In Proceedings of ICDE'96: The 1996 International Conference on Data Engineering, New Orleans, Louisiana, March 1996.

Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems*

Azer Bestavros
(best@cs.bu.edu)
Computer Science Department
Boston University, MA 02215

Abstract

We present two server-initiated protocols to improve the performance of distributed information systems (e.g. WWW). Our first protocol is a hierarchical data dissemination mechanism that allows information to propagate from its producers to servers that are closer to its consumers. This dissemination reduces network traffic and balances load amongst servers by exploiting geographic and temporal locality of reference properties exhibited in client access patterns. Our second protocol relies on "speculative service", whereby a request for a document is serviced by sending, in addition to the document requested, a number of other documents that the server speculates *will be requested in the near future. This speculation reduces service time by exploiting the spatial locality of reference property. We present results of trace-driven simulations that quantify the attainable performance gains for both protocols.*

1 Introduction

Current research in automated replication techniques concentrate on client-based caching, whereby recently/frequently accessed information is cached at the client (or at a proxy thereof) in anticipation of future accesses to the same information. Traditionally, this was done in the realms of distributed file systems [13, 17, 14, 18], but have been recently extended to distributed information systems (*e.g.* FTP and HTTP) [15, 10, 1, 16, 7, 9, 4]. We believe that these local (myopic) solutions, focussing exclusively on a particular client or set of clients, are likely to have limited impact. In this paper we offer a more global solution that allows the service/replication of information to be done on a supply/demand basis and to be controlled by servers, who unquestionably have a better view of data access patterns than clients. In that respect, we present two server-initiated techniques for reducing server load, network traffic, and service time for distributed information systems. 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.

Our first technique (introduced in [2]) provides a mechanism whereby "popular" data is disseminated automatically and dynamically towards consumers—the more popular the data, the closer it gets to the clients. The extent of this duplication (how much, where, and on how many sites) depends on two factors: the popularity of the data and the expected reduction in traffic if dissemination is done in a particular direction. Our second technique relies on what we term as "speculative service", whereby a server responds to a client's 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.

It is interesting to note that both techniques capitalize on different locality of reference properties that are exhibited in distributed information systems. We single out three such locality of reference properties: temporal, geographical, and spacial. Temporal locality of reference implies that frequently accessed objects (*i.e.* popular objects) are likely to be accessed again in the future. Geographical locality of reference implies that an object accessed by a client is likely to be accessed again in the future by "nearby" clients. Spatial locality of reference implies that an object "neighboring" a recently accessed object is likely to be accessed in the future. Our data dissemination protocol exploits the temporal and geographical locality of reference properties, whereas our speculative service protocol exploits the spatial locality of reference property.

The remainder of this paper is organized as follows. In section 2, we overview the popularity-based data dissemination protocol and validate an analytical model of document popularity. Based on that model, we derive a dissemination strategy that optimizes the allocation of resources dedicated for data replication. In section 3, we introduce the notion of speculative service and document access interdependencies, describe our simulation model and baseline parameters, and 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. Our conclusion and future work is presented in section 4.

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

2 Data Dissemination Protocol

Figure 1 shows the frequency of remote access of individual 256KB document[1] blocks available through `http://cs-www.bu.edu`. The horizontal axis depicts these blocks in a decreasing *remote popularity*. Out of some 2000+ files available through the WWW server only 656 files were remotely accessed at least once. The size of these 656 files totalled some 36.5 MBytes, which represents 73% of the 50+MBytes available through the server. Alone, the most popular 256KB block of documents (that is 0.5% of all available documents) accounted for 69% of all requests. Only 10% of all blocks accounted for 91% of all requests![2]

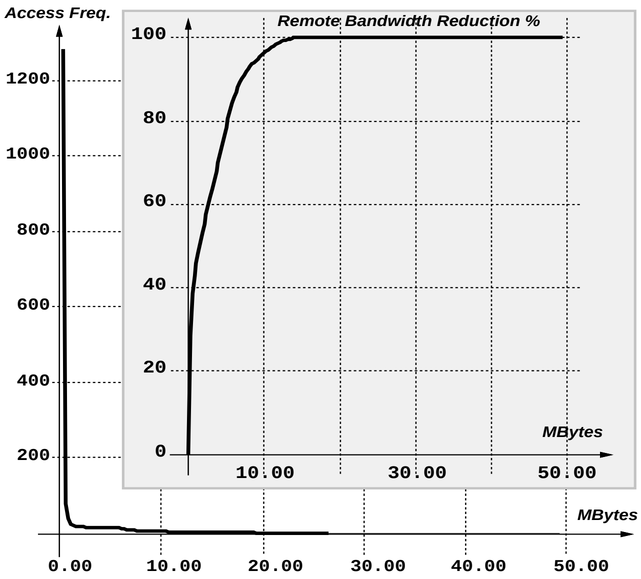

Figure 1: Popularity of various data blocks

The above observation leads to the following question: How much bandwidth could be saved if requests for *popular* documents from outside the LAN are handled at an earlier stage (*e.g.* using a proxy at the "edge" of the organization)? Figure 1 also shows the percentage of the server bandwidth that would be saved if various block sizes of decreasing popularity are serviced at an earlier stage.

A closer look at the logs of `http://cs-www.bu.edu` reveals that there are three distinct classes of documents. Out of the 974 documents accessed during the analysis period: 99 documents had a remote-to-local access ratio larger than 85%—we call these *remotely popular documents*; 510 documents had a remote-to-local access ratio smaller than 15%—we call these *locally popular documents*; 365 documents had a remote-to-local access ratio between 85% and 15%— we call the remaining 365 documents *globally popular documents*.

[1] In this paper we use the term "document" to refer to *any* multimedia object.

[2] These observations have been corroborated in [2] by analyzing the HTTP logs of the Rolling Stones web site, serving more than 1GB/day of multimedia data to tens of thousands of clients.

We monitored the *date of last update* of remotely, locally, and globally popular documents for 186 consecutive days (from March 28, 1995 to October 7, 1995). We observed that both remotely popular and globally popular documents were updated very infrequently (less than 0.5% update probability per document per day), whereas locally popular documents were updated more frequenlty (about 2% update probability per document per day).[3] In all cases, we observed that frequent updates were confined to a *very small* subset of documents. We call these documents *mutable* documents.

The classification of documents into globally, remotely, and locally popular (*i.e.* based on temporal/geographical locality of reference), and into mutable and immutable could be easily done by servers in order to decide which documents to disseminate.

2.1 System Model and Analysis

In our model, information is disseminated from *home servers* (producers) to *service proxies* (agents) closer to *clients* (consumers). We assume the existence of a many-to-many mapping between home servers and service proxies. A service proxy along with the set of home servers it represents form a *cluster*. We model the WWW (Internet) as a hierarchy of such clusters.

Our notion of a service proxy is similar to that of a client proxy, except that the service proxy acts on behalf of a cluster of servers rather than a cluster of clients. In practice, we envision service proxies to be information "outlets" that are available throughout the Internet, and whose bandwidth could be (say) *"rented"*. Alternately, service proxies could be public engines, part of a national computer information infrastructure, similar to the NSF backbone. For the remainder of this paper we use *proxy* to refer to a service proxy.

Our model does not limit the number of service proxies that could be used to "front-end" a particular server. Each server in the system may belong to a number of clusters, and thus may have a number of service proxies acting on its behalf, thus disseminating its documents along multiple routes (or towards various subnetworks). A server is allowed to use (through bidding for example) a subset of these *service proxies* to disseminate its data to clients. Service proxies, themselves, are allowed to use other service proxies to further disseminate this data to clients, and so on.

For a given home server, we view the WWW clientele (Internet) as a tree rooted at the server. The leaves of that tree are the clients and the internal nodes are the potential proxies. In [6], we describe an efficient technique for building such a tree using the *record route* option of TCP/IP. Furthermore, by analyzing the access patterns of clients (as recorded in the server logs), we could *optimally* locate the set of tree nodes to use as service proxies for that home server.[4] Gwertzman and Seltzer [12] sketched an alternative technique for choosing the set of service proxies based on geographical information (such as the distance in actual miles between servers and clients).

[3] Multiple updates to a document within one day were counted as *one* update.

[4] This was done for the `http://cs-www.bu.edu` home server, whose 22-week clientele tree consisted of more than 34,000 nodes.

Let $\mathcal{C} = \mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ denote all the servers in a particular cluster, where $\mathcal{S}_0$ is distinguished as the service proxy (or simply the proxy) of $\mathcal{C}$. Let R_i denote the total number of bytes per unit time (say one day) serviced by server $\mathcal{S}_i$ in a cluster $\mathcal{C}$ to clients outside that cluster. Furthermore, let $H_i(b)$ denote the probability that a request for a document on $\mathcal{S}_i$ will be possible to service at proxy $\mathcal{S}_0$ as a result of disseminating the most popular b bytes from $\mathcal{S}_i$ to $\mathcal{S}_0$. Finally, let B_i denote the number of bytes that proxy $\mathcal{S}_0$ duplicates from server $\mathcal{S}_i$ and let B_0 denote the total storage space available at proxy $\mathcal{S}_0$ (*i.e.* $B_0 = B_1 + B_2 + \ldots + B_n$). By intercepting requests from outside the cluster, we may expect $\mathcal{S}_0$ to be able to service a fraction of these requests. Let $\alpha_{\mathcal{C}}$ be that fraction.

$$\alpha_{\mathcal{C}} = \frac{\sum_{i=1}^{n} R_i \times H_i(B_i)}{\sum_{i=1}^{n} R_i} \tag{1}$$

The objective of $\mathcal{S}_0$ is to allocate storage spaces $B_1, B_2, \ldots, B_n$ so as to maximize the value of $\alpha_{\mathcal{C}}$. The maximum for $\alpha_{\mathcal{C}}$ occurs when for all $j = 1, 2, \ldots, n$:

$$\begin{aligned} \frac{\delta}{\delta B_j}\alpha_{\mathcal{C}} &= k, \text{ for a constant k} \\ \frac{\delta}{\delta B_j}\left(\frac{\sum_{i=1}^{n} R_i \times H_i(B_i)}{\sum_{i=1}^{n} R_i}\right) &= k \\ h_j(B_j) &= k \cdot \frac{\sum_{i=1}^{n} R_i}{R_j} \end{aligned} \tag{2}$$

where $h_j(B_j)$ denotes the Probability Density Function (PDF) corresponding to $H_j(B_j)$. In equation 2 the value of k is chosen to satisfy the constraint $B_0 = B_1 + B_2 + \ldots + B_n$.

Our desire to make our protocol "useful" restricts the type of assumptions we could make. Thus, in our protocol, we have avoided using any parameters that could not be readily estimated from available logs of network protocols (*e.g.* HTTP and FTP). This, however, does not prohibit future work along the same lines from making use of other information to better tune the system. For example, if information about the communication cost between servers, proxies, and clients is available, then our protocol could be easily adapted to weigh such knowledge into our resource allocation methodology.

2.2 Exponential Popularity Model

We use an exponential model to approximate the function $H_i(b)$. Namely, we assume that for $i = 1, 2, \ldots, n$, $H_i(b) = 1 - e^{-\lambda_i . b}$ where λ_i is the distribution's constant. The PDF to $H_i(b)$ is $h_i(b)$, where

$$h_i(b) = \frac{\delta}{\delta b}H_i(b) = \lambda_i e^{-\lambda_i . b} \tag{3}$$

Given a particular server $\mathcal{S}_j$, where $1 \leq j \leq n$, we substite for $h_j(b)$ in equation 2 to get a value for B_j. .

$$B_j = \log\left(\frac{\lambda_j}{k}\frac{R_j}{\sum_{i=1}^{n} R_i}\right)^{\frac{1}{\lambda_j}} \tag{4}$$

Equation 4 specifies a set of n equations to ration the total buffering space B_0 available at $\mathcal{S}_0$ amongst the servers $\mathcal{S}_i$, for $i = 1, 2, \ldots, n$. In order to do so, we must find the value of the constant k. This can be done by observing the requirement that $B_0 \leq B_1 + B_2 + \ldots + B_n$, which results in the following expression for k.

$$k = \frac{1}{\sum_{i=1}^{n} R_i}\left(\frac{\prod_{i=1}^{n} (\lambda_i R_i)^{\frac{1}{\lambda_i}}}{e^{B_0}}\right)^{\frac{1}{\sum_{i=1}^{n} \frac{1}{\lambda_i}}} \tag{5}$$

Substituting for k from equation 5 into equation 4, we get the optimum storage capacity to allocate on $\mathcal{S}_0$ for a particular server $\mathcal{S}_j$, where $1 \leq j \leq n$.

The above calculations require that R_i and λ_i be estimated, for $i = 1, 2, \ldots, n$. This can be easily and efficiently computed from the server logs. As a matter of fact, figure 1 was produced by a program that computed these parameters for `cs-www.bu.edu`. Moreover, our measurements suggested that these parameters are quite static, in that they change only slightly over time. Hence, the calculation of R_i and λ_i as well as the allocation of storage space on $\mathcal{S}_0$ for servers $\mathcal{S}_i$, for $i = 1, 2, \ldots, n$ need not be done frequently. It could be calculated either off-line or periodically (*e.g.* weekly).

2.3 Special Cases

In order to develop an understanding of our demand-based document dissemination protocol, we consider several special cases.

Equally Effective Duplication:
Let $\lambda_i = \lambda$ for $i = 1, 2, \ldots, n$. That is, we assume that the reduction in bandwidth that results from duplicating some number of bytes from a particular server $\mathcal{S}_j$ is equal to the reduction in bandwidth that results from duplicating the same number of bytes from *any* other server $\mathcal{S}_i$ for $i = 1, 2, \ldots, n$. Substituting in equation 5 and then equation 4, we get:

$$B_j = \frac{B_0}{n} + \frac{1}{\lambda}\log\frac{R_j}{\sqrt[n]{\prod_{i=1}^{n} R_i}} \tag{6}$$

Equation 6 suggests that popular servers are allocated *extra* storage capacity on the proxy. This extra storage depends on two factors: $\frac{1}{\lambda}$, which is a measure of duplication effectiveness, and $\log(R_j/\sqrt[n]{\prod_{i=1}^{n} R_i})$, which reflects a server's popularity relative to the geometric mean of all servers in the system. This dual dependency on duplication effectiveness and relative popularity gives us a handle on how to extend our results for arbitrary distributions of $H_i(b)$. In particular, if the skewness of $H_i(b)$ could be measured for a particular server (by analyzing its logs as suggested earlier), then this measure could be used instead of $\frac{1}{\lambda}$.

Equally Popular Servers:
Let $R_i = R$ for $i = 1, 2, \ldots, n$. That is, we assume that all servers in the system are equally popular. Substituting in equation 5 and then in equation 4, we get:

$$B_j = \frac{1}{\sum_{i=1}^{n} \frac{\lambda_j}{\lambda_i}}\left(B_0 + \sum_{i=1}^{n} \frac{1}{\lambda_i}\log\frac{\lambda_j}{\lambda_i}\right) \tag{7}$$

Equation 7 suggests that servers, whose data are accessed more uniformly (*i.e.* servers with a smaller λ) should be alloted more storage capacity on the proxy as long as the total capacity available on the proxy is large enough (*i.e.* $B_0 \gg \frac{n}{\lambda_i}$). However, if the storage capacity of the server is not big enough, servers with intermediate values for λ should be favored. Figure 2 shows the optimal storage capacity to be allocated to server $\mathcal{S}_j$ for various values of λ_j assuming that all other $n-1$ servers have equal λ_i and that $B_0 = \frac{1}{\lambda_i}$ (tight) or $B_0 = 10\frac{1}{\lambda_i}$ (lax), for $1 \leq i \leq n$ and $i \neq j$.

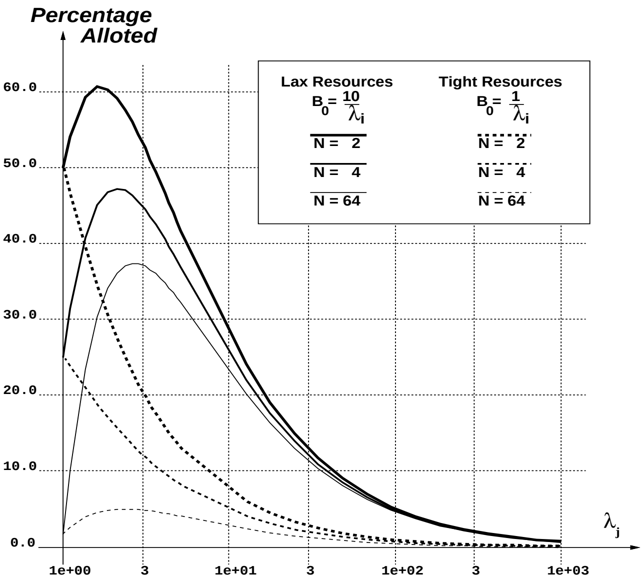

Figure 2: Storage allocation for $R_i = R$ and $B = \frac{1}{\lambda_i}$

Symmetric Clusters:
In order to appreciate the effectiveness of our demand-based document dissemination, we consider a symmetric cluster, where all servers have identical values for R_i and λ_i. From equation 5 and equation 4, we get

$$B_j = \log\left(\frac{\lambda}{\frac{\lambda}{n}\cdot e^{-\frac{\lambda}{n}B_0}}\frac{R}{\sum_{i=1}^{n} R}\right)^{\frac{1}{\lambda}} = \frac{B_0}{n} \quad (8)$$

As expected, equation 8 provides equal allocation of storage on $\mathcal{S}_0$ for all the servers in the cluster. By substituting the value of B_j into equation 1, we get:

$$\alpha_{\mathcal{C}} = \frac{\sum_{i=1}^{n} R \times H(\frac{B_0}{n})}{\sum_{i=1}^{n} R} = 1 - e^{-\lambda\frac{B_0}{n}} \quad (9)$$

Equation 9 could be used to estimate the storage requirements on the proxy as a function α.

$$B_0 = \frac{n}{\lambda}\log\frac{1}{\alpha_{\mathcal{C}}} \quad (10)$$

Equation 10 suggests that if (say) the `http://cs-www.bu.edu` server is one of 10 servers, whose most popular data are duplicated on a proxy, then in order to reduce the remote bandwidth by (say) 90% on *all* servers, the proxy must secure 36 MBytes to be divided equally amongst all servers. This assumes a value of $\lambda = 6.247 \times 10^{-7}$, which was estimated from the HTTP demon logs on the `cs-www.bu.edu` server. With a storage capacity of 500 MBytes, a proxy could shield 100 servers from as much as 96% of their remote bandwidth. These numbers, of course, raise a legitimate question: If 96% of all remote accesses to 100 servers (or even 90% of all accesses to 10 servers) are now to be served by *one* proxy, isn't that proxy going to become a performance bottleneck? The answer is yes, *unless* the process of disseminating popular information continues for another level, and so on. If that is not possible, then another solution would be for the proxy to *dynamically* adjust the level of "shielding" it provides for its constituent servers. In other words, when the proxy becomes overloaded, then B_0 is reduced, thus forcing more of the requests back to the servers.

2.4 Achievable Bandwidth Reduction

We performed simulations of our dissemination strategy. Our simulations were driven by the traces of 22-weeks obtained from `http://cs-www.bu.edu` [6]. Figure 3 shows the expected percentage reduction in bandwidth (measured in *bytes* $\times$ *hops*) that is achievable by disseminating a percentage of the most "popular" data available on the server. The horizontal axis shows the number of proxies to which this data was disseminated. In our simulation, the same data is disseminated to all proxies. Better results are attainable if the dissemination strategy takes advantage of the geographic locality of reference[5] Two curves are shown. The first assumes that the most popular 10% of the data is to be disseminated, whereas the second assumes that the most popular 4% of the data is to be disseminated. The curves are labeled by the total storage capacity needed on all proxies for each level of dissemination.

3 Speculative Service Protocol

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 clients access patterns to identify the spatial locality of reference that may exist amongst the various documents.

3.1 Document Access Interdependencies

Let $p[i,j]$ denote the conditional probability that $\mathcal{D}_j$ will be requested, within a limited window of time T_w, given that $\mathcal{D}_i$ has been already requested. Thus, 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

[5] By disseminating different data to different proxies based on the access patterns of clients served by each proxy.

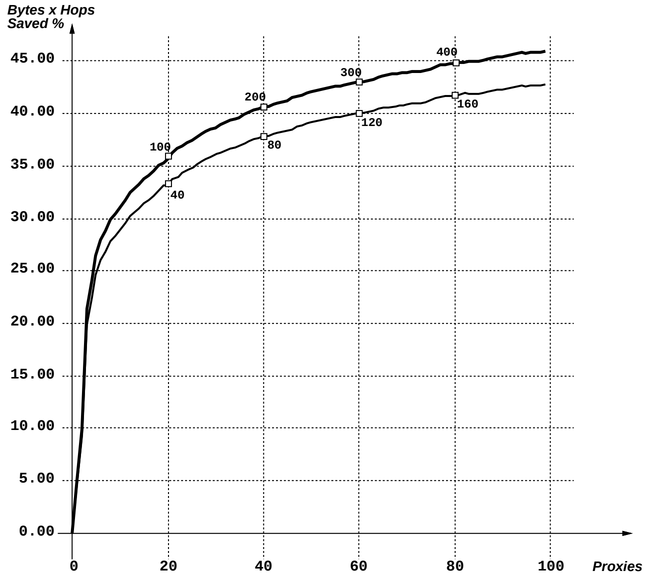

Figure 3: Bandwidth saved as a result of dissemination

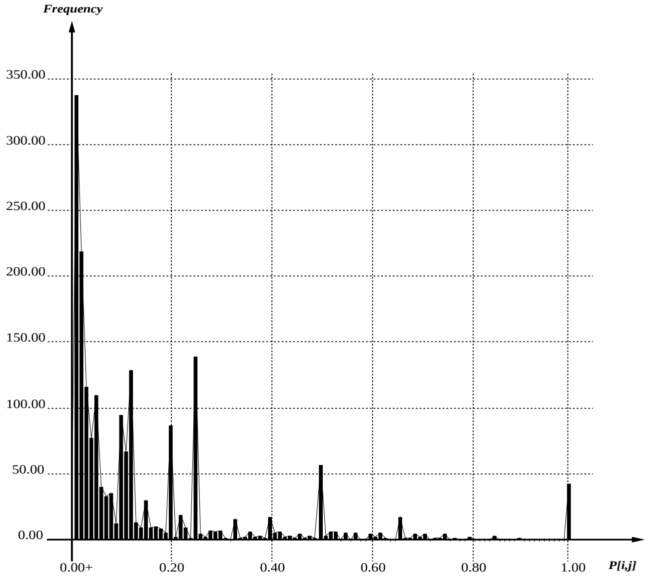

Figure 4: # of $(\mathcal{D}_i, \mathcal{D}_j)$ pairs for various ranges of $p[i, j]$

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 `http://cs-www.bu.edu` for the month of January 1995 (more than 50,000 accesses) we computed the function P and its closure P^*. Figure 4 shows a histogram of the number of document pairs ($\mathcal{D}_i$ and $\mathcal{D}_j$) against various ranges of $p[i,j]$, assuming that the value of T_w is set at 5 seconds. Figure 4 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 4 suggests that for a large number of documents, the probability of following these anchors is equal.

We distinguish between two types of document 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 4, embedding dependencies are responsible for the peak at the rightmost part of the graph.

3.2 System Model and Simulation

We ran a number of trace simulations to evaluate the potential from exploiting the spatial locality of reference property exhibited in the P and P^* relationships. 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.

We assume that if two requests from the *same* client are issued within `StrideTimout` seconds of each other, then these requests are *dependent*, and thus significant in calculating the dependencies captured by the functions P and P^*. We define a *traversal stride* to be a sequence of requests 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. Setting `StrideTimeout` to a very small value restricts the definition of document dependency to embedding dependencies, whereas setting it to a larger value loosens the definition to include traversal dependencies as well.

We assume 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), and 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 where the time between successive requests is less than `SessionTimeout` seconds. By controlling the value of `SessionTimeout`, we can emulate various caching policies. Setting `SessionTimeout` to ∞ emulates a client with an infinite-size multi-session cache (*e.g.* the LAN cache proposed in [4]). Setting `SessionTimeout` to (say) 60 minutes emulates a client with an infinite-size single-session cache. Setting `SessionTimeout` to 0 emulates a client with no cache.

The cost model we adopted 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 `http://cs-www.bu.edu` by processing the logs for January, February, and March 1995.[6] In our simulations we assumed that a constant number of days (`HistoryLength`) is 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 below.

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

3.3 Baseline Model Results

Figure 5 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 6 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 [4]. Figures 5 and 6 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.

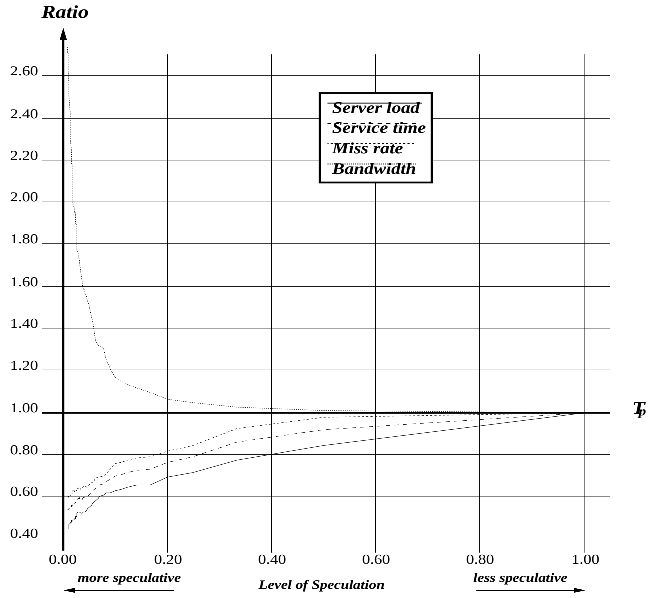

Figure 5: Baseline simulation results

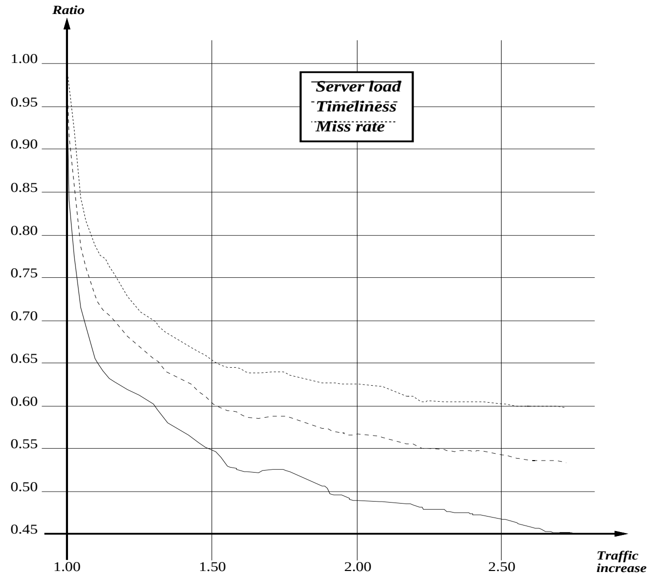

Figure 6: Performance gains versus bandwidth used

[6] This processing involved the removal of accesses to non-existent documents, to live documents, and to scripts, as well as renaming accesses to aliases of a document.

Several interesting observations are evident. First, from the rightmost part of Figure 5 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.

3.4 Fine-tuning speculative service

In the remainder of this section, we discuss the performance of speculative service under various assumptions and refinements. Detailed experiments and results are described in [3].

Stability of the P and P^* Relations
In order to measure the stability of the embedding/traversal dependencies captured in the P and P^* relations, we performed trace simulations of a speculative server that updates the relations P and P^* every D days using the traces of the previous D' days. We conducted three sets of experiments for D=1, 7, and 60 and D'=60, all under the baseline parameters. Our experiments suggest that, indeed, the relations P and P^* do change (albeit very slowly) with time. This change resulted in an average of 7% absolute degradation in all measured metrics for a 60-day update cycle and an average of 3% absolute degradation in all measured metrics for a 7-day update cycle, both compared to the 1-day update cycle. Our experiments also indicate that the performance degradation is less crucial when only modest speculation is done.

The second parameter that affects the P and P^* relation is the extent of time (D') to be used to compute them. In the above experiments D' was fixed to 60 days. By changing D' to 30 days, we observed 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 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] and the findings of Gwertzman in [11] 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 phenomenon, we performed simulations for various values of `MaxSize`. Our experiments showed that there exists an optimal `MaxSize` for each level of extra bandwidth. For example, if only 3% extra bandwidth is tolerable, then `MaxSize` = 15KB results in the best possible reduction in server load and in latency. Similarily, if 10% extra bandwidth is tolerable, then `MaxSize` = 29KB is optimal.

Effect of Client Caching
We simulated the performance of speculative service under various client caching capabilities. Our experiments showed that 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 performance.

We have also simulated the performance of speculative service when an infinite client cache is available. From these experiments we conclude 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%.

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.

Server-assisted Prefetching
Rather than speculatively serving documents to clients, servers could assist clients in prefetching decisions. In particular, servers could 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.

Alternately, client-initiated prefetching could be based on user logs (as opposed to server logs). In an on-going study [5], extensive user logs [8] 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 not effective at all for access patterns that involve newly-traversed documents. For such access patterns, only speculative service could improve performance. This suggests the incorporation of client-initiated prefetching (based on user access patterns) and server-initiated speculative service (based on server logs) into a single protocol.

4 Conclusion

Current service protocols in large-scale distributed information systems are client-based; they do not make use of the knowledge amassed at servers concerning client access patterns. In this paper we have demonstrated that such knowledge could be used effectively, and in that respect, we have identified two server-initiated mechanisms that make use of such knowledge, namely data dissemination and speculative service. Using extensive trace simulations we have quantified the potential gains of such protocols. In particular, the demand-driven dissemination of data from producers to consumers was shown to be most effective in reducing network traffic (by as much as 40%) and in balancing the load amongst servers. Speculative service, on the other hand, was shown to be quite effective in reducing service time (by as much as 50%) and server load (by as much as 42%).

The protocols presented in this paper are meant to exemplify the potential value of client access patterns in engineering scalable protocols and services in large-scale distributed information systems (such as the WWW). Work in progress involves the development of prototypes to test and evaluate these protocols and variations thereof.

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 7th IEEE Symposium on Parallel and Distributed Processing*, San Anotonio, Texas, October 1995.

[3] Azer Bestavros. Using speculation to reduce server load and service time on the www. In *Proceedings of CIKM'95: The 4th ACM International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1995.

[4] 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.

[5] 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.

[6] Azer Bestavros and Carlos Cunha. Replica placement for distributed information systems. Technical Report (In Progress), Boston University, CS Dept, Boston, MA 02215, November 1995.

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

[8] 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.

[9] 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.

[10] 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.

[11] James Gwertzman. Autonomous replication in wide area networks, 1995. Senior Thesis, Harvard University, DAS.

[12] 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.

[13] 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.

[14] J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith. Andrew: a distributed personal computing environment. *Comm. ACM*, 29(3):184–201, Mar. 1986.

[15] 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.

[16] 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.

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

[18] 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.