to achieve a byte-hit rate of α for an N-way-shared cache. $\mathcal{L}(\alpha, N)$ can be obtained from Figure 3. A *Sharing Index* of 1 means that increasing the number of users does not necessitate increasing the size of the cache to keep the byte-hit rate at a constant level. A *Sharing Index* of 0 means that increasing the number of users will necessitate increasing the size of the cache proportionally to keep the byte-hit rate at a constant level. Tables 7 and 8 show various values of $SI(\alpha, N)$ for the Local and Remote policies, respectively.

α	2	5	15	30	50
0.06	1.00	1.00	0.99	0.98	0.99
0.08	0.98	1.00	1.00	1.00	0.99
0.10	0.98	0.99	1.00	0.99	0.99
0.12	0.98	1.00	1.00	n/a	n/a

Table 7: *Sharing Index* for Local Policy

α	2	5	15	30	50
0.06	0.89	0.82	0.89	0.90	0.90
0.08	0.92	0.75	0.85	0.87	0.88
0.10	0.67	0.71	0.84	0.86	0.89
0.12	0.60	0.64	0.87	0.86	0.88

Table 8: *Sharing Index* for Remote Policy

The significant difference in sharing patterns demonstrated in Tables 7 and 8 suggests that caching policies might profitably exploit the distinction between local and remote documents. To quantify the potential benefits from this approach, we define the *Cache Expansion Index* (CEI) for a particular level of byte-hit rate α and a particular number of users N to be the ratio of $\mathcal{L}(\alpha, N)$ to $\mathcal{L}(\alpha, 1)$. This is the expected "expansion" in cache size that is necessary to maintain the byte-hit rate at a constant α, while the number of users sharing the cache increases from 1 to N. A larger CEI signifies a smaller level of sharing, whereas a smaller CEI signifies a larger level of sharing. Figure 4 illustrates the value of CEI for various numbers of users and for various byte-hit rates. Again, we notice that, due to the higher level of N-way sharing of local documents (compared to remote documents), the Local policy exhibits a small CEI, compared to the Remote policy. Figure 4 suggests that the CEI for both the Local and Remote policies is linear in N. The constant for the Local policy is very small (0.03)[3], whereas the constant for the Remote policy is much larger (0.12). In both cases, there is no indication of the CEI reaching a plateau (at least for the number of users we considered in our simulations). Figure 4 also shows no particular correlation between the CEI and the desired hit rate level.

[3] This means that a 3% increase in cache size is necessary to maintain the same hit rate with one additional user

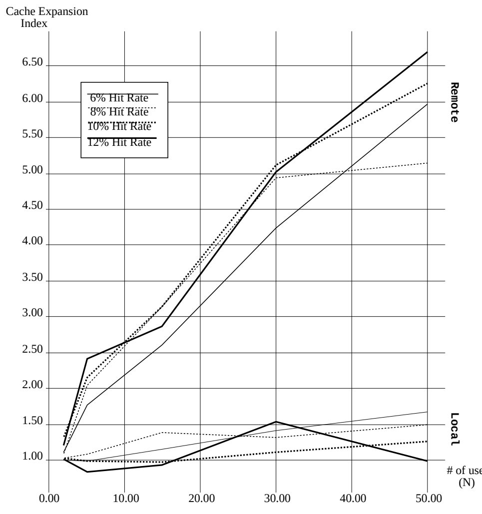

Figure 4: Cache Expansion Index

3.2.3 Cache Space Utilization

In order to explore resource utilization trade-offs we ran simulations for all three caching granularities with various limited cache sizes and LFU replacement.

The graphs in Figures 5 and 6 show the hit rate and byte-hit rate respectively for the three caching granularities with cache sizes ranging from 10KB to 2GB. For session and host granularities the total number of cache bytes is equally divided among five hosts since our traces were collected on five workstations. Since the total number of unique bytes accessed by all references is 157MB, ideal caching occurs at or above this cache size for the LAN granularity. That is B approaches 48%, the value found for an infinite cache of LAN granularity.

Figure 5 shows the clear superiority of cooperative LAN caching regardless of cache size as measured by hit rate. Session caching gives the smallest hit rate at all granularities while host caching equals the performance of session caching at smaller cache sizes and rises to within three percentage points of LAN caching performance at higher cache sizes.

Once again we focus on the more informative measure of byte-hit rate. Figure 6 gives the byte-hit rates over the range of cache sizes studied. Here again the benefit of LAN caching is clearly evident.

Session caching actually outperforms Host caching at cache sizes less than 4MB (800 KB/host) due to interference between users sharing a cache on a workstation. Above this level the cache is big enough that documents can remain in the cache long enough for sharing to occur and be reflected in the byte-hit rate.

At a cache size of 9MB, LAN caching achieves a 37% byte-hit rate. To get the same benefit from host caching would require approximately 90MB or 10 times as much memory. Session caching never achieves this rate, even with infinite caches for each client session. Viewed another way, given a total cache size of 250KB, deployment of five 50KB host caches gives

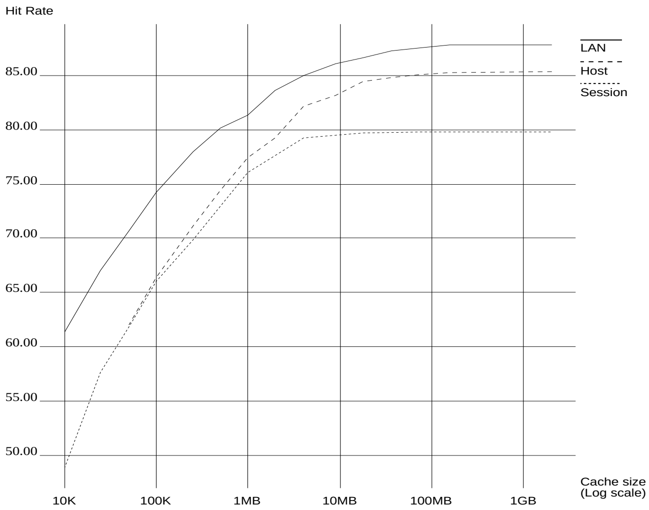

Figure 5: Hit Rate Vs. Cache Size

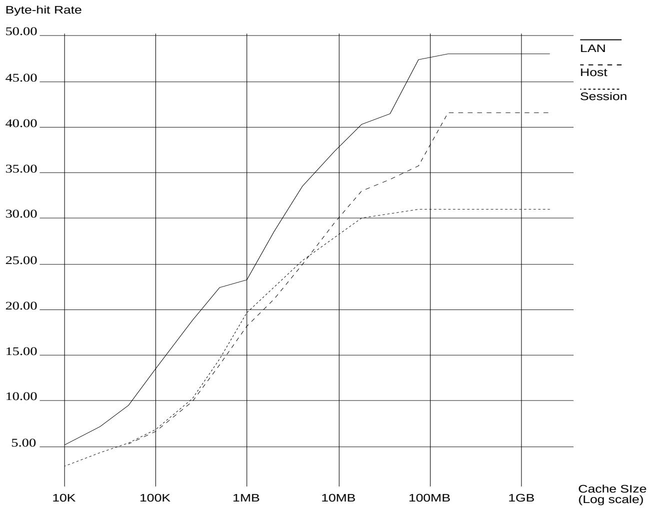

Figure 6: Byte-hit Rate Vs. Cache Size

a byte-hit rate of 10% while cooperative sharing of a 250KB LAN cache achieves a rate of 19%. And this relative difference remains consistently robust: for a size of 75MB, session caching reaches its peak byte-hit rate of 31% while the same size cache shared by session on each host improves the byte-hit rate to 36% and a cooperatively shared cache at the LAN provides further improvement to 48%, the performance of an infinite cache.

4 Related Work

A great deal of research on caching and replication in distributed file systems has been conducted previously (*e.g.,* [14, 15]). In such research the main goal has been to improve the overall performance of the system. In contrast to these studies, the material presented in this paper focuses mainly on reducing response time through caching at the application level, rather than caching at the file system level.

Danzig et al. [5] propose a hierarchical caching system that caches files at Core Nodal Switching Subsystems within the NSFNET. The main goal of their research is to reduce the bandwidth used by the system; their study shows that the NSFNET backbone traffic can be reduced by as much as 21%. Such schemes do not make use of application-level information. In our study significant differences were observed between documents identified at the application to be local from those identified as remote. Although we did not report on network bandwidth reduction, we have performed preliminary studies that show a significant potential for network bandwidth reduction by application-level caching.

The reduction of network traffic due to intelligent data placement and replication is also studied in [1]. They present a distributed dynamic replication scheme, which uses a finite state automaton-based technique to learn file access patterns. In contrast, our focus is on data caching rather than replication and placement techniques.

In [12] the authors approximate an optimal caching schedule based on fixed network and storage costs. This schedule indicates where and when a file should be cached. In the worst-case the algorithm produces a schedule that is no worse than twice the optimal one. Their theoretical work is an off-line algorithm in comparison to the work presented here in which trace data from user accesses is used to study on-line algorithms.

Muntz and Honeyman [10] performed simulations on a two level caching system, intermediate servers and clients. Although the intermediate cache reduces both the peak load at upstream servers and network load, the average hit rate at the intermediate cache is not significant. We further extend their work showing the improvements in latency due to application caching.

In [3] Blaze presents a dynamic hierarchical file system. Each client can service requests issued by other clients from the local disk cache. The focus of that work was to reduce server load. Here we focus on reducing latency through the use of application-level caching; we plan future work to explore the potential for reduction in server load possible via application-level caching.

Performance benefits of cooperative caching have been studied in [4]. Through the use of trace driven simulations a range of caching algorithms were studied. Their results show that an improvement of 73% in file read performance can be achieved. Our work extends this study by comparing cooperative caching with caching at the session and host level, by considering the the resource demands of all three approaches, and by attempting to define the types of documents that should be cached at the different levels.

A different approach to reducing server load and latency for distributed information systems, such at the WWW, is based on the popularity-based dissemination of information from servers to proxies, which are *closer* to clients. There are three problems to be tackled for such an approach, namely *what, how far,* and *in which direction(s)* to disseminate. The work in [2] addresses the first two aspects, whereas the work in [8] investigates the third. Supply-based dissemination of information is complementary to demand-driven caching; the former aims primarily at reducing traffic and balancing load (through replication), whereas the latter aims primarily at reducing the service time (through caching).

5 Conclusion

In this paper we have presented results of a study tracing user accesses to the World Wide Web, and the results of simulations employing those traces to study caching algorithms for document transfer systems.

Our trace data shows that a high hit rate is possible in terms of document accesses; however, the fraction of bytes that could be found even in an infinite cache is much lower. This occurs because a large fraction of documents requested are small, despite the large file sizes needed for multimedia. Thus, effectively eliminating latency to large, infrequently accessed documents is not well addressed in this work.

Given the relatively low upper bound for byte-hit rate, we show that session level strategies (the easiest to implement) can achieve much of the performance benefit of LAN level strategies (which require inter-client cooperation). This is shown by the fact that the maximum possible byte-hit rate results in a document latency reduction of 77% for the LAN level strategy, compared to 64% achievable using a session level strategy.

When the resource requirements of the three caching levels are considered, the LAN level becomes much more desirable. LAN level caching consistently requires less cache space to achieve a given byte-hit rate when compared to host and session level caching. In a wide range (up to 30% byte hit rate) LAN level caching can perform as well as session or host level caching in approximately one-fourth the space.

Finally, we consider the recognition of organizational boundaries as an example of the use of application-level information in the caching process. We show that local documents experience a higher degree of sharing among clients on our LAN than do remote documents. It is important to note that while organizational boundaries might be deduced or known at the level of the filesystem, the proper response of the caching algorithm might vary depending on the application.

While this study yields a number of insights into application-level document caching, it also suggests a number of areas of future work. We are beginning a longer term project to study many of these issues.

References

[1] Swarup Acharya and Stanley B. Zdonik. An efficient scheme for dynamic data replication. Technical Report CS-93-43, Brown University, Providence, Rhode Island 02912, September 1993.

[2] Azer Bestavros. Demand-based document dissemination for the World Wide Web. Technical Report TR-95-003, Boston U., CS Dept, Boston, MA 02215, Feb. 1995.

[3] Matthew Addison Blaze. *Caching in Large Scale Distributed File Systems.* PhD thesis, Princeton University, 1993.

[4] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proc. of the 1st Symposium on Operating Systems Design and Implementation*, pages 267–280, Nov. 1994.

[5] Peter Danzig, Richard Hall, and Michael Schwartz. A case for caching file objects inside internetworks. Technical Report CU-CS-642-93, University of Colorado at Boulder, Boulder, Colorado 80309-430, March 1993.

[6] National Center for Supercomputing Applications. Mosaic software and documentation.

[7] Steven Glassman. A Caching Relay for the World Wide Web. In *First International Conference on the World-Wide Web*, CERN, Geneva (Switzerland), May 1994. Elsevier Science.

[8] James Gwertzman and Margo Seltzer. The case for geographical push-caching. Technical Report HU TR-34-94 (excerpt), Harvard U., DAS, Cambridge, MA 02138, 1994.

[9] Merit Network Incorporated. NSFNet performance statistics.

[10] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems or your cache ain't nuthing but trash. In *Proceedings of the Winter 1992 USENIX*, pages 305–313, January 1992.

[11] John K. Ousterhout, Herve Da Costa, David Harrison, John A. Kunze, Michael Kupfer, and James G. Thompson. A trace-driven analysis of the UNIX 4.2BSD file system. Technical Report CSD-85-230, Dept. of Computer Science, University of California at Berkeley, 1985.

[12] Christos H. Papadimitriou, Srinivas Ramanathan, and P. Venkat Rangan. Information caching for delivery of personalized video programs on home entertainment channels. In *Proceedings of the International Confrence on Multimedia Computing and Systems*, pages 214–223, May 1994.

[13] James E. Pitkow and Margaret M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns. In *Electronic Proc. of the 2nd WWW Conference*, 1994.

[14] R. Sandber, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network file system. In *Proc. USENIX Summer Conference*, 1985.

[15] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Streere. Coda: A highly available file system for distributed workstation environments. *IEEE Transactions on Computers*, 39(4), April 1990.

[16] Jeff Sedayao. "Mosaic Will Kill My Network!" – Studying Network Traffic Patterns of Mosaic Use. In *Electronic Proc. of the 2nd WWW Conference*, Chicago, Illinois, October 1994.