In Proceedings of CIKM'95: The 4th ACM International Conference on Information and Knowledge Management, Baltimore, MD, Nov 1995.

Using Speculation to Reduce Server Load and Service Time on the WWW*

Azer Bestavros
(best@cs.bu.edu)

Computer Science Department
Boston University
Boston, MA 02215

Abstract

Speculative service implies that a client's request for a document is serviced by sending, in addition to the document requested, a number of other documents (or pointers thereto) that the server *speculates* will be requested by the client in the near future. This speculation is based on statistical information that the server maintains for each document it serves. The notion of speculative service is analogous to prefetching, which is used to improve cache performance in distributed/parallel shared memory systems, with the exception that servers (not clients) control when and what to prefetch. Using extensive trace simulations based on the logs of our departmental HTTP server `http://cs-www.bu.edu`, we show that *both* server load and service time could be reduced considerably, if *speculative service* is used. This is *above and beyond* what is currently achievable using client-side caching [3] and server-side dissemination [2]. We identify a number of parameters that could be used to fine-tune the level of server speculation and we discuss variations of speculative service that involve cooperation with clients.

1 Introduction

Current protocols for accessing distributed information systems do not scale, partly due to the inability of servers to cope with the increasing volume of client requests. Perhaps the best "living" proof of the seriousness of this problem is the fate of many multimedia information servers on the Internet: they are unreacheable as soon as they become popular. In this paper, we propose the use of speculative service protocols, whereby a server responds to a clients' request by sending, in addition to the data (documents) requested, a number of other documents that it *speculates* will be requested by that client in the near future. We use the World Wide Web (WWW) as the underlying distributed computing resource to be managed. First, the WWW offers an unmatched opportunity to inspect a wide range of distributed object types, structures, and sizes. Second, the WWW is fully deployed in thousands of institutions worldwide, which gives us an unparalleled opportunity to apply our findings to an already-existing real-world application.

Server speculation is based on statistical information that the server maintains for each document it serves. We used the logs of our departmental `http://cs-www.bu.edu` server to drive preliminary trace simulations to evaluate the benefits that could be gained from speculative service. Our results show two possible benefits. On the one hand, our results demonstrate that appropriate speculation could be used to reduce both server load and service time, *without* increasing the required network bandwidth. On the other hand, our results demonstrate that if agressive speculation is adopted network bandwidth could be *traded for* considerable improvements in service time. This could be desirable for distributed real-time applications.

The remainder of this paper is organized as follows. In Section 2, we introduce the notion of *document access interdependencies*, which encompasses both embedding dependencies and traversal dependencies. In Section 3, we present our simulation results. We start by describing our simulation model and baseline parameters and proceed to present a host of experiment results that allowed us to identify a number of parameters (and protocol variations) that could be used to fine-tune the optimum level of speculation to be performed by the server. In section 3, we discuss related research work as well as on-going and future research work of ours. Our conclusion is in Section 4.

2 Document Access Interdependencies

Given that a client has requested a particular document (say $\mathcal{D}_i$), what is the likelihood that it will request another document (say $\mathcal{D}_j$) in the *near* future? In some instances, the answer to this question is evident. For example if $\mathcal{D}_j$ is embedded in $\mathcal{D}_i$, then the probability that it will be requested given that $\mathcal{D}_i$ has been requested is *always* 1. In general, the answer to this question is not straightforward and requires a thorough analysis of the client access patterns.

Let $p[i,j]$ denote the conditional probability that document $\mathcal{D}_j$ will be requested, within a limited window of time T_w, given that document $\mathcal{D}_i$ has been already requested. In other words, if $\mathcal{D}_i$ is requested at time t, then with a probability $p[i,j]$, $\mathcal{D}_j$ will be requested within the interval $[t, t+T_w]$, for some constant T_w. Let P denote the square matrix representing $p[i,j]$, for all possible documents $0 \geq i, j \leq N$. We define P^* to be the closure of P, where $P^* = P^N$. Obviously, $p^*[i,j]$ denotes the probability that there will be a sequence of requests starting with document $\mathcal{D}_i$ and ending with document $\mathcal{D}_j$, in which every request is separated by at most T_w units of time from the previous request in that sequence.

By analyzing the logs of the `cs-www.bu.edu` HTTP server for the month of January 1995 (more than 50,000 accesses)

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

we computed the function P and its closure P^*. Figures 1 and 2 show histograms of the number of document pairs ($\mathcal{D}_i$ and $\mathcal{D}_j$) against various ranges of $p^[i, j]$ and $p^*[i, j]$, respectively, assuming that the value of T_w is set at 5 seconds. Figure 1 can be characterized as having a series of peaks around values of $P = \frac{1}{k}, i = 1, 2, 3, \ldots$. Given that the number of links (anchors) for any document is an integer, Figure 1 suggests that for a large number of documents, the probability of following these anchors is equal.

We distinguish between two types of document dependencies, namely *embedding* and *traversal* dependencies. An embedding dependency occurs when a document $\mathcal{D}_j$ is *always* requested when another document $\mathcal{D}_i$ is requested. A traversal dependency occurs when a document $\mathcal{D}_j$ is *sometimes* requested when another document $\mathcal{D}_i$ is requested. In Figure 1, embedding dependencies are responsible for the peak at the rightmost part of the graph.

3 Simulation Results

Using the function P and its closure P^*, we ran a number of trace simulations. In this section we present the results of our experiments. We start by describing our simulation model and parameters. Next, we present our simulation results for a baseline model, which is used as a reference point for other experiments. Next, we discuss and evaluate the performance of speculative service under various assumptions and refinements.

System Model: In our simulations we assumed that, when a request for a document $\mathcal{D}_i$ is received, the server responds by sending to the client $\mathcal{D}_i$ as well as any other document $\mathcal{D}_j$ that satisfies an inequality based on the function P and P^*. This inequality determines the particular `Policy` employed. An example policy would be simply to service a document $\mathcal{D}_j$ along with a requested document $\mathcal{D}_i$ if $p^*[i, j] \geq T_p$, for some threshold probability $0 < T_p \leq 1$. A document $\mathcal{D}_j$ is never speculatively serviced if its size is greater than `MaxSize`. This provision was added to avoid situations in which huge documents would be speculatively serviced in vain.

In our simulations if two requests from the *same* client were issued within `StrideTimout` seconds of each other, then these requests are assumed to be *dependent*, and thus significant in calculating the embedding/traversal dependencies captured by the functions P and P^*. Otherwise, they are assumed to be *independent* and thus not significant in the computation of P and P^*. We define a *traversal stride* to be a sequence of requests (from the same client) where the time between successive requests is less than `StrideTimeout` seconds. By controlling the value of `StrideTimeout`, we can restrict or loosen up our definition of document dependency. In particular, setting `StrideTimeout` to a very small value will restrict the definition of document dependency to embedding dependencies, whereas setting it to a larger value will loosen the definition of document dependency to include traversal dependencies as well.

In our simulations, we assumed that clients use a caching policy, whereby a document is cached after it is first retrieved (as a result of a client-initiated request or as a result of a server-initiated speculative service). This document remains in the cache until it is purged at the end of the session. We define a *session stride* to be a sequence of requests (from the same client) where the time between successive requests is less than `SessionTimeout` seconds. By controlling the value of `SessionTimeout`, we can emulate various caching policies. In particular, setting `SessionTimeout` to ∞ could be used to emulate a client with an infinite-size multi-session cache (*e.g.*, the LAN cache proposed in [3]). Setting `SessionTimeout` to (say) 60 minutes could be used to emulate a client with an infinite-size single-session cache. Setting `SessionTimeout` to 0 could be used to emulate a client with no cache.

The cost model we adopted in our simulations assumes a symmetric network, where the cost of communicating one byte between any server and any client is `CommCost`. In comparison, the cost of servicing one request is `ServCost`. These two parameters allow us to weight the reduction in a server's load against the increase in network traffic as a result of speculative service.

The results of our simulations are summarized using four metrics. The first (`Bandwidth ratio`) is the ratio between the total number of bytes communicated when speculation is employed to the total number of bytes communicated when speculation is not employed. The second (`Server Load ratio`) is the ratio between the number of requests for service when speculation is employed to the number of requests for service when speculation is not employed. The third (`Service Time ratio`) is the ratio between the latency of document retrieval when speculation is employed to the latency of document retrieval when speculation is not employed. Finally, the fourth (`Miss rate ratio`) is the ratio between the byte miss rate when speculation is employed to the byte miss rate when speculation is not employed, where the byte miss rate for a given client is the ratio of bytes not found in the client's cache to the total number of bytes accessed by that client.

The trace we used to drive our experiments consisted of 205,925 accesses from 8,474 different clients, representing over 20,000 sessions. This trace was obtained from our departmental HTTP server (`http://cs-www.bu.edu`) by processing the logs for January, February, and March 1995. This processing involved the removal of accesses to nonexistent documents, to live documents, and to scripts, as well as renaming accesses to aliases of a document (*e.g.*, accesses to `http://cs-www.bu.edu` are identical to accesses to `http://cs-www.bu.edu/Home.html`). In our simulations we assumed that a constant number of days (`HistoryLength`) is to be used to estimate the P and P^* relations. Furthermore, we assumed that this estimation is performed periodically, every `UpdateCycle` days.

The parameter settings for our baseline model are summarized in Table 1.

Parameter	Base Value
`CommCost`	1 unit
`ServCost`	10,000 unit
`StrideTimeout`	5.0 secs
`SessionTimeout`	∞ secs
`MaxSize`	∞ (no limit)
`Policy`	$p^*[i, j] \geq T_p$
`HistoryLength`	60 days
`UpdateCycle`	1 day

Table 1: Baseline parameter settings

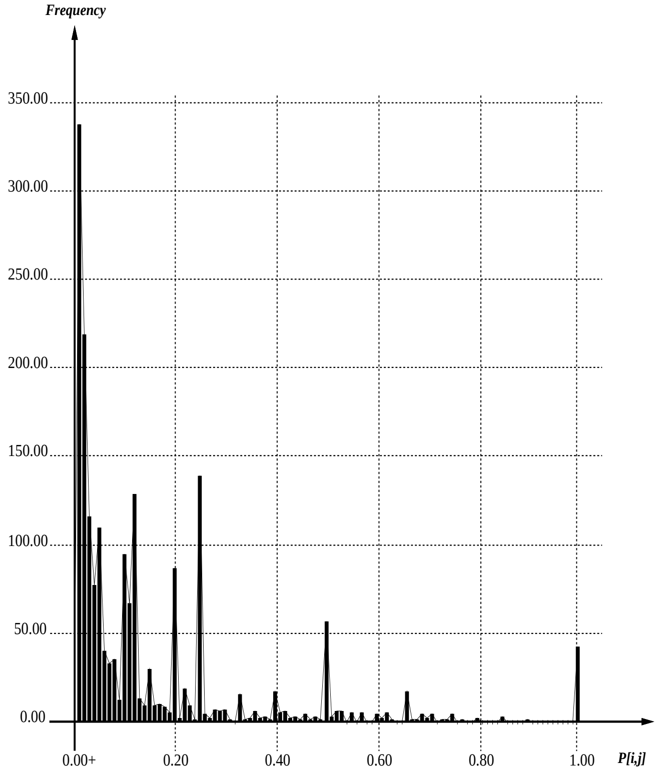

Figure 1: Histogram of dependencies for ranges of $p^{[}i, j]$.

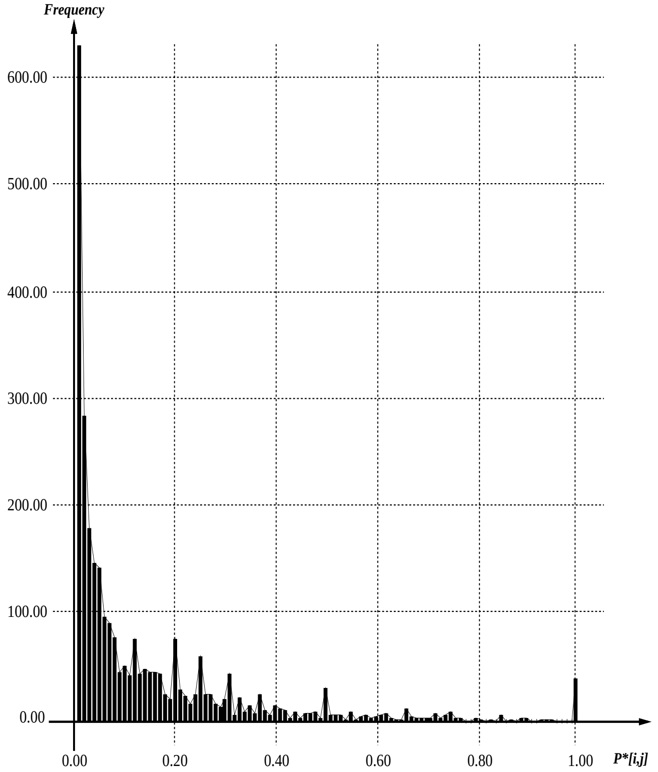

Figure 2: Histogram of dependencies for ranges of $p^*[i, j]$.

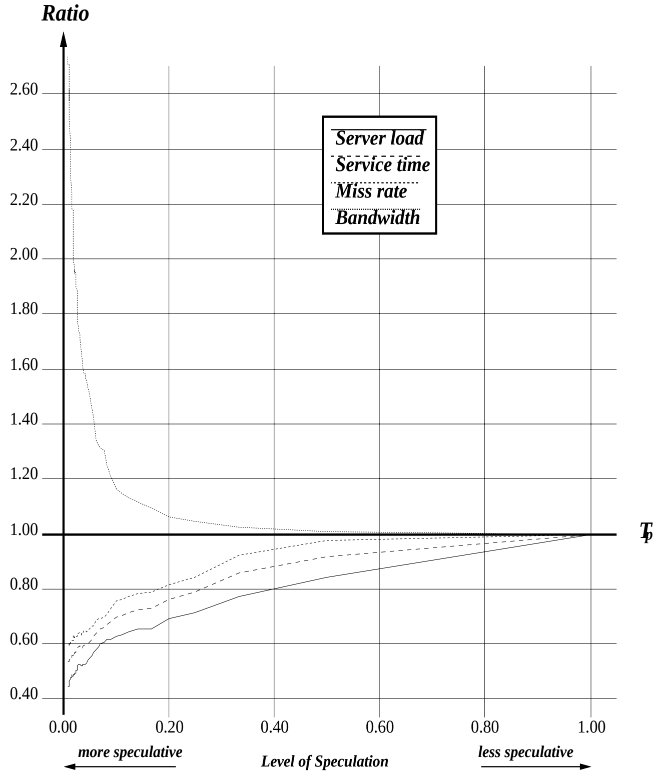

Figure 3: Baseline results for various levels of speculation.

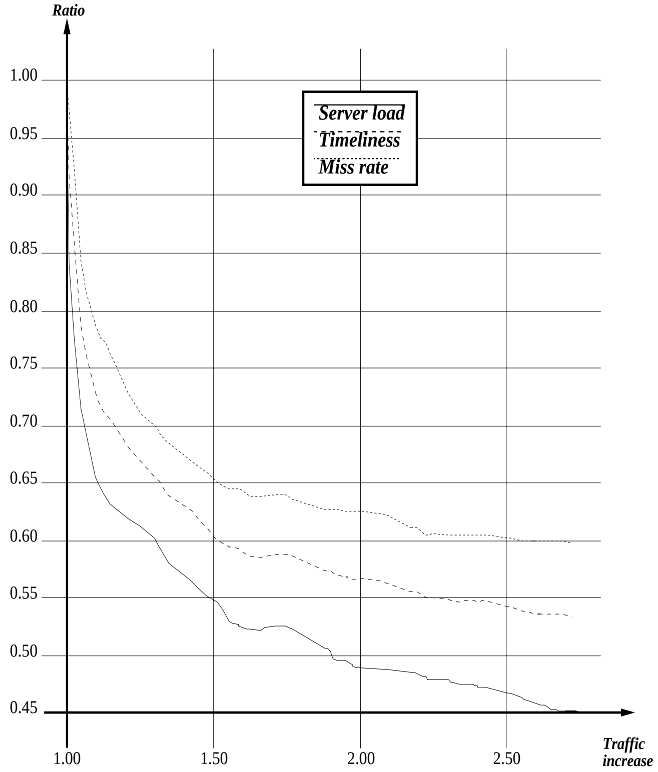

Figure 4: Baseline results for various levels extra traffic.

Baseline Model Results: Figure 3 shows the reduction in server load, in service time, and in client caching miss rate for various levels of speculative service (T_p). The figure also shows the resulting increase in traffic. Figure 4 shows the reduction in server load, the reduction in service time, and the reduction in client caching miss rate as a function of the percentage increase in traffic.

These results suggest that a significant improvement in performance could be achieved for a miniscule increase in traffic. In particular, using only 5% extra bandwidth results in a whopping 30% reduction in server load, a 23% reduction in service time, and a 18% reduction in client miss-rate. Using 10% extra bandwidth results in a reduction of 35%, 27%, and 23% in these metrics, respectively. These performance improvements are above and beyond what is achievable by performing caching at the clients [3]. Figures 3 and 4 suggest that speculation is most effective when done conservatively. Beyond some point, speculation does not seem to pay off. In particular, an aggressive speculation that results in a 50% increase in traffic yields a 45% reduction in server load, a 40% reduction in service time, and a 35% reduction in client miss rate. Increasing traffic by another 50% (for a total of 100% extra traffic) improves performance by only 7%, 6%, and 2%, respectively.

Several interesting observations are evident. First, from the righmost part of Figure 3 we may conclude that capitalizing on embedding dependencies (for which $T_p \approx 1$) does not result in any increase in traffic. This is expected since embedded documents are certainly (and not speculatively) needed at the client; thus, sending them along with the embedding document could not yield any watsed bandwidth. Second, we notice that despite the evident benefits of sending embedded documents along with embedding documents, such benefits are small; they amount to less than 5% improvement in our performance metrics. Most of the performance improvements seem to result from a minimal level of speculation on traversal dependencies.

Stability of the P and P^* Relations: In order to measure the stability of the dependencies captured in the P and P^* relations, we performed trace simulations of a speculative server that updates P and P^* every D days using the traces of the previous D' days. We conducted three sets of experiments (under the baseline parameters) for D=1, 7, and 60 and D'=60. The results are illustrated in Figures 5 and 6, where the performance of the 60-day and 7-day update cycles is compared to that of the 1-day update cycle.

Figures 5 and 6 suggest that, indeed, P and P^* do change (albeit very slowly) with time. This change resulted in an average of 7% and 3% absolute degradation in all measured metrics for the 60-day update cycle and the 7-day update cycle, respectively (both compared to the 1-day update cycle). Figures 5 and 6 also indicate that the performance degradation is less crucial when only modest speculation is done. This implies that high levels of dependency (either embedding or traversal dependency) between documents are less likely to change with time.

The second parameter that affects P and P^* is the extent of time (D') to be used to compute them. In the above experiments D' was fixed to 60 days. Figure 7 shows the performance when $D' = 30$. The figure shows an improvement of about 5% in absolute performance. In a real implementation we envision the use of an aging mechanism to phase-out dependencies exhibited in on older traces, in favor

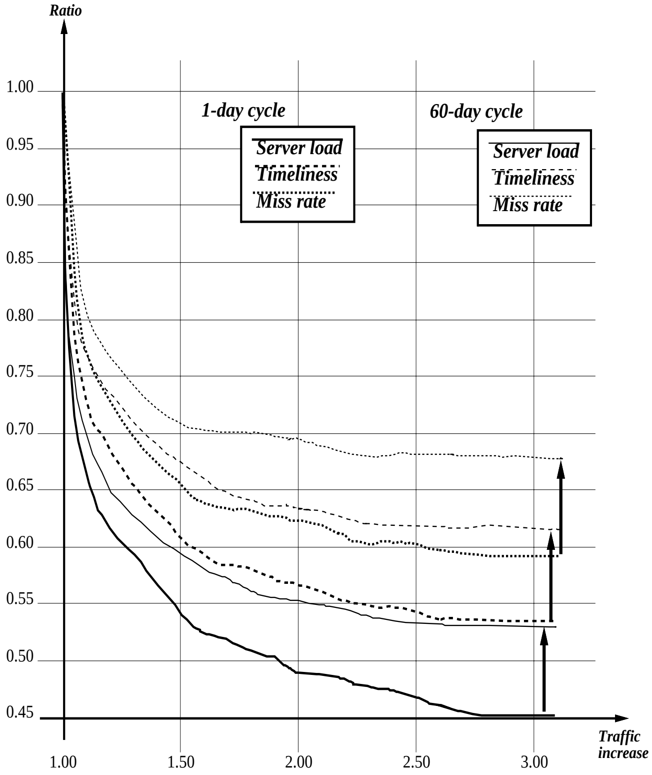

Figure 5: Baseline Simulation results for long update cycle

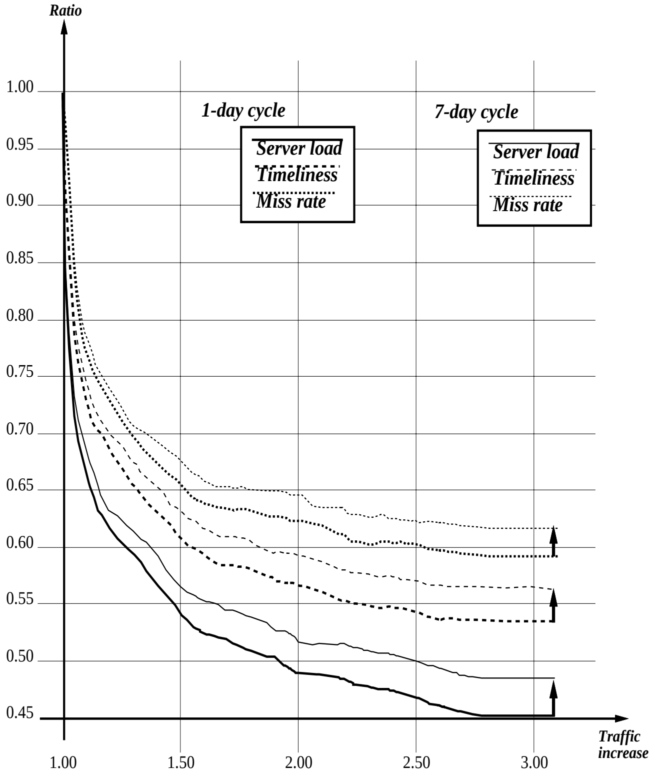

Figure 6: Baseline Simulation results for short update cycle

of dependencies exhibited in more recent traces. This aging mechanism depends highly (among other things) on the frequency and pattern of document updates on the server.

The relative stability of P and P^* observed in the above experiments reinforces our findings in [2] that for WWW documents the popularity profile tends to be stable and updates tend to be infrequent.

Effect of Document Size: The benefits of speculation are most pronounced when documents serviced speculatively are small. This could be explained by noting that if a small document is serviced speculatively, then if it turns out that the document is indeed needed, then the benefit is great—the server is spared from having to respond to an individual request for that document, and the client doesn't have to wait for the overhead of communicating the document's "few" bytes. On the other hand, if it turns out that the document is not needed, then the penalty is minor—only a small increase in bandwidth. To quantify this observation, we performed simulations for various values of `MaxSize`.

Figure 8 shows the achievable performance gains as a function of `MaxSize` for very small levels of speculation (only 3% extra bandwidth generated). From it we conclude that restricting the size of documents speculatively serviced to be under 15K Bytes results in the best possible reduction in server load and in latency.[1] Figure 9 illustrates the effect of `MaxSize` for larger levels of speculation (10% extra bandwidth generated). From it, we notice that the best value for `MaxSize` seems to be around 30K.

Effect of Client Caching: Figures 10 and 11 show the results of simulating clients with modest session caching and clients with no caching capabilities (`SessionTimout` =, 120, and 5 respectively). These simulations clearly suggest that the performance gains achievable through speculative service are possible even in the absence of any long-term client cache. The presence of such a cache (even if modest) is likely to further improve the performance of speculative service as demonstrated in Figure 10.

We have also simulated the performance of speculative service when an infinite client cache is available (i.e. once cached at a client, a document is never requested again from the server.) This corresponds to a `SessionTimout` of ∞. Figure 12 shows the results of our experiments, which suggest that the relative performance of speculative service and non-speculative service in the presence of infinite cache is not as dramatic as with a finite cache. For example, under the baseline parameters, an extra 10% of traffic yields improvements 35%, 27%, and 23% in server load, service time, and miss rate, respectively. For an infinite client cache, these improvements are 32%, 24%, and 19%.

Figure 13 shows the effect of client caching by plotting the performance improvement against `SessionTimeout` for traffic inflation of 10% and 50%. This figure suggests that a threshold exists for `SessionTimeout`. A client cache that keeps all documents speculatively served within a session stride defined using that threshold is expected to reap most of the gains of speculative service. Independent of the level of speculation, the value of that threshold was found to be around `SessionTimeout` = 60 seconds. Another way to interpret the results of Figure 13 is to say that if a speculatively-served document is not accessed by the client to whom it has been served within the same session stride (defined by a `SessionTimeout` = 60 seconds), it is unlikely to be accessed at all. Thus, keeping it or removing it from the cache does not affect the performance gains achievable through speculation.

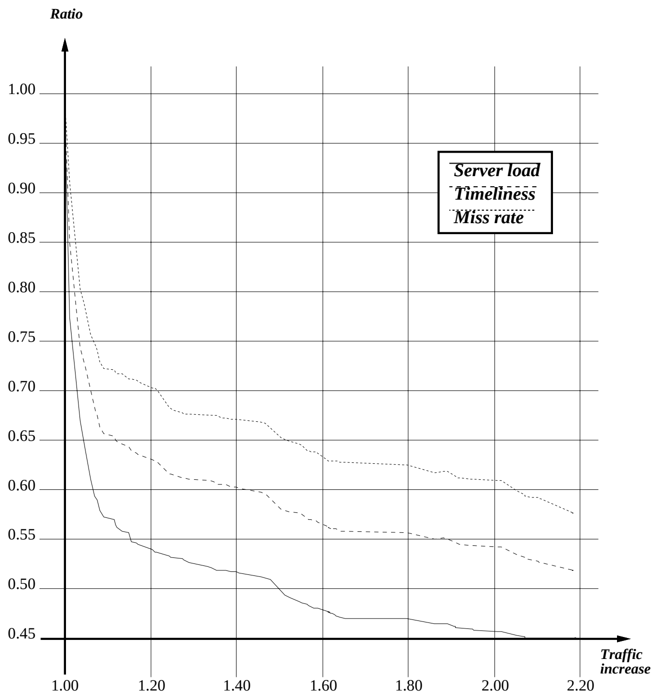

Figure 7: Baseline Simulation results for $D' = 30$

Cooperative Clients: All the trace simulations performed so far have assumed that the server has no knowledge of what the clients are already caching. As a result, it is very possible that a server, when speculatively serving documents to a client, will send documents that are already in the client's cache. Obviously, this is wasteful of bandwidth. To remedy this, we studied the performance of speculative service when clients are cooperative. In particular, we assume that when a client requests a particular document from a server, it piggy-backs its request with a list of document IDs that it already has in its cache from this server. In responding to such a request, the server sends to the client the document it is requesting, along with other documents (*not in the client's cache*) that it speculates will be needed by the client in the future. As expected, our simulations showed that speculative service with cooperative clients results in better bandwidth utilization, especially when the client performs considerable caching. Figures 14 and 15 show the performance of speculative service with and without cooperative clients, for modest client caching (`SessionTimeout`=120) and for extensive client caching (`SessionTimout`=∞).

4 Related and Future Work

Current research in protocols to alleviate network latency and save bandwidth have focussed on caching and/or replication. Traditionally, this has been done in the realms of distributed file systems [10].

[1] Since we are simulating an infinite client cache, the miss rate will always be monotonically decreasing.

Caching to reduce the bandwidth requirements for the FTP protocol on the NSFNET has been studied in [8]. In this study, a hierarchical caching system that caches files at Core Nodal Switching Subsystems is shown to reduce the NSFNET backbone traffic by 21%. The effect of data placement and replication on network traffic was also studied in [1], where file access patterns are used to suggest a distributed dynamic replication scheme. A more static solution based on fixed network and storage costs was suggested in [12]. Multi-level caching was studied in [11], where a two-level caching system is shown to reduce both network and server loads. In [5], a dynamic hierarchical file system, which supports demand-driven replication is proposed, whereby clients are allowed to service requests issued by other clients from the local disk cache. A similar cooperative caching idea was suggested in [7]. A different approach to reducing server load and service time is based on the popularity-based dissemination of information from servers to proxies, which are *closer* to clients. Our work in [2] allows this dissemination to be done so as to make the distance between a client and a document server (or proxy thereof) inversly proportional to the popularity of that document. A similar philosophy was sketched in [9].

In this paper we assumed that servers (and not clients) are the ones to initiate speculative services. This need not be the case, for it is possible to adopt a protocol whereby servers attach to each document they serve a list of document URLs that are highly likely to be accessed in the near future, leaving it to the clients to decide what to prefetch. Also, it is possible to adopt a hybrid protocol whereby server-initiated speculative service is restricted to documents that have a very high probability of being accessed in the near future (e.g. embedded documents), leaving less probable future accesses to client-initiated prefetching.

In an on-going study [4], we evaluated client-initiated prefetching protocols. In that study, extensive user logs [6] are analyzed to obtain a per-user relationship similar to the P and P^* relationships (i.e. a user profile). Such a relationship is used to initiate document prefetching. Preliminary results indicate that client-initiated prefetching is extremely effective for access patterns that involve frequently-traversed documents, but (obviously) not effective at all for access patterns that involve newly-traversed documents. For such an access pattern, only speculative service could improve performance. This led us to consider the incorporation of client-initiated prefetching (based on user access patterns) and server-initiated speculative service (based on server logs) into a single protocol.

5 Conclusion

The notion of speculative service in distributed information systems is novel. It is analogous to prefetching, which is used to improve cache performance in distributed/parallel shared memory systems, with the exception that servers (not clients) control when and what to prefetch. Speculative service could be used efficiently to reduce *both* server load and service time *above and beyond* what is currently achievable using client-based caching. In this paper we have demonstrated the efficacy of speculative service for distributed information systems, such as the WWW, by performing extensive trace simulations to gauge the expected gains of such a technique. We identified a number of issues that may impact the performance of speculative servers.

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 to reduce traffic and balance load in distributed information systems. In *Proceedings of SPDP'95: The 7^{th} IEEE Symposium on Parallel and Distributed Processing*, San Anotonio, Texas, October 1995.

[3] Azer Bestavros, Robert Carter, Mark Crovella, Carlos Cunha, Abdelsalam Heddaya, and Sulaiman Mirdad. Application level document caching in the internet. In *IEEE SDNE'96: The Second International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, June 1995.

[4] Azer Bestavros and Carlos Cunha. A prefetching protocol using client speculation for the www. Technical Report TR-95-011, Boston University, CS Dept, Boston, MA 02215, April 1995.

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

[6] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of www client-based traces. Technical Report TR-95-010, Boston University, CS Dept, Boston, MA 02215, April 1995.

[7] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and Dacid A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *First Symposium on Operating systems Design and Implementation (OSDI)*, pages 267–280, 1994.

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

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

[10] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[11] 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.

[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.

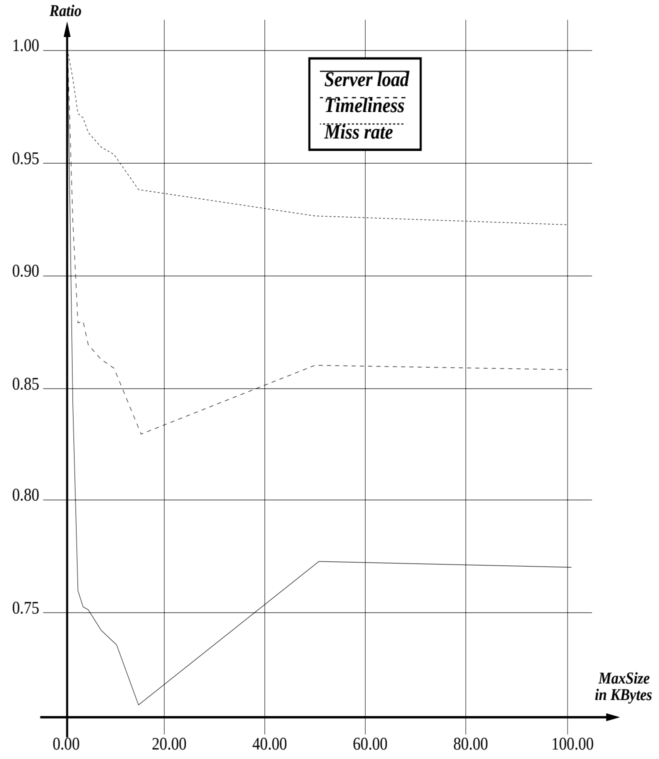

Figure 8: Effect of MaxSize for lower speculation levels

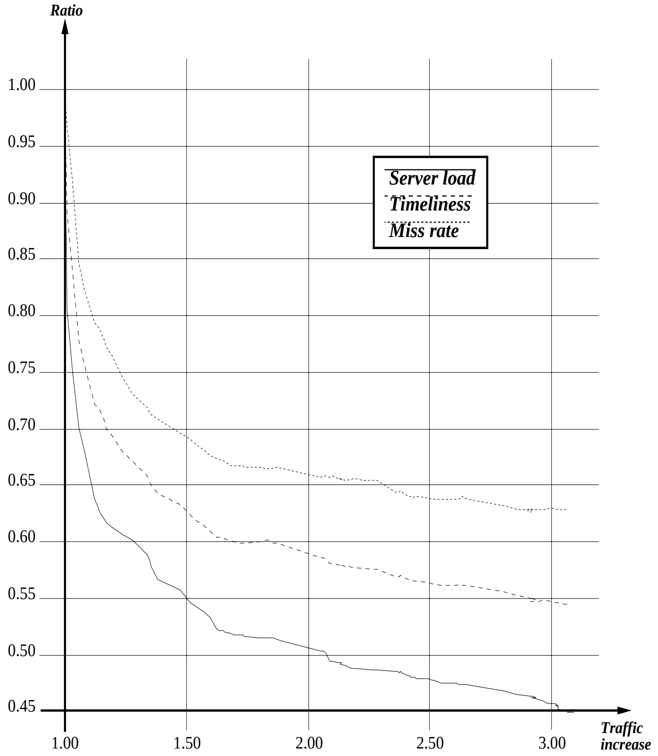

Figure 10: Effect of modest client caching

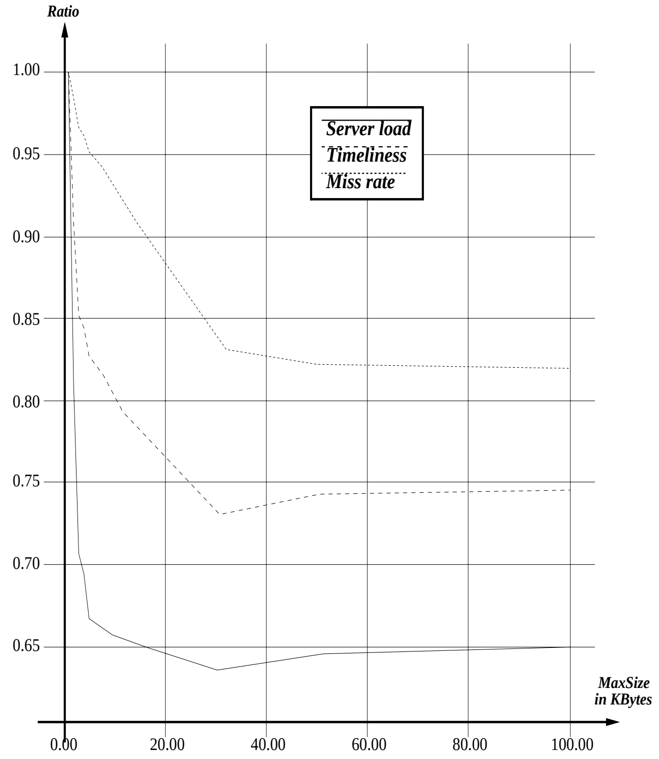

Figure 9: Effect of MaxSize for higher speculation levels

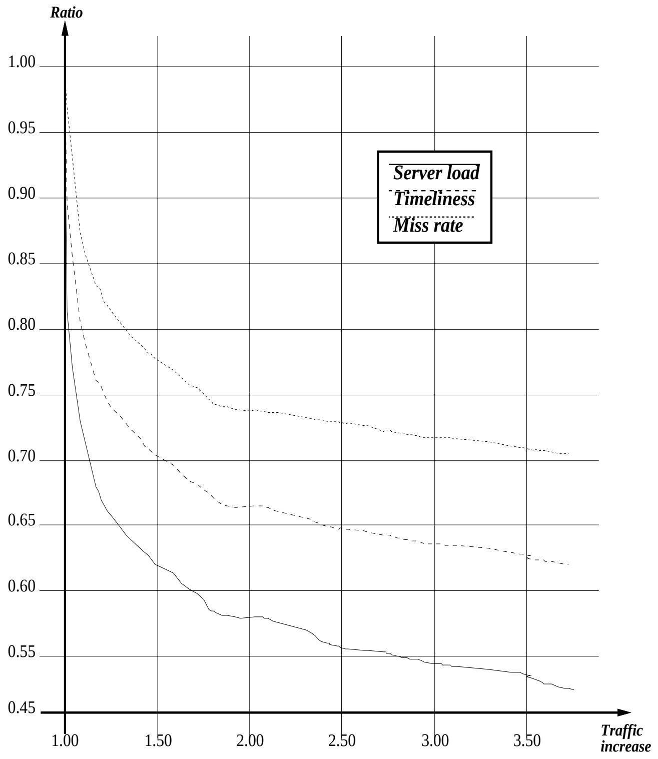

Figure 11: Effect of no client caching

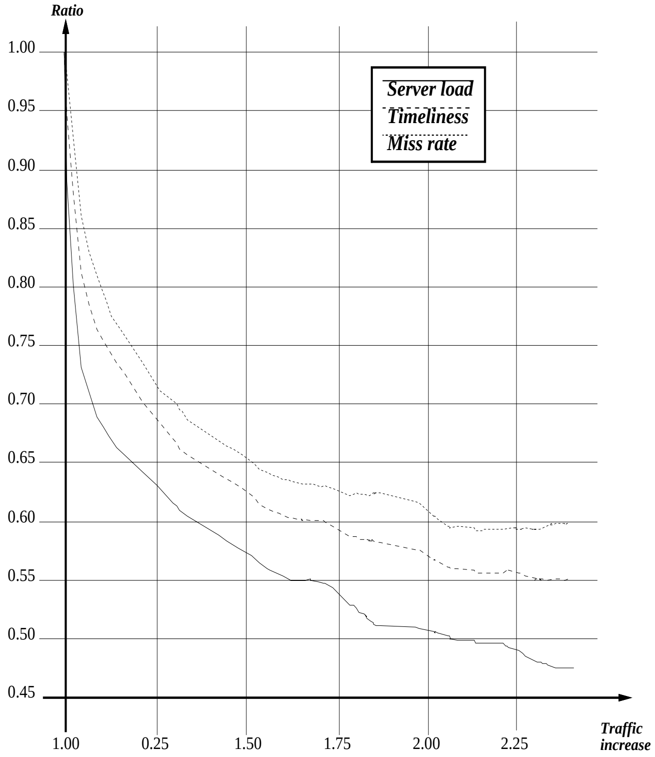

Figure 12: Effect of client caching (infinite caching)

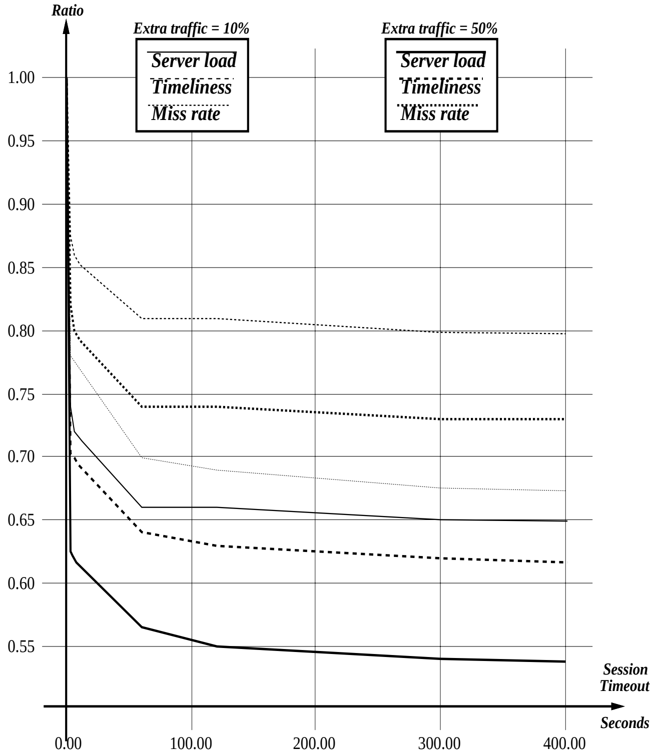

Figure 13: Effect of Client caching (How much is too much?)

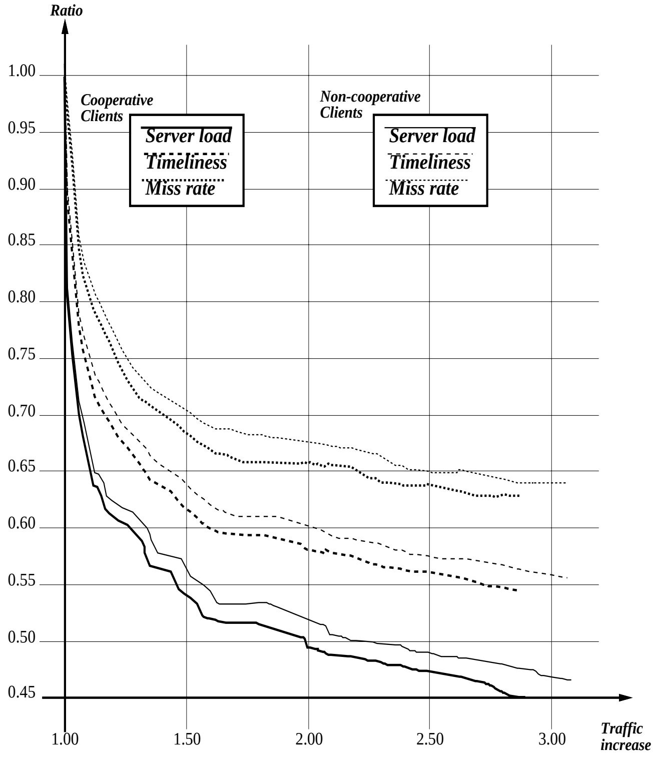

Figure 14: Effect of Cooperative Clients (Modest caching)

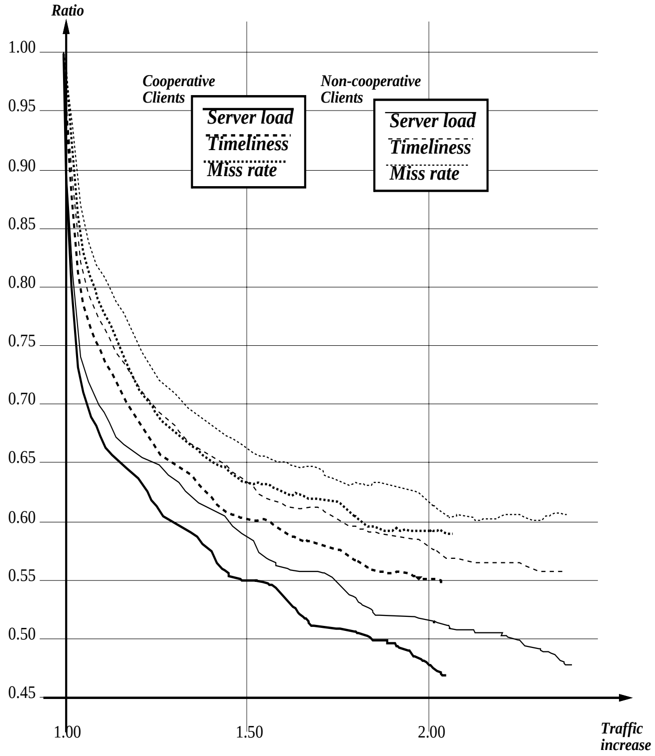

Figure 15: Effect of Cooperative Clients (Extensive caching)