In Proceedings of SPDP'95: The 7th IEEE Symposium on Parallel and Distributed Processing, San Anotonio, TX, Oct 1995.

Demand-based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems*

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

Abstract

Research on replication techniques to reduce traffic and minimize the latency of information retrieval in a distributed system has concentrated on client-based caching, whereby recently/frequently accessed information is cached at a client (or at a proxy thereof) in anticipation of future accesses. We believe that such myopic solutions—focussing exclusively on a particular client or set of clients—are likely to have a limited impact. Instead, we offer a solution that allows the replication of information to be done on a global supply/demand basis. We propose a hierarchical demand-based replication strategy that optimally *disseminates information from its producer to servers that are closer to its consumers. The level of dissemination depends on the relative popularity of documents, and on the expected reduction in traffic that results from such dissemination. We used extensive HTTP logs to validate an analytical model of server popularity and file access profiles. Using that model we show that by disseminating the most popular documents on servers closer to clients, network traffic could be reduced considerably, while servers are load-balanced. We argue that this process could be generalized to provide for an automated server-based information dissemination protocol that will be more effective in reducing* both *network bandwidth and document retrieval times than client-based caching protocols.*

1 Introduction

Current protocols for accessing distributed information systems are inefficient, wasteful of bandwidth, and exhibit a large degree of performance unpredictability. Furthermore, the growing disparity between the volume of data that becomes available and the retrieval capacity of existing networks is a critical issue in the design and use of future distributed information systems. Perhaps the best "living" proof of the seriousness of this problem is the fate of many information servers on the Internet: they are unreacheable as soon as they become popular. In a recent solicitation [7] from the National Science Foundation's ES and MSA programs, the following research topics were deemed critical for projected applications of the National Information Infrastructure (NII):

- ◇ *New techniques for organizing cache memories and other buffering schemes to alleviate memory and network latency and increase bandwidth.*
- ◇ *Partitioning and distribution of system [resources] throughout a distributed system to reduce the amount of data that must be moved.*

To tackle the abovementioned challenge, we propose a novel protocol for improving the availability and responsiveness of 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.

The basic idea of our protocol is to *off-load* popular servers by duplicating (on other servers) only a small percentage of the data that such servers provide. The extent of this duplication (how much, where, and on how many sites) depends on two factors: the popularity of the server and the expected reduction in traffic if dissemination is done in a particular direction. In other words, our protocol 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.

There has been quite a bit of research on caching and replication to improve the availability and performance of scalable distributed file systems [9]. Example systems include the Sun NFS [13], the Andrew File System[10], and the Coda system [14]. Recently, there have been some attempts at extending caching and replication to distributed information systems (*e.g.* FTP and HTTP). Caching to reduce the bandwidth requirements for the FTP protocol on the NSFNET has been studied in [6]. 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 stor-

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

age costs for the delivery of multimedia home entertainment was suggested in [12]. Multi-level caching was studied in [11], where simulations of a two-level caching system is shown to reduce both network and server loads. In [3], 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 [5]. The proposed research work of Gwertzman and Seltzer sketched in [8] is the closest to ours. In particular, they propose the implementation of what they termed as geographical push-caching, which allows servers to decide when and where to cache information based on geographical information (such as the distance in actual miles between servers and clients). Their work provides no information about resource allocation strategies and seems to be static.

2 Server Log Analysis

Figure 1 shows the frequency of remote access of individual 256KB document[1] blocks available through the `cs-www.bu.edu` HTTP server. The horizontal axis of figure 1 depicts these blocks in a decreasing *remote popularity*. Only those blocks accessed at least once are shown. 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.

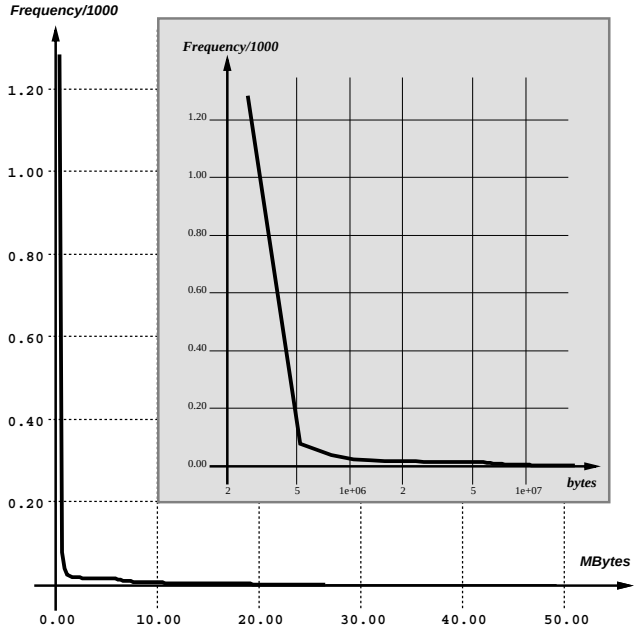

Figure 1: Popularity of various data blocks

Figure 2 shows the cumulative probability of access, where the horizontal axis depicts the various data blocks in a decreasing order of *remote popularity*. 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!

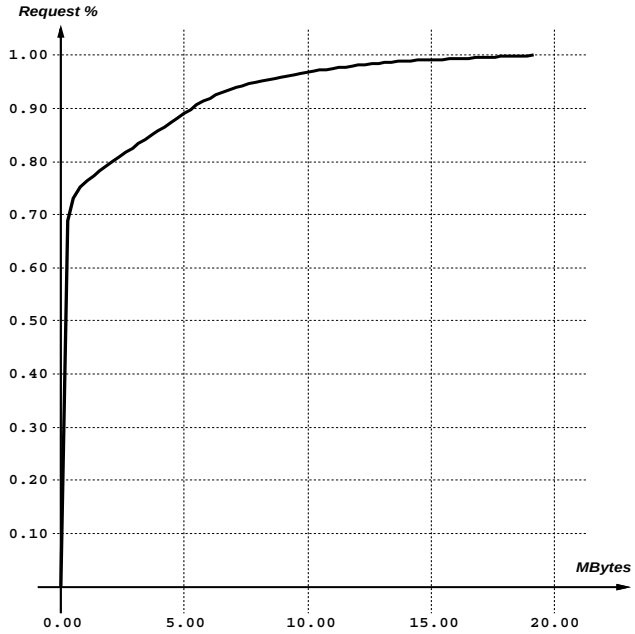

Figure 2: Cumulative popularity of data blocks

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 3 shows the percentage of the remote bandwidth that would be saved if various block sizes of decreasing popularity are serviced at an earlier stage.

The above observations have been corroborated by analyzing the HTTP logs of the Rolling Stones server `http://www.stones.com/` from November 1, 1994 to February 19, 1995. Unlike the `cs-www.bu.edu` HTTP, this server is intended to serve exclusively remote clients. It is a very popular server with more than 1 GigaByte of multimedia information per day (exactly 1,009,146,921 Bytes/day) serviced to tens of thousands (distinct) clients (namely 60,461 clients retrieved at least 10 files during the duration of the analysis). Figure 4 shows the frequency of access for all the documents that have been serviced at least once. Figure 5 shows the percentage of the remote bandwidth that would be saved if various block sizes of decreasing popularity are serviced at some other server. Of the 400 MBytes of information accessed at least once[2] during the analysis period, only 21 MBytes (5.25%) were responsible for 85% of the traffic.

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

[2] Notice that the total number of bytes *available* from that server is much larger than 400 MBytes.

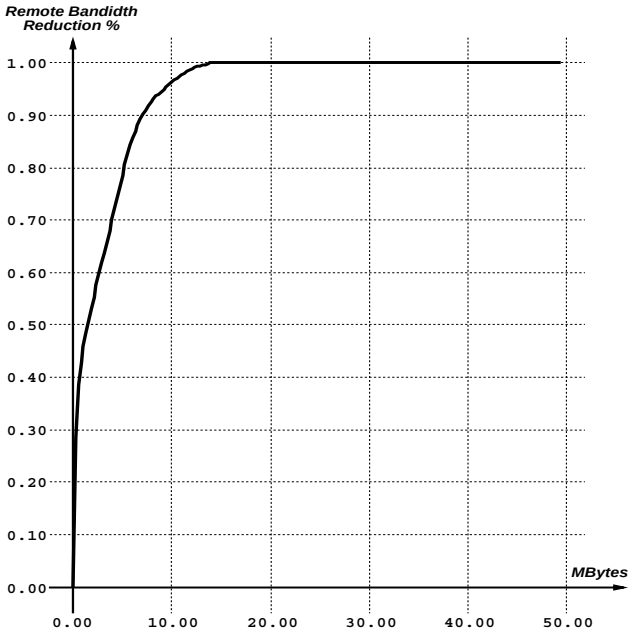

Figure 3: Bandwidth savings against proxy size

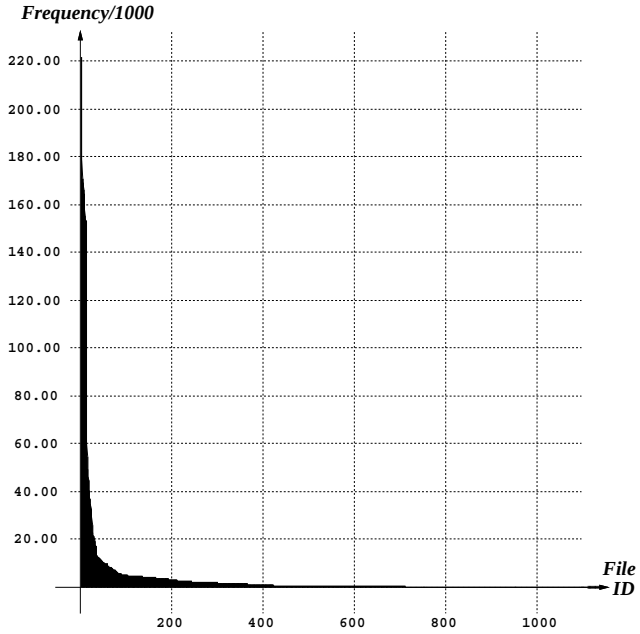

Figure 4: Access frequency for `www.stones.com` server

A closer look at the logs of the HTTP server at `cs-www.bu.edu`, which is a typical example of servers that cater primarily to local clients, reveals that there are three distinct classes of documents: *locally popular documents*, *remotely popular documents*, and *globally popular documents*. Figure 6 shows the ratio of remote-to-local (and local-to-remote) accesses for each one of the 974 documents accessed at least once during the analysis period. From this figure we notice that 99 documents had a remote-to-local access ratio larger than 85%. We call these *remotely popular documents*. Also, we notice that more than 510 documents had a remote-to-local access ratio smaller than 15%. We call these *locally popular documents*. We call the remaining 365 documents *globally popular documents*.

We monitored (on a daily basis) the *date of last update* of remotely, locally, and globally popular documents for a period of one month (from January 17 to February 17). 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 the updates were confined to a very small subset of documents. We call these documents *mutable* documents. The classification of documents into globally/remotely/locally popular and into mutable/immutable documents could be easily done by servers. Such a classification could be used by servers to decide which documents to disseminate. It is interesting to note that our update frequency measurements (like those discussed in [8]) depart significantly from the synthetic workload used in recent WWW coherence studies [15]. This has implications regarding the overhead of maintaining the coherence of disseminated documents. In particular, given the rarity of popular documents updates, we argue that simple protocols such as the Time-To-Live (TTL) and Alex [4] protocols are attractive alternatives to the high-overhead invalidation-based protocols [15].

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

3 System Model and Analysis

We model the WWW (Internet) as a hierarchical set of clusters. A cluster consists of a number of servers. One of these servers acts as a *service proxy* (or front-end) for the cluster. The notion of a service proxy is similar to that of a client proxy, except that the proxy acts on behalf of a cluster of servers rather than a cluster of clients.

In our model, a cluster corresponds to an institution or an organization. For example, we may model all the WWW servers at Boston University as servers within a cluster, with a particular machine (say `www.bu.edu`) acting as a service proxy for the whole institution. In the meantime, one of the servers in the Boston University cluster (say `cs-www.bu.edu`) may itself be a service proxy for another cluster of servers (say the various LANs within the CS department). This correspondence between clusters and organizations is only for the purpose of illustration. 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.

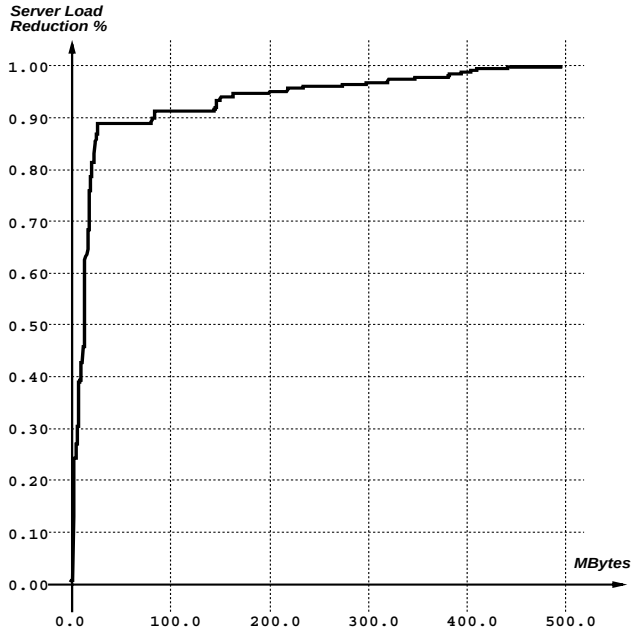

Figure 5: Bandwidth reduction for `www.stones.com`

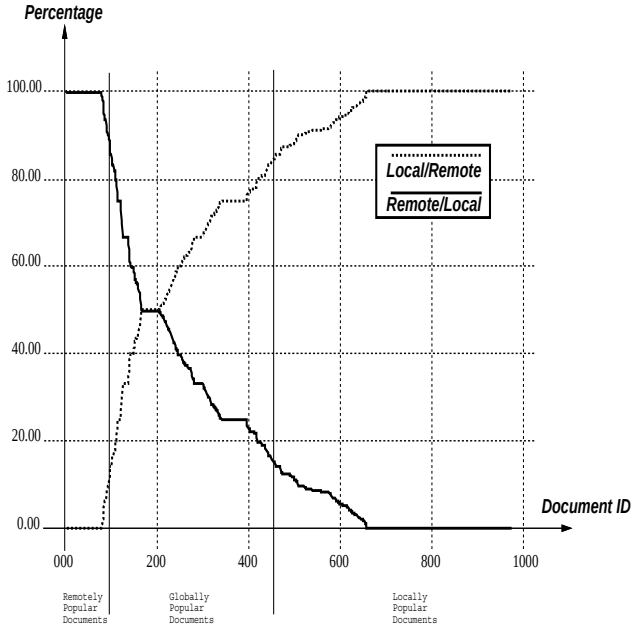

Figure 6: Local vs remote popularity of documents

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. In this paper, and without loss of generality, we assume that each server belongs to exactly one cluster, and thus has only one service proxy.

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 $\mathcal{S}_0$. An example of this probability function is shown in figure 3. 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.

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

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 $j = 1, 2, \ldots, n$:

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

where $h_j(B_j)$ denotes the Probability Density Function corresponding to $H_j(B_j)$. In equation 2 the value of k is chosen so as 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.

3.1 Exponential Popularity Analysis

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 Probability Density Function corresponding to $H_i(b)$ is $h_i(b)$, where

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

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

$$\begin{aligned} h_j(B_j) &= k \cdot \frac{\sum_{i=1}^{n} R_i}{R_j} \\ \lambda_j e^{-\lambda_j . B_j} &= k \cdot \frac{\sum_{i=1}^{n} R_i}{R_j} \\ B_j &= \log \left(\frac{\lambda_j}{k} \frac{R_j}{\sum_{i=1}^{n} R_i} \right)^{\frac{1}{\lambda_j}} \end{aligned} \tag{4}$$

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 $B_0 \leq B_1 + B_2 + \ldots + B_n$.

$$\begin{aligned} \sum_{i=1}^{n} B_i &= B_0 \\ \sum_{i=1}^{n} \log \left(\frac{\lambda_i}{k} \frac{R_i}{\sum_{i=1}^{n} R_i} \right)^{\frac{1}{\lambda_i}} &= B_0 \\ \left(\frac{1}{k \sum_{i=1}^{n} R_i} \right)^{\sum_{i=1}^{n} \frac{1}{\lambda_i}} \cdot \prod_{i=1}^{n} (\lambda_i R_i)^{\frac{1}{\lambda_i}} &= e^{B_0} \end{aligned}$$

which results in the following expression for k.

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

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 done in a variety of ways, which we discuss later in our protocol. For now, it suffices to say that these parameters could be easily and efficiently computed from the server logs. As a matter of fact, figures 1, 2, 3 were produced by programs 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 (say every week).

3.2 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 $i = 1, 2, \ldots, n$. We call this the *equally effective duplication* assumption. Substituting in equation 5, we get:

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

Substituting for k into equation 4, we get:

$$\begin{aligned} B_j &= \log \left(\frac{\lambda}{\frac{\lambda}{\sum_{i=1}^{n} R_i} \left(\frac{\prod_{i=1}^{n} R_i}{e^{\lambda B_0}} \right)^{\frac{1}{n}}} \frac{R_j}{\sum_{i=1}^{n} R_i} \right)^{\frac{1}{\lambda}} \\ B_j &= \frac{B_0}{n} + \frac{1}{\lambda} \log \frac{R_j}{\sqrt[n]{\prod_{i=1}^{n} R_i}} \end{aligned} \tag{6}$$

Under the equally effective duplication assumption, equation 6 suggests that popular servers are allocated *extra* storage capacity on the proxy. This extra storage depends on two factors, namely $\frac{1}{\lambda}$, 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 in the paper), 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. We call this the *equally popular servers* assumption. Substituting in equation 5, we get:

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

Substituting for k into equation 4, we get:

$$B_j = \log\left(\frac{\lambda_j}{\frac{1}{n}\left(\frac{\prod_{i=1}^{n}\lambda_i^{\frac{1}{\lambda_i}}}{e^{B_0}}\right)^{\sum_{i=1}^{n}\frac{1}{\lambda_i}}}\frac{R}{\sum_{i=1}^{n}R}\right)^{\frac{1}{\lambda_j}}$$

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

Under the equally popular servers assumption, equation 7 suggests that servers, whose data are accessed more uniformly (*i.e.* servers with a smaller value for 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, then equation 7 suggests that servers with a intermediate values for λ should be favored. For example, figure 7 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 $B_0 = \frac{1}{\lambda_i}$, for $1 \leq i \leq n$ and $i \neq j$. Figure 8 depicts the optimal allocation when $B_0 = 10\frac{1}{\lambda_i}$.

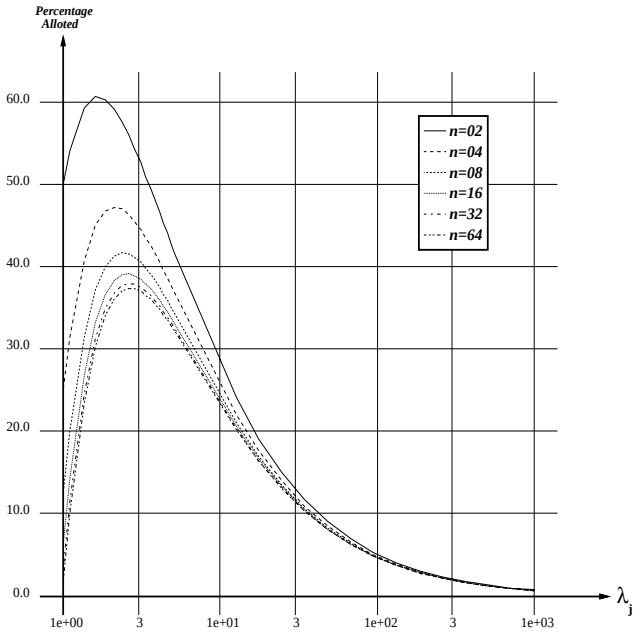

Figure 7: Storage allocation for $R_i = R$ and $B = \frac{1}{\lambda_i}$

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

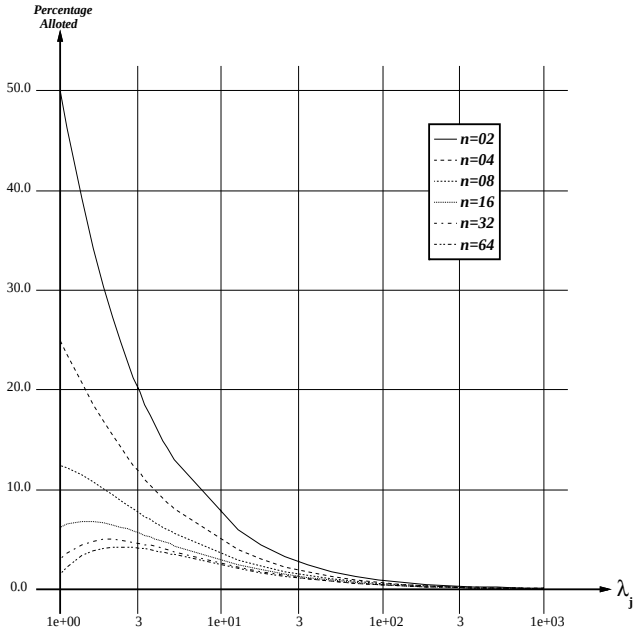

Figure 8: Storage allocation for $R_i = R$ and $B = 10\frac{1}{\lambda_i}$

R_i and λ_i. From equation 5, we get:

$$k = \frac{\lambda}{n} \cdot e^{-\frac{\lambda}{n}B_0}$$

Substituting in 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 `cs-www.bu.edu` server is only 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, of course, 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, if (or when) it is determined that the proxy is overloaded, then B_0 could be reduced, thus forcing more of the requests back to the servers.

4 The DDD-WWW Protocol

We present our protocol at a high level by describing its components at the clients and servers. Notice that we make no distinction between servers and proxies, since for all practical purposes, if a client *knows* that a particular document has been disseminated to a particular proxy, then it could simply use that proxy as the server, from which to fetch the document.

Client Fetching Protocol: DDD-WWW requires clients to maintain a *URL translation lookaside buffer*, where *recent* URL chasings are cached. Notice that the upkeep of very similar information is needed *anyway* by the caching protocols employed at the client. The first step in fetching a `URL` involves looking up the URL translation lookaside buffer (see the *Cal Mapped*() and *WhereMapped*() functions in figure 9). If a mapping is found, then the client uses it as the initial (seed) `EffectiveURL`. The second step involves chasing the document until a valid `EffectiveURL` is *found*, in which case the document is fetched.[4] Figure 9 shows these steps.

```
ClientFetch(URL)
    State ← Unresolved;
    TempURL ← URL;
    If (Mapped(URL))
        TempURL ← WhereMapped(URL);
    While (State == Unresolved){
        EffectiveURL ← TempURL;
        ServerReply ← ServerQuery(EffectiveURL) ;
        State ← ServerReply.Ack ;
        TempURL ← ServerReply.NextURL ; }
    If (State == Found)
        Fetch(EffectiveURL) ;
    Else
        FailFetch("Document not found.") ;
```

Figure 9: Client protocol for fetching a URL

[4]Notice that our protocol does not preclude the `EffectiveURL` from pointing to the local cache of the client itself (whether at the *session*, *machine*, or *LAN* levels [2]). This makes for a natural integration of producer-based dissemination and consumer-based caching of documents.

Server Query Protocol: DDD-WWW requires servers to maintain a (possibly one-to-many) mapping between local URLs and the URLs of corresponding disseminated copies. The first step for a server to respond to a query from a client involves looking up this mapping (see the *Cal Disseminated*() and *WhereDisseminated*() functions in figure 10). If the document in question has been disseminated, then the server simply returns to the client the URL of the (best) disseminated copy, otherwise it returns an acknowledgment that indicates whether the document is availabe (*Found* or *Invalid*). Figure 10 shows these steps.

```
ServerQuery(URL)
    If (Disseminated(URL)){
        Reply.Ack ← Unresolved ;
        Reply.NextURL ← WhereDisseminated(URL) ; }
    Else
        If (Available(URL))
            Reply.Ack ← Found ;
        Else
            Reply.Ack ← Invalid ;
```

Figure 10: Server/Proxy protocol for Servicing a URL

Document Dissemination Protocol: The last component of DDD-WWW is responsible for the dissemination of popular documents between servers. In order to so, each server must collect statistics on the *popularity* of each document it maintains. We denote by $\mathtt{F}(\mathcal{S}_i)$ the set of all files (documents) available at server $\mathcal{S}_i$. This includes duplicated document that the server keeps on behalf of other servers.

We assume that each server keeps logs of the client requests that were honored at that server. Using these logs, the server is capable of computing the popularity of each document it maintains—namely, how many times (per unit time) a document was serviced. Let $Freq(\mathcal{S}_i, \mathtt{f})$ denote the frequency with which a file `f` was serviced by server $\mathcal{S}_i$ to a non-local client.[5] Let $Home(\mathcal{S}_i, \mathtt{f})$ denote the server that disseminated file `f` to $\mathcal{S}_i$. In particular, if file `f` is local, then $\mathcal{S}_i = Home(\mathcal{S}_i, \mathtt{f})$. Also, let $Proxy(\mathcal{S}_i, \mathtt{f})$ denote the set of servers that are acting as proxies for file `f` of server $\mathcal{S}_i$. $Freq(\mathcal{S}_i, \mathtt{f})$ does not account for the popularity of `f` at $Proxy(\mathcal{S}_i, \mathtt{f})$. Let $Pop(\mathcal{S}_i, \mathtt{f})$ denote the cumulative frequency with which a file `f` was serviced from server $\mathcal{S}_i$ *as well as* from any other server in $Proxy(\mathcal{S}_i, \mathtt{f})$. Figure 11 shows the steps that need to be executed (periodically) by each server (say $\mathcal{S}_j$) so as to propagate the popularity information $Pop(\mathcal{S}_i, \mathtt{f})$, for all servers and files in the system. Function *ReportPop*() communicates the cumulative popularity of a file at a proxy to the server that requested that the file be duplicated at that proxy.

The calculation of $Pop(\mathcal{S}_i, \mathtt{f})$ for all files $\mathtt{f} \in \mathtt{F}(\mathcal{S}_i)$ allows each server $\mathcal{S}_i$ to compute the remote popular-

[5]The term "non-local client" is loosely defined to be a client outside the cluster of $\mathcal{S}_i$, *i.e.* a client whose requests could be serviced by a proxy.

```
ServerStats()
    Forall f ∈ F(S_i){
        Pop ← Freq(S_j , f) ;
        Forall s ∈ Proxy(S_j ,f){
            Pop ← Pop + Pop(s, f) ;
            Pop(S_j , f) ← Pop ;}
        If (S_j ≠ Home(S_j , f))
            ReportPop(f, Pop, Home(S_j , f)) ;}
```

Figure 11: Periodic process to keep popularity profile

ity of the various *blocks* in the system (see figures 1 and 2), and thus estimate the value of λ_i used in our analytical study to characterize the $H_i(b)$ distribution. Also, the value of $Pop(\mathcal{S}_i, \texttt{f})$ for all files $\texttt{f} \in \texttt{F}(\mathcal{S}_i)$ could be combined to evaluate the total number of bytes per unit time serviced by (or on behalf of) $\mathcal{S}_i$, and thus estimate the value of R_i used in our analytical study to characterize the relative popularity of a server in a given cluster. The process of deciding what to disseminate from the servers in a cluster to the proxy of that cluster is straightforward.

5 Conclusion

There are many reasons for advocating the development of an automated information dissemination protocol as a way of controlling traffic as opposed to simply increasing the available bandwidth in the system. First, adding servers (*i.e.* proxies) to the internet is much cheaper than adding (upgrading) internet links [6]. Second, increasing the available bandwidth is a temporary solution; it's only a matter of time before the added bandwidth is consumed by the ever increasing number of users. Demand-based dissemination of information from producers to consumers is not a new idea: it is used in the retail of commodities, newspaper distribution, among other things. In this paper, we proposed to use the same philisophy for distributed information systems. We presented an analytical model (supported by data from actual logs of typical institutional and commercial servers) that demonstrates how such dissemination could be done, both efficiently and with minimal changes to the prevailing client-server infrastructure of the Internet.

Acknowledgments: I would like to thank M. Crovella, A. Heddaya, and all members of the *Oceans* group `http://cs-www.bu.edu/groups/oceans` for the many discussions and feedback on this work. Also, I would like to thank S. Fitch and S. Sclaroff for providing me with the Rolling Stones logs.

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, 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.

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

[4] V. Cate. Alex — a global filesystem. In *Proceedings of the 1992 USENIX File System Workshop*, Ann Arbor, MI, May 1992.

[5] 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.

[6] 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.

[7] Michael Foster and Robert Jump. NSF Solicitation 94-75. STIS database, May 1994.

[8] 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.

[9] 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.

[10] 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.

[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.

[13] 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.

[14] 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.

[15] K. Worrell. Invalidation in large scale network object caches, 1994. Master's Thesis, University of Colorado, Boulder.