In Proceddings of ACM SIGMOD'96 Data Mining Workshop, Montreal, Canada, June 1996.

Middleware Support for Data Mining and Knowledge Discovery in Large-scale Distributed Information Systems*

Azer Bestavros
(best@cs.bu.edu)

Computer Science Department
Boston University, MA 02215

Abstract

With the growth of the Internet and the corresponding agreement on a common network and transport infrastructure comes a new opportunity for innovative network services. Examples include the rapid evolution of the World-Wide Web (WWW), and the ensuing interest in mining knowledge from the large body of WWW databases and repositories. At the same time, the explosive growth in availability and use of the Internet is creating new challenges. In particular, the existing infrastructure must be extended if it is to continue to scale. This demands that we raise the level of common services and introduce new types of higher-level services. These common services, termed *"middleware"* services [pc95], would sit above the traditional network protocols and provide means for extending commonly available services on the network to enclose higher layers of abstraction. Many of these middleware services are crucial for the design of *scalable* data mining and knowledge discovery systems. In this paper we focus on three such candidate middleware components, namely information dissemination, speculative service, and resource management for responsive computing. We argue that these middleware components have the potential to alleviate the latency and bandwidth requirements of data mining and knowledge discovery systems.

1 Introduction

The explosive growth in availability and use of the Internet demands that we raise the level of common services and introduce new types of higher-level services. These common services would lie between the transport and application levels, hence the term *"middleware"* [pc95], and would provide means for extending commonly available *metacomputing* services on the Internet[1] to support the demands of High Performance Computing (HPC) applications, including those requiring data mining and knowledge discovery services.

There are opportunities to create many such middleware components, including caching and replication services, brokerage and resource management services, indexing services, remote scripting environments, data typing and structuring primitives, and higher level communication abstractions such as multicast and causal broadcast. Many of these middleware components are crucial for the design of *scalable* data mining and knowledge discovery systems. In this paper we focus on three such candidate components, namely resource management and brokerage services for responsiveness, information dissemination, and speculative service. We argue that these middleware components have the potential to alleviate the latency and bandwidth requirements of data mining and knowledge discovery systems. In our work, we use the WWW as the underlying distributed computing resource to be managed by these middleware components. 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 research in middleware services is aimed at: 1) Improving the predictability of network services through efficent resource management protocols; and 2) Improving the scalability of information retrieval for data mining and knowledge discovery applications through proper partitioning and distribution of data. In the remainder of this paper we overview three specific middleware components that aim at addressing these issues.

2 Middleware Resource Management

Our first middelware component adds responsiveness (timeliness + fault-tolerance) to the services available in Internet *metacomputing* environments. FAFNER [Col] is an example of such an environment. FAFNER brings resources and expertise from many sites world-wide to solve the problem of factoring RSA-130. The approach that FAFNER uses to harness the Internet resources available at its disposal can be best described as a *best-effort* approach, which does not offer any guaranteed

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

[1] In the remainder of this paper, we will use the term "Internet" to mean either *the* Internet or the intranet of a particular enterprise

performance. While useful for a variety of applications, *best effort* Metacomputing is not sufficient for applications that are subject to timing and reliability constraints. Examples of such applications include: interactive applications (*e.g.*, battlefield group simulations), time-constrained database queries for real-time applications (*e.g.* image search for tactical image analysis), and data-mining and knowledge discovery applications that involve temporal data (*e.g.* stock market modeling and weather prediction). The real-time and reliability constraints imposed by these (and other) applications require *responsive* rather than merely *best-effort* metacomputing. We call such an environment a *Responsive Web Computer* (RWC).

The *best-effort* philosophy of current metacomputing platforms is due to the unpredictability of the underlying computing infrastructure, which is due to the inability of applications to control or negotiate the resources they need. Therefore, in order to achieve responsive metacomputing, new middleware services need to be developed to reduce this unpredictabilty and to allow a certain level of commitment when resources are contributed to a metacomputing platform.

The Resource Management Interface (RMI) is a middleware service (abstraction) that allows the computational requirements of processes to be matched with the resources available at the disposal of the system through a schedule that satisfies the timing and fault-tolerance requirements of these processes. The overall structure of middleware RMI is shown in Figure 1. There are three main services to be supported. The Task Registration Service allows the computational resources needed by, and the performance constraints imposed on a task to be specified. The Resource Registration Service allows the computational resources contributed (or leased) to the system to be specified. The Resource Management Service provides the admission control and scheduling protocols for managing the registerd resources in accordance with the performance constraints of the registered tasks.

2.1 Task Specification

To motivate the issues involved in the specification of a task to the resource manager, consider a simple situation where a real-time application is to be executed. Furthermore, assume that the real-time application consists of a large set of periodic processes, where each process requires various resources (*e.g.* CPU cycles, network bandwidth, *etc.*) As a concrete example, consider a Web agent, which is responsible for monitoring the contents of a particular object (*e.g.* areal radar map, wheather map, number of objects in a given database, stock quote). Assume that the performance of that agent is constrained so as to report back the results of its operation periodically (say every minute) to another agent (*i.e.* another RWC task). Furthermore,

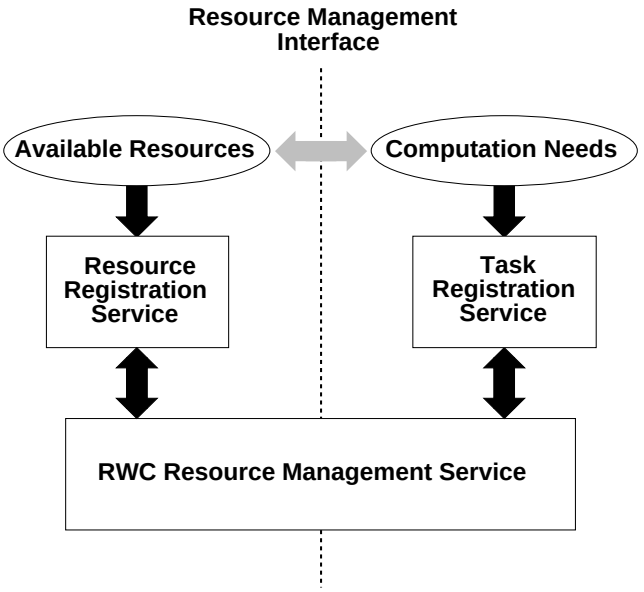

Figure 1: Components of the middleware RMI

assume that it is requested that the agent's periodic behavior must be started within one hour time.

A possible representation of this application would be a set of tuples of the form $(P_i, R_i, S_i, C_i, T_i, L_i)$, where P_i is a process ID, R_i is a resource ID, T_i is the period of P_i, S_i is the startup demand of P_i from R_i, C_i is the cumulative demand of P_i from R_i per period T_i, and L_i is the latest time for starting the periodic component of P_i (in other words, it is the deadline imposed on finishing up the startup, or non-periodic component of P_i.) To clarify this particular model, we look at an example (see Figure 2 for others). If R_i is a CPU resource, then S_i would be the CPU time needed to startup the periodic behavior of P_i, and C_i would be the CPU time needed by P_i every T_i units of time thereafter. The periodic behavior of P_i must start not later than time L_i.

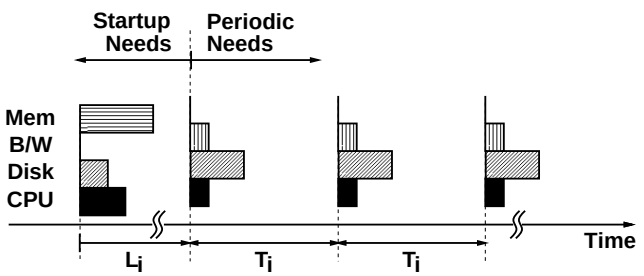

Figure 2: Specification of the Resource Requirements and Timing Constraints of a Task

2.2 Resource Specification

To motivate the issues involved in the specification of a resource to the resource manager, consider a simple situation in which a Web server is to be contributed to the RWC (say overnight), for a period of 12 hours. However, assume that this server is also expected to provide other services (*e.g.* FTP, HTTP, *etc.*). Rather than totally committing this server to the RWC (and thus making it unavailable for other tasks), it may be prudent to commit only (say) 50% of its CPU time to the RWC and only (say) 100 MB of its disk space, for example. This level of commitment may be different depending on the mode of the Web server's operation, *etc.*

A possible representation of resources contributed to the RWC would be to use tuples of the form $(Q_i, R_i, C_i, T_i, L_i)$, where Q_i is a Web Computer ID and R_i, C_i, and T_i are as before, except that instead of *demand*, C_i represents *supply* (or availability). L_i demarks the end of the commitment period. One way of thinking about this model of RWC resources is to think of the contributor of the resource as specifying a *budget* C_i for the RWC to expend out of this resource until time L_i. This budget is replenished to C_i every P_i units of time. The rational behind representing the availability of resources in this fashion is to allow RWC contributors to control the level at which their resources are to be committed to the RWC. In particular, by adjusting the ratio of C_i to T_i, a contributor controls the percentage of the resource capacity to be committed to the RWC. The intrusion of such commitment can be further tuned by adjusting the period T_i—in effect limiting the length of time the contributor's local processing would have to wait for RWC processes.

2.3 Resource Management Algorithms

Once an appropriate interface is defined, the assignment of processes to RWC resources could be accomplished. For example, the process of finding a Web Computer that could satisfy the requirements of a particular process P_i amounts to searching for a contributor Q_j such that for every tuple $(P_i, R_i, S_i, C_i, T_i, L_i)$, there exists a tuple (Q_j, R_i, C_j, T_j) such that the following is true:

$$\lceil \frac{S_i}{C_j} \rceil \times T_j \leq L_i - t \quad (1)$$

$$\frac{C_i}{T_i} \leq \frac{C_j}{T_j} \quad (2)$$

$$T_j \leq T_i \quad (3)$$

In order to provide the resource management algorithm with the information it needs about processes and resources, it is necessary to devise a protocol for the collection/dissemination of such information. For example, one could use a hierarchical registration protocol such as the one implemented in FAFNER [Col]. Alternatively, one could devise a more distributed *peer-to-peer* protocol, whereby resource availability is dessiminated passively (via gossiping [BS93]) or actively (via bidding [Sta89]). The hierarchical approach has the advantage of being simpler to implement than the peer-to-peer approach. However, the peer-to-peer approach offers more resiliency against failures, *etc.*

2.4 Other Issues

The simplistic resource management model and algorithm presented above are just to motivate issues involved in resource management for a RWC. A realistic model must consider many more issues, including: dealing with priority issues, dealing with different deadline semantics, dealing with inter-process synchronization issues, dealing with fault-tolerance requirements, dealing with non-homogeneous platforms, and defining/classifying more complex resources and processes.

3 Middleware Data Dissemination

Our second middleware component [Bes95a] 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. Data dissemination has the potential for reducing *considerably* the amount of resources needed for data mining. In a sense, data dissemination reduces *"the depth"* of data mining.

3.1 Motivation

Figure 3 shows the frequency of remote access of individual 256KB document[2] 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![3]

The above observation leads to the following question: How much bandwidth could be saved if requests for

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

[3] These observations have been corroborated in [Bes95a] 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.

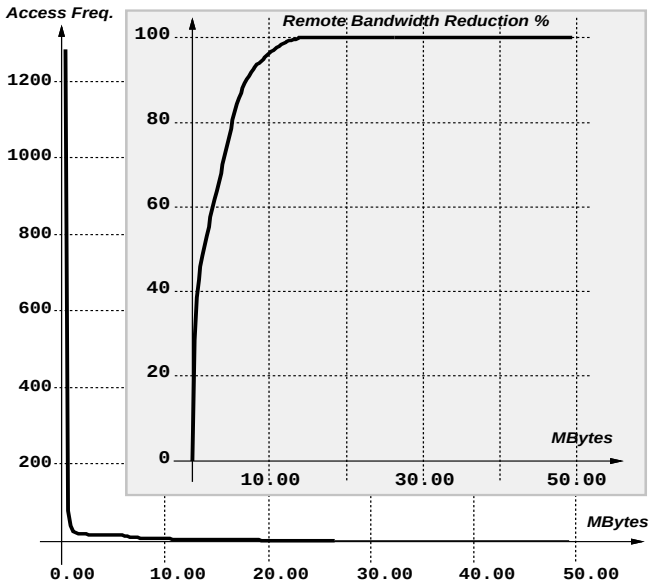

Figure 3: Popularity of various data blocks

popular documents are handled at an earlier stage (*e.g.,* using a proxy at closer to the Internet backbone)? Figure 3 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.

3.2 System Model

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 [BC95], 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 [GS95] 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).

3.3 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` [BC95]. Figure 4 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. 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.

4 Middleware Speculative Service

Our third middleware component relies on what we term as "speculative service" [Bes95b], 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. This speculation is based on statistical information that the server maintains for each document it serves.

4.1 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

[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.

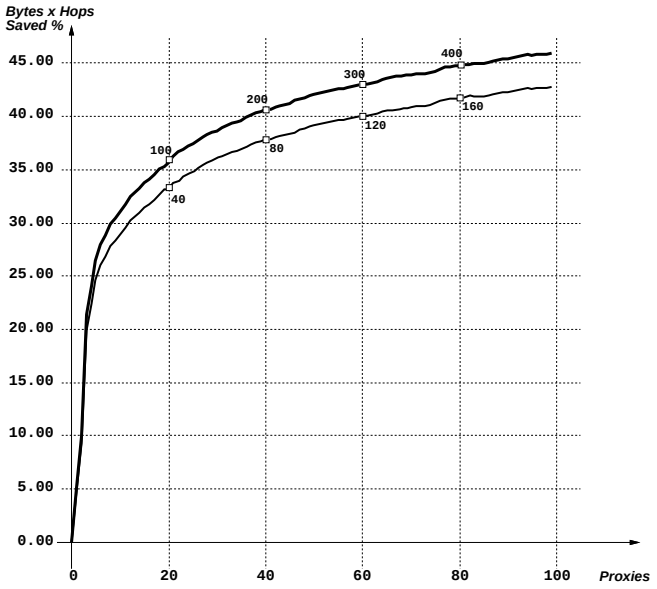

Figure 4: Bandwidth saved as a result of dissemination

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.

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 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.

4.2 System Model and Simulation

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^*. Furthermore, 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 $0 < T_p \leq 1$.

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. 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. 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 detailed in [Bes95b].

4.3 Baseline Model Results

Figure 5 (left) 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

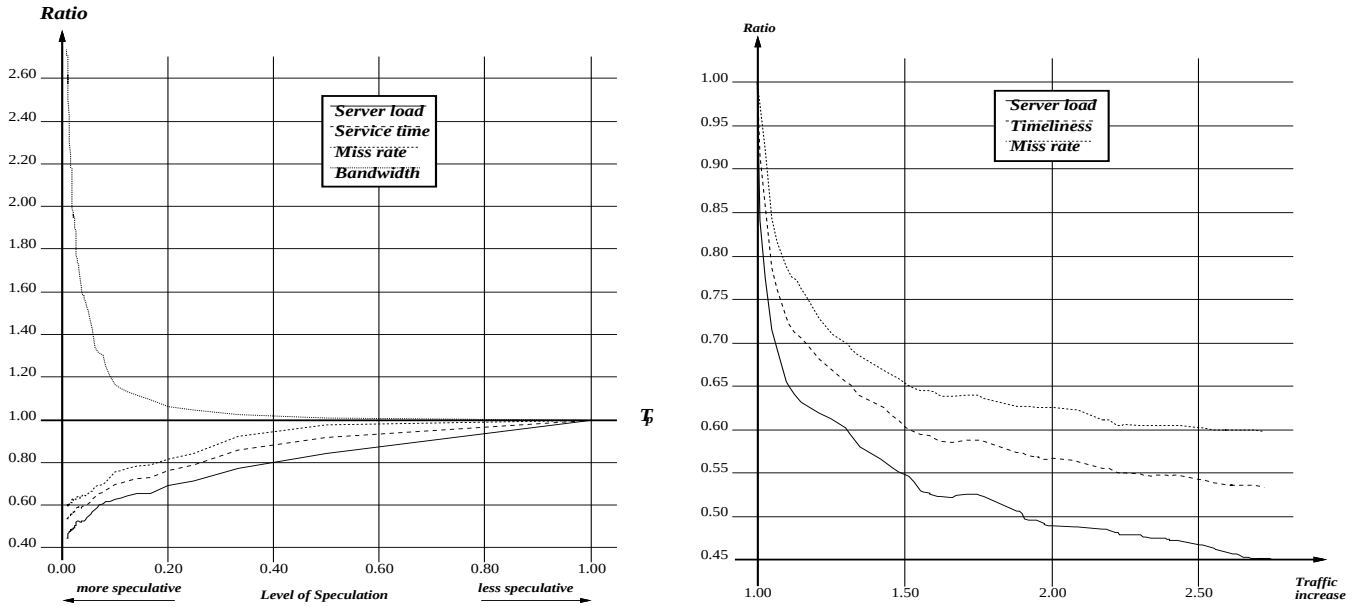

Figure 5: Performance of speculative service

the resulting increase in traffic. Figure 5 (right) 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. Figure 5 suggests 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.

Acknowledgments

I would like to thank all members of the OCEANS research group `http://cs-www.bu.edu/groups/oceans` for their stimulating discussions and their feedback on the work overviewed in this paper. The performance evaluation of the data dissemination protocol was performed by Carlos Cunha, to whom I am grateful.

References

[BC95] Azer Bestavros and Carlos Cunha. Replica placement for distributed information systems. Technical Report (In Progress), Boston University, CS Dept, Boston, MA 02215, November 1995.

[Bes95a] 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.

[Bes95b] Azer Bestavros. Using speculation to reduce server load and service time on the www. In *Proceedings of CIKM'95: The 4^{th} ACM International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1995.

[BS93] Azer Bestavros and Dimitrios Spartiotis. Probabilistic Job Scheduling for Distributed Real-time Applications. In *Proceedings of the First IEEE Workshop on Real-Time Applications*, New York, NY, May 1993.

[Col] RSA130 Collaboration. FAFNER: Factoring via Network-Enabled Recursion. (`http://cooperate.com/cgi-bin/FAFNER/factor.pl`).

[GS95] James Gwertzman and Margo Seltzer. The case for geographical push caching. In *In Proceedings of HotOS'95: The Fifth IEEE Workshop on Hot Topics in Operating Systems*, Washington, May 1995.

[pc95] Barry M. Leiner (program chair). Workshop on Middleware (in conjunction with sigcomm'95), March 1995. Cambridge, MA.

[Sta89] John Stankovic. Decentralized decision making for task allocation in a hard real-time system. *IEEE Transactions on Computers*, March 1989.