Dynamic Parallel-Access to Replicated Content in the Internet

Pablo Rodriguez Ernst W. Biersack
Institut EURECOM
2229, route des Crêtes. BP 193
06904, Sophia Antipolis Cedex, FRANCE
{rodrigue, erbi}@eurecom.fr

(Published in IEEE/ACM Transactions on Networking, August 2002)

Abstract—**Popular content is frequently replicated in multiple servers or caches in the Internet to offload origin servers and improve end-user experience. However, choosing the best server is a non-trivial task and a bad choice may provide poor end user experience. In contrast to retrieving a file from a single server, we propose a parallel-access scheme where end users access multiple servers at the same time, fetching different portions of that file from different servers and reassembling them locally. The amount of data retrieved from a particular server depends on the resources available at that server or along the path from the user to the server. Faster servers will deliver bigger portions of a file while slower servers will deliver smaller portions. If the available resources at a server or along the path change during the download of a file, a dynamic parallel-access will automatically shift the load from congested locations to less loaded parts (server and links) of the Internet. The end result is that users experience significant speedups and very consistent response times. Moreover, there is no need for complicated server selection algorithms and load is dynamically shared among all servers. The dynamic parallel-access scheme presented in this paper does not require any modifications to servers or content and can be easily included in browsers, peer-to-peer applications or content distribution networks to speed up delivery of popular content.**

I. Introduction

In order to offload popular servers and improve end user experience, copies of popular content are often stored in different locations. With network caching, geographically dispersed caches store copies of the documents required by their clients. With mirror site replication, documents from a primary site are proactively replicated at secondary sites. With peer-to-peer applications, users fetch and store content from other peer-users in a effort to share the load among nodes at the edge of the network and bring content closer to the users.

When a copy of the same document exists at multiple servers, choosing the server that provides the best response time is not trivial and the resulting performance can dramatically vary depending on the server selected [15] [**?**] [30]. Even when the fastest server has been selected, its performance can fluctuate during a download session, resulting sometimes in a poor response time at the end of the download. Rather than trying to choose the fastest available server, users can experience a better and more uniform performance by connecting to several servers that have an exact copy of the document: Instead of downloading the entire document from one server, a user downloads different parts of the same document from each of the servers in parallel. Once all the parts of the document are received, the user reconstructs the original document by reassembling the different parts.

In this paper we propose a parallel-access scheme to download content from multiple servers at the same time. We consider two different parallel-access schemes, (i) **history-based TCP** parallel-access, and (ii) **dynamic TCP** parallel-access . With a history-based parallel-access, clients specify *a-priori* which part of a document must be delivered from each mirror server, e.g., server one sends the first half of the document, and server two sends the second half. The portion of a document delivered by one server should be proportional to its service rate, thus, a slow server will deliver a small part of the document while a fast server will deliver a big part of the document. To calculate the portion of a document assigned to each server, a history based parallel-access uses a database of previous server rates, which is refreshed periodically, e.g. every few minutes. A history-based parallel-access scheme can speedup the download of a document when the network/server conditions are stable or easily predictable. However, when network/server conditions change rapidly, server rates are hard to predict and a history-based parallel-access performs poorly.

With a dynamic parallel-access, on the other hand, a client partitions a document into a large number of small blocks. The client first requests a different block from each server. Whenever a server finishes transmitting a block, the client issues a new request for a block that has not yet been requested from any other server. The same process is repeated for each block until all blocks are fully received. Note that at any given point in time all servers are kept busy sending a block (except for the idle times between block requests). When the client receives all blocks it reassembles to reconstruct the whole document.

There are several advantages to using a dynamic parallel-access. First, since the block size is small, a dy-

namic parallel-access can easily adapt to changing network/server conditions. The servers contacted share the load in a way that is proportional to the available resources at the server or in the path from the user to the server, therefore, performing automatic load balancing. Fast servers will deliver bigger portions of a document while slow servers will deliver smaller portions. This automatic load balancing is performed without any a-priori information about server rates. Second, since the client is using several connections to different servers, a parallel-access is more resilient to congestion and failure in the network/servers than connecting to a single server. Load is automatically shifted from congested parts of the Internet to other parts with more abundant resources. Third, the server selection process is eliminated since clients connect to all available servers with a document copy. Fourth, the throughput seen by the client increases. Ideally, the total throughput seen by the client is equal to the sum of the bandwidths from each individual server to the client.

A parallel-access, however, has some additional overhead compared a single access. There is an additional overhead incurred when opening multiple connections and extra traffic generated to perform block requests (for a complete discussion on the costs a parallel-access see Section VI-A). To minimize the impact of these costs, a parallel-access must be employed only with documents of a certain size, e.g. in the order of several hundreds of Kbytes. In the Web the number of objects of this size is relatively small, thus, instead of suggesting a parallel-access for general Web downloads, we propose a dynamic parallel-access for other content types such as large documents, software downloads, music, video clips, or images.

Using analytical and experimental results we evaluate the performance of a parallel-access scheme. We study the parallel-access behavior under different number of servers, document sizes and various network/server conditions including high-speed links as well as congested slow links. In addition, we use parallel-access implementation to better understand its advantages and limitations in a real deployment scenario and better define the scenarios where a parallel-access is most beneficial.

The rest of the paper is organized as follows. Section II presents and analyzes a history based parallel-access. In Section III we present the dynamic parallel-access and demonstrate that it offers dramatic speedups for different document sizes, number of servers, and network conditions. Section IV considers a dynamic parallel-access where a client is connected through a modem link. Section V compares a dynamic parallel-access with a scheme where the client opens multiple parallel connections to the same server, and also studies the impact of request pipelining. Section VI discusses several important issues for the deployment of a parallel-access. Section VII discusses related work and section VIII concludes the paper.

II. History Based Parallel-Access

A history-based parallel-access uses information about the previous transmission rates between the client and every mirror server. It needs this information to decide a-priori which part of a document should be delivered by each server. The client divides a document into M disjoint blocks, and requests one block from every mirror server. Let μ_i be the transmission rate for server i, $1 \leq i \leq M$ and let S be the document size. If $\alpha_i S$ is the size of the block delivered by server i then $T_i = \frac{\alpha_i S}{\mu_i}$ denotes the download time of this block. To achieve a maximum speedup, all servers must finish transmitting their block at the same time, thus, $T_i = T_j$ for all $i, j \in \{1, ..., M\}$. When all servers transmit their block at the same time, there are no servers that stop transmitting before the document is fully received. The rate μ_p achieved with parallel-access when all servers keep sending useful data until the document is fully received, is equal to the sum of the individual rates of all servers, i.e. $\mu_p = \sum_{i=1}^{M} \mu_i$. Fast servers send a bigger portion of the document, while slow servers send smaller portions. To achieve a maximum speedup, the size $\alpha_i S$ of the block sent by server i, must be equal to $\alpha_i S = \frac{\mu_i}{\mu_p}$.

A history-based parallel-access needs to keep a database with information about the previous rates from the different servers to the receiver in order to estimate the rate μ_i to every server. Instead of having one database per-client, a single database could be shared by a group of receivers connected through a proxy-cache. The database is actualized every time that a client connects to a server or can be updated periodically with an automated probing from the proxy.

A. Experimental Setup

To evaluate history-based parallel-access we have implemented this scheme as a JAVA client program that takes as input parameters the URLs and uses a database of previous rates from every mirror server to the client. The JAVA client performs a history-based parallel-access for the requested document, saves the document locally, and records the time it took to download the document. To calculate the size of every block, clients need to know the total document size S. To obtain S, the parallel-access JAVA client polls the servers using a HTTP request at the beginning. The document size could also be pre-recorded in a proxy cache or given to the client through a DNS server, thus, avoiding additional RTTs to poll the servers.

To analyze the performance of a history-based parallel-access scheme, we performed several experiments using mirror servers in the Internet. In particular, we considered several mirror servers of the Squid Web Page (http://squid.nlanr.net/) [32]. Figure 1 shows a network map with the mirror servers considered and the bandwidth of the slowest link in every path as given

by pathchar [18]. The Java client is always located at EURECOM, France. Since the servers are situated in different countries and given that the connection from our institution (EURECOM) into the Internet has a high access rate, a parallel-access connection from a EURECOM client to the mirror servers is likely to be bottleneck-disjoint.

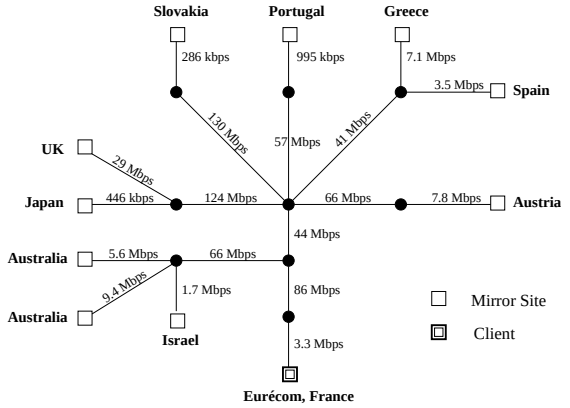


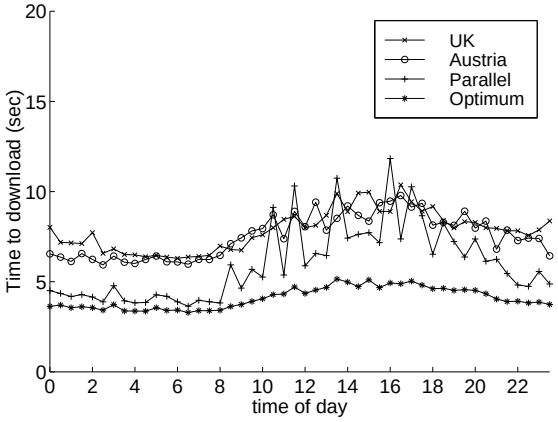
Fig. 1. Mirror servers for the Squid home page. Client is located at EURECOM, France.

We evaluated a history-based parallel-access scheme every 15 minutes, making sure that different experiments do not overlap. We run the experiments 24 hours a day during a 10-day period and averaged over the 10-day period.

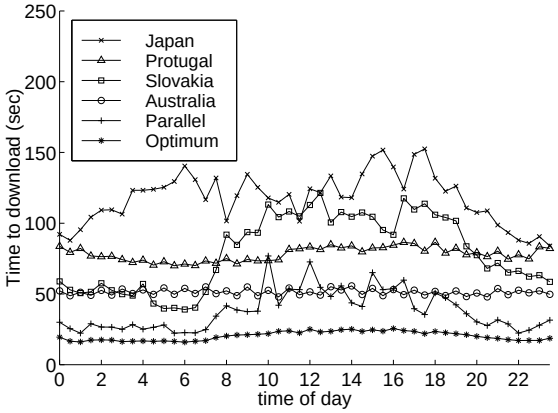
B. Analysis of the Results

Next, we present the performance results of a history-based parallel-access where a client at EURECOM requests a document of 763 KBytes from two servers (Austria and UK), which have average transmission rates between 80-100 Kbps. The document requested is the gzipped beta version of the SQUID 1.2 software [32]. The database with the previous rates from the client to every server is updated when the JAVA client performs a request for the document, that is every 15 minutes. The client assumes that the average rate μ_i offered by every server will be equal to the rate obtained 15 minutes before.

In Figure 2 we show the download time obtained using a history-based parallel-access, and the download time obtained using an individual connection to every server. We see that during the nights, when network conditions do not vary much, a history-based parallel-access can efficiently estimate the average rate offered by every server and allows to significantly decrease the download time compared to the situation where the client accesses a single server. However, during day-time, network conditions rapidly change and estimating the rate to every server by using the previously achieved rates, results in poor estimates. Thus, the download times obtained with a history-based parallel-access can be higher than the download times when clients access



(a) $S = 763$ KB, $M = 2$.



(b) $S = 763$ KB, $M = 4$.

Fig. 2. History-based parallel-access .

a single server. Similar performance for a history-based parallel-access is also obtained for a different set of mirror servers (Figure 2(b)). In figure 2 we also show the optimum download time. The **optimum** download time is defined as the download time achieved by a parallel-access scheme where all servers send useful information until the document is fully received. To calculate (a-posteriori) the optimum download time, the average rates obtained from every server after the reception of a document are used.

To improve the performance history-based parallel-access schemes, the database could be refreshed more frequently during day-time and other more sophisticated estimation algorithms could be used. However, finding the right refresh period and a good algorithm to estimate the rates is not an easy task. In next section, we present another parallel-access scheme that does not require any past information and does not need to estimate the rates to the servers. Instead, the scheme dynamically

adapts to the changing network conditions in real-time.

III. Dynamic Parallel-Access

We consider now a parallel-access scheme that uses dynamic requests between a client and the servers as the download of the document progresses. With a dynamic parallel access, the document is divided by the client into B blocks of **equal** size. In the case of Web access, a block is specified as a range of bytes in a document, e.g. from byte 100 to byte 200. A block requested can be specified using the HTTP1.1 byte-range header [16]. The dynamic parallel-access scheme proceeds as follows:

- A client first requests one block from every server.
- Every time a client has completely received one block from a server, the client requests from this server another block that has not yet been requested from any other server.
- When the client has received all blocks, it reassembles them to reconstruct the whole document.

Since a client typically issues several requests to the same server during the download of a document, TCP-persistent connections are used between the client and every server to minimize the overhead of opening multiple TCP connections [27].

In order to best exploit the advantages of parallel-access , one needs to keep all servers busy until all blocks have been received. In the dynamic parallel-access scheme outlined above this is not the case.

- Each server will be idle between two consecutive block transfers (see Figure 3). This idle time is referred to as *inter-block* idle time and corresponds to one round-trip time RTT. One can entirely avoid the inter-block idle times by *pipelining* requests for different blocks to the same server: A request for a new block is made before the previous block is fully received, thus keeping the server busy at all the time while there are still blocks to transmit. We will elaborate more on this point in Section IV-A.
- Not all servers terminate at the same time: If there are fewer than M blocks left (where M is the number of servers contacted) that have not been received, some servers will no longer transmit a block. The period of time since there are less than M servers transmitting blocks until the instant the document is fully received is referred to as *termination* idle time. The termination idle time is smaller or equal than $\frac{S}{B}/\mu_s$, where $\frac{S}{B}$ is the block size and μ_s is the rate to the slowest server.

The following points need to be considered when determining the size of the blocks requested:

- The number B of blocks should be chosen to be much larger than the number M of mirror servers that are accessed in parallel.
- Each block should be small enough to provide a fine granularity of striping and ensure that the transfer of the last block requested from each server terminates at about the same time, thus, fully utilizing the server and network resources until complete termination of the document transfer.
- Each block should also be sufficiently large as to keep the inter-block idle time small compared to the download time of a block.

To reconcile the last two points, the document requested via parallel-access must be sufficiently large, i.e. in the order of several hundreds of KBytes.

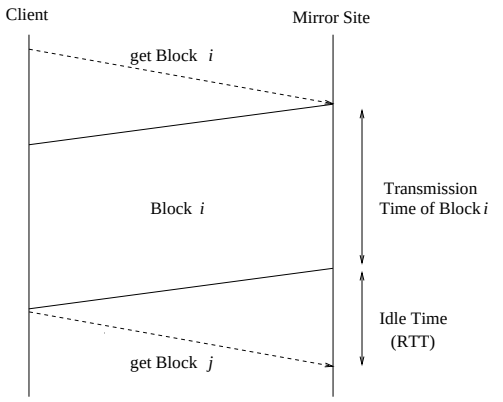


Fig. 3. Dynamic parallel-access : Block request.

A. Analysis of the results

To evaluate the performance of dynamic parallel-access, we implemented the scheme as a JAVA client. The JAVA client takes as input parameters the URLs of the servers with replicated content, performs a dynamic parallel-access, saves the document locally, and records the transmission rate obtained. We evaluate the dynamic parallel-access scheme using the experimental setup described in Section II-A.

Our current implementation of dynamic parallel-access does not consider pipelining to reduce the inter-block idle times. However, the implementation reduces the termination idle time as follows: When there are fewer than M blocks missing, the client requests idle servers to deliver a block that is already requested from another server but that has not yet been fully received. With this approach, clients experience a transmission rate that is at least equal to the transmission rate of the fastest server at the expense of some bandwidth overhead. The bandwidth wasted in the worst case, is equal to $\sum_{i=1}^{M-1} i\frac{S}{B}$, where $\frac{S}{B}$ is the block size. The bandwidth wasted on average is much smaller than the worst case scenario since slow servers that did not complete the transmission of their last block are stopped after the document is fully received.

There are some more ways to minimize the bandwidth wasted[1]

- One can decrease the block size $\frac{S}{B}$ of the blocks requested from slow servers. More general, by dynamically adjusting the block size so that all servers finish

[1] These improvements have not be included in our implementation of the parallel access scheme.

at the same time, the bandwidth wasted would be zero. However, this approach requires accurate bandwidth estimations for each connection at runtime, which may be difficult to obtain [13].

• One can stop slow servers as soon as there are less than M blocks missing and request the missing bytes only of the last blocks from the fastest server.

We now compare the transfer time of dynamic parallel-access with the transfer time of an individual access to every server and the optimum transfer time that would be achieved if there are neither intra-block nor termination idle times. The optimum transfer time provides a lower bound on the transfer time achievable by any implementation of a parallel-access scheme.

We first consider a dynamic parallel-access to download a 763 KByte document, which is replicated in $M = 4$ mirror servers (Figure 4). The actual servers are located in Australia, Japan, Slovakia, and Portugal, to ensure disjoint paths. The average rate to these servers ranges from 5 to 15 KBytes/sec, however, the instantaneous rates greatly fluctuate during the course of the day. We have chosen $B = 30$ blocks.

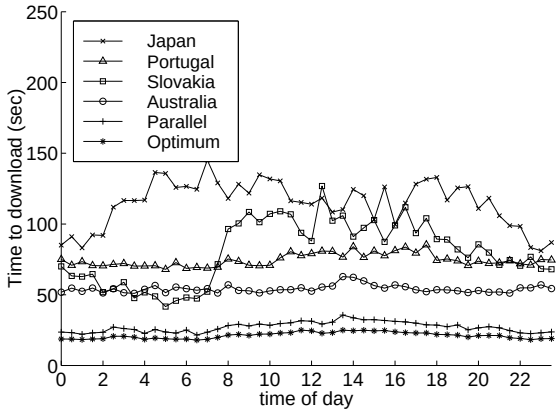


Fig. 4. Dynamic parallel-access . $S = 763$ Kbytes, $B = 30$, $M = 4$.

From Figure 4 we can see that dynamic parallel-access offers significant speedups compared to an individual document transfer from any single server. The transfer time is reduced from 50-150 seconds to 20 seconds during all the periods of the day. Even during highly congested periods, where the network conditions rapidly change, a dynamic parallel-access offers very small transfer times. We also observe that the transfer time of a dynamic parallel-access is very close to the optimum transfer time. Moreover, a dynamic parallel-access that would implement pipelining to avoid inter-block idle times would have performance almost equal to that of the optimum parallel scheme.

Next, we consider the situation with two fast servers (70 KBytes/sec) and two slow ones (10 KBytes/sec). The fast servers are located in Greece and Spain, and the slow ones in Australia and Israel (Figure 5). The document size is smaller than in the previous experiment, $S = 256$ KBytes, and we have also reduced the number of blocks to $B = 20$ to avoid that inter-block idle times account for too high a percentage of the total transfer time (the document requested is the FAQ from SQUID in postscript format [32]).

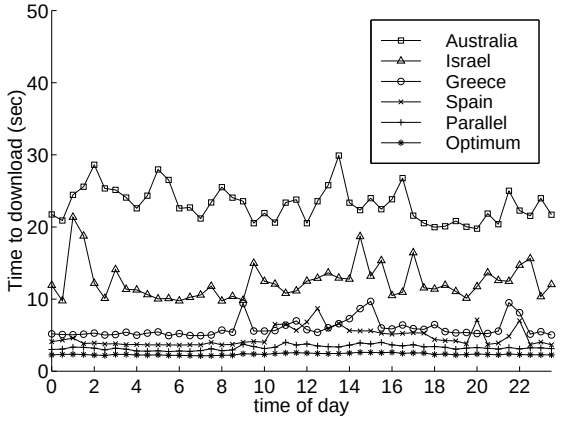


Fig. 5. Dynamic parallel-access . $S = 256$ Kbytes, $B = 20$, $M = 4$.

We can see that a dynamic parallel-access scheme achieves a transfer time that is almost half the transfer time of the fast servers (slow servers only contribute very few blocks and decrease the transfer time of the document by little). The latency benefits may not seem so important if they are compared to the case where a client connects to a fast server (from 4-5 seconds to 2 seconds). However, if the client chooses the wrong server and connects to a slow server, it will end up experiencing transfer times up to 30 seconds.

In the next experiment we consider only two mirror servers (Austria and UK) and perform a dynamic parallel-access for a large document of 5 MBytes (Figure 6). Since both servers have a similar rate, a parallel-access will reduce the transfer time by half. The time to download the document of 5 MBytes from a single server can be up to 80 seconds. Using a dynamic parallel-access, the transfer rate is less than 30 seconds. The difference between the transfer time measured and the one of the optimal parallel-access scheme is due to the inter-block idle times.

B. Performance of Parallel Access to Small Documents

Even though a parallel-access scheme is not intended to be used with small documents, we study the performance of a dynamic parallel-access with small documents of several KBytes in size.

In Figure 7 we see the performance of a dynamic parallel-access scheme for a 10 KByte document. We consider two mirror servers (Spain and Greece) and choose $B = 4$ blocks. We see that a dynamic parallel-access has a download time that varies very little with the time of the day and is in most cases lower than the

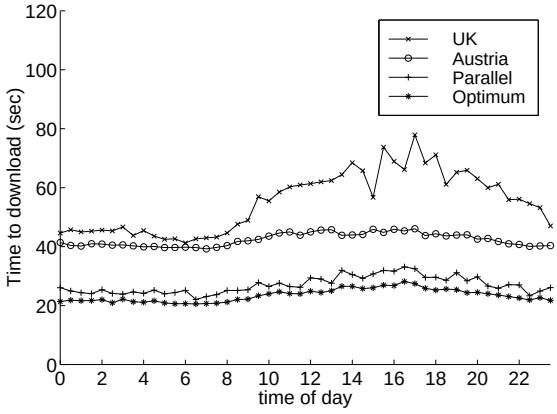


Fig. 6. Dynamic parallel-access . $S = 5$ Mbytes, $B = 40$, $M = 2$.

download time of the fastest server. Compared to the optimum download time, a dynamic parallel-access has a much higher download time since the inter-block idle times account for a high percentage of the total download time.

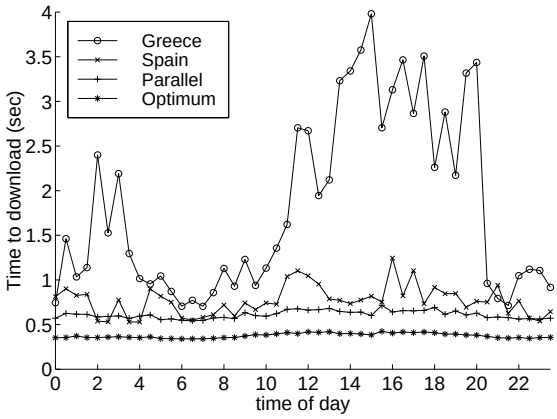
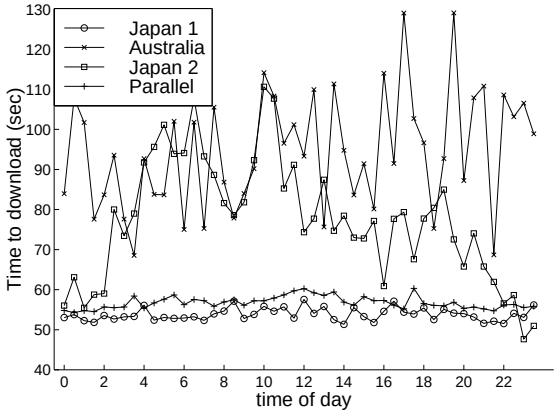


Fig. 7. Dynamic parallel-access . $S = 10KB$, $B = 4$, $M = 2$.

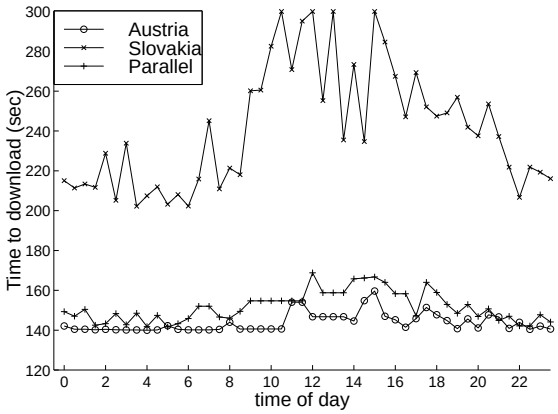
In addition, with small documents the connection setup time may account for a non-negligible portion of the total download time. While a parallel-access scheme speeds up the download time of the document it can not do anything about the connection time. To obtain better performances with a parallel-access, several small documents could be grouped together, i.e. all documents in a Web page, to perform one dynamic parallel-access to the bigger document.

IV. Dynamic parallel-access in case of a shared bottleneck link

In this section we study the performance of a dynamic parallel-access where a client is connected through a modem link, i.e. a low speed access link. In this case the paths from the client to the servers are not bottleneck-disjoint. A single server may already consume all the bandwidth of the modem link. Therefore, when a client uses another server in parallel there is no residual network bandwidth and packets from different servers interfere and compete for bandwidth.



(a) $S = 256$ KBytes, $B = 20$, $M = 3$.



(b) $S = 763$ KBytes, $B = 30$, $M = 2$.

Fig. 8. Retrieval latency for a parallel-access scheme and for an individual-access scheme to every server over a shared bottleneck.

In Figure 8 we consider dynamic parallel-access for a client connected through a modem line at 56 Kbits/sec. We show the download time achieved when connecting to every server individually and when connecting in parallel to all servers using a dynamic parallel-access. In Figure 8(a) we consider two slow servers (Japan 2 and Australia) and a fast server (Japan 1). In case of an individual access to the fast server, the modem bandwidth is fully utilized and the download time varies little during all the periods of the day. For an individual access to one of the slow servers, the rates obtained are about 24 Kbits/sec, which is much lower than the modem speed of 56 Kbits/sec. In this situation, the modem link is not fully utilized and the download time fluctuates de-

pending on the different levels of congestion in the network/servers along the day. A similar effect can be seen in Figure 8(b), where there are two mirror-servers, a fast one and a slow one.

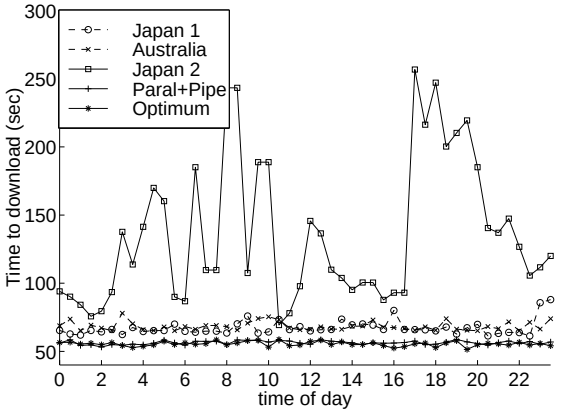
For the dynamic parallel-access, we see that the download time achieved is close to the one of the fastest server, which is limited by the transmission rate of the modem link. The fact that the download time obtained with a dynamic parallel-access is always slightly higher than the download time obtained for the fastest server is due to the inter-block idle times. Next, we study the performance of a dynamic parallel-access that uses pipelining to avoid idle these times.

A. Evaluation of Request Pipelining

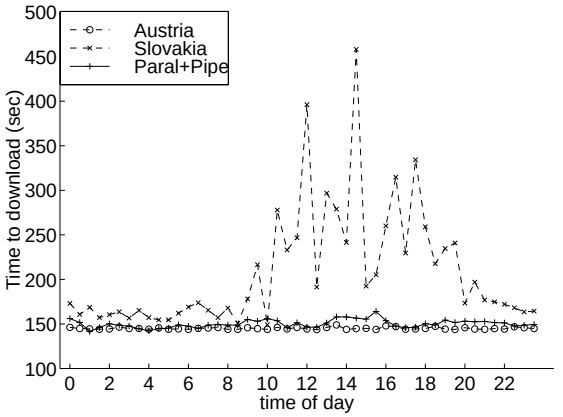
In this section we repeat the previous experiments (Figure 8(a) and 8(b)) and simulate a dynamic parallel-access with block request pipelining. With request pipelining, a new block is requested from a server before the previously requested block is fully received. To fully avoid inter-block idle times, a new block should be requested at least one RTT before the current block is completely received (see Figure 3). Pipelining therefore requires a minimum block size. The block size should such that $\frac{S}{B} > RTT \cdot \mu$. For instance, if the RTT between the client and most distant server is equal to RTT=100 msec and the server has a transmission rate $\mu = 10$ KBytes/sec, the block size must be $\frac{S}{B} > 1KB$.

To estimate the improvement offered by request pipelining, we first measure for each server i, $i \in \{1, ..., M\}$ the download time tr_i obtained by parallel-access without pipelining and the average round-trip time RTT_i. Then, we assume the inter-block idle time to be equal to RTT_i. If server i has transmitted nb_i blocks, we estimate the download time tr_i^p with request pipelining as $tr_i^p = tr_i - (nb_i - 1) * RTT_i$. The estimated time T^p to completely download the document using pipelining is then given as $T^p = \max_{i \in \{1,...,M\}} tr_i^p$.

In Figure 9 we show the estimated download time T^p achieved by a parallel-access with pipelining through a modem link. We observe that the download time of a parallel-access with pipelining is smaller (Figure 9(a)) or equal (Figure 9(b)) than the download time achieved by a single connection to the fastest server. In Figure 9(a) the download time of each server is much smaller than the maximum modem link rate, thus, a single server connection does not fully utilize the modem link. In this case, a parallel-access with pipelining can speedup the transfer of a document compared to a single server connection, and achieve download times that are even smaller than those offered by the fastest server. From Figure 9(a) it is also important to notice, that the transfer time achieved with a dynamic parallel-access using pipelining is almost equal to the optimum download time. Thus, the additional delay that the JAVA implementation of dynamic parallel-access introduces is very small.



(a) $S = 256$ KBytes, $B = 20$, $M = 3$. Pipelining



(b) $S = 763$ KBytes, $B = 30$, $M = 2$. Pipelining

Fig. 9. Retrieval latency for a dynamic parallel-access scheme and for an individual-access scheme to every server through a modem link.

We would like to note that it is easy to implement pipelining. It does not require to calculate the exact RTTs but one simply needs to estimate an upper bound on the RTTs. Using an overestimated (too high a) RTT, pipelining eliminates the idle times with no performance degradation.

V. Dynamic parallel-access vs. Parallel Access to a Single Server

In this section we compare a dynamic parallel-access to multiple mirror-servers with a parallel-access to a *single* server. For a fair comparison, we consider the situation where a single client opens M TCP-parallel connections to the same server and compare this case to a dynamic parallel-access to M servers. Let μ_s be the rate to the slowest server, and μ_f be the rate to the fastest server. If the residual bandwidth of the path from the

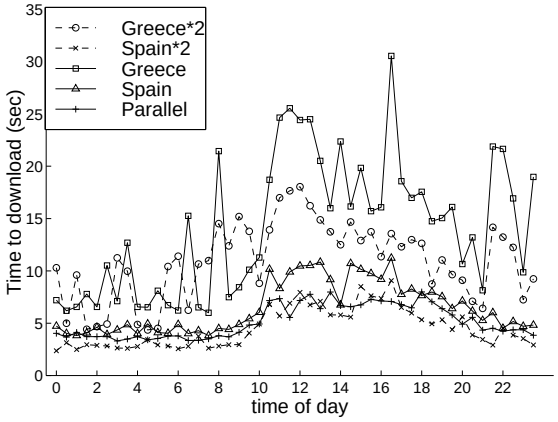
client to the server is large enough, a M-parallel connection to a single server with rate μ_i will have in the best case a transmission rate equal to $M \cdot \mu_i$. A dynamic parallel-access to M servers has a transmission rate $\mu_p = \sum_{i=1}^{M} \mu_i$, which is higher than the transmission rate of a M-parallel-access to the slowest server, but smaller than the transmission rate of a M-parallel-access to the fastest server, $M \cdot \mu_s \leq \mu_p \leq M \cdot \mu_f$.

Next, we consider the situation where there are two mirror servers, a slow one in Greece and a fast one in Spain, and perform the following experiments (i) clients retrieve a document using a single connection to each server, (ii) clients retrieve a document using a dynamic parallel-access to both servers, (iii) clients retrieve a document using a dynamic parallel-access with two connections to the same server.

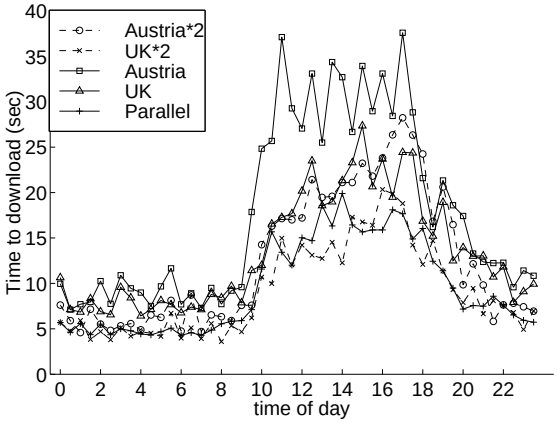
Figure 10 shows the download time obtained for the different schemes and for two different document sizes. For the fast server in Spain, the available resources from the client to the server are abundant, and therefore a two parallel connections to this server result in a reduction of the download time compared to a single connection to the same server. However, when two connections are simultaneously opened to the slow server in Greece, the resulting download time is sometimes higher than the download time obtained if the client would open only one connection to this server. This is due to the fact that the server or network path to Greece is very instable and variable. Thus, opening two TCP connections to the same server does not ensure better response times since the server or the networks may be experiencing high load at that time. With a dynamic parallel-access to both servers, on the other hand, load dynamically shifts to use resources where they are available, thus keeping the load on congested servers/network paths low. As a result, the download time for a dynamic parallel-access to both servers is smaller than a parallel-access to the slowest server and quite comparable to a parallel-access to the fastest server only.

In Figure 11 we have considered the situation where the client opens four parallel connections to a single server and we compare the obtained speed-up with that of a dynamic parallel-access to both servers. We can see that for both Greece and Spain, opening four connections to the same server gives better performance than opening just one. While opening four connections to the fast server in Spain offers smaller download times than a dynamic parallel-access to both servers, in the case of opening four connections to the slow server in Greece, the download time is equal or even higher than a dynamic parallel-access to both servers.

Therefore, while parallel connections to a single server may result in high speedups if the fastest server is selected, they will also result in little speedup if a slow server is selected, even for a high number of concurrent connections. On the other hand, a dynamic parallel-access to both servers automatically achieves very good



(a) $S = 256$ KBytes, $B = 20$.



(b) $S = 763$ KBytes, $B = 30$.

Fig. 10. Retrieval latency for a dynamic parallel-access scheme to $M = 2$ servers compared to a double parallel connection to the same server.

speedups without any server selection. Moreover, when using multiple connections to the same server the links close to the server or the actual server may become congested, and clients will not experience any speedup. With a dynamic parallel-access to different servers, on the other hand, the load is gracefully shared among the servers and there is a higher number of receivers that can experience significant speedups.

VI. Discussion

A. Cost of Parallel-Access

A parallel-access improves the download performance and provides several other performance advantages compared to accessing a single server. However, a parallel-access also has several costs involved that need to be considered.

There is the overhead of doing an extra server access

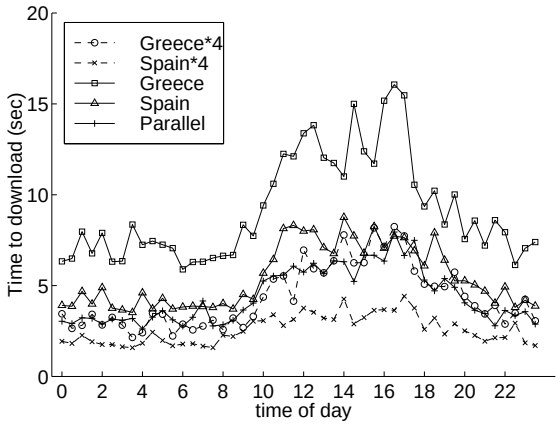


Fig. 11. Retrieval latency for a dynamic parallel-access scheme to $M = 2$ servers compared to a four parallel connections to the same server. $S = 256$ KBytes, $B = 20$.

to find out the document size (this could be done with a HEAD request [16]). Note that for large documents the overhead incurred in obtaining the document size is negligible. Moreover, issuing a first block request with a predetermined fixed size, which would include the document size plus some data, and then using a parallel-access on further block requests, could eliminate this overhead.

There is the overhead incurred by the block request messages. To reduce this cost, an intelligent block assignment policy can be used to gradually increase the size of those blocks assigned to fast servers.

There is an increase in the overall number of TCP connections compared to a single server access. However, the duration of the TCP connections with a parallel-access is smaller since each server delivers only a portion of the whole document. Also, the connection setup is relatively insignificant when big documents are considered. Finally, using servers in different timezones (e.g. selecting servers from places in the world where it is night time), reduces the impact of opening multiple connections since idle servers can be selected.

B. When to use a Parallel-Access: Benefits and Limitations

To outweigh the costs described in the previous section, a parallel-access must be applied to large files in the order of several hundreds of KBytes. Examples of such files are large document downloads, software packages, music, video, and large images. In the Web, most of the files tend to be quite small, and only the root file of a Web page is usually big enough to benefit from a parallel-access. Nevertheless, parallel-access can be used to download Web content if several Web objects are bundled together into a big file. This technique allows, for instance, to rapidly prefetch all the components of a Web page into a Web proxy cache.

When clients access the Internet through slow links, e.g. modem link, the rate of each server may be higher than the rate at the client's access link. In this case performing a parallel-access may result in no additional speedup versus an individual access to one server. However, the servers rates are usually not known a-priori by the client and servers or network conditions may fluctuate during a download session. A parallel-access reduces the uncertainty of selecting a very slow or instable server and brings the client's access link to its full utilization, thus, providing a faster and more consistent experience. One important point to note in this scenario is that disjoint paths are not a necessary condition for speedup. Instead a necessary condition is spare or abundant resources (both at the server and along the path).

When clients access the Internet through fast links, it is very likely that the bottleneck capacity is somewhere in the network or in the servers themselves. As more and more users access the same popular content, the likelihood of sharing the same bottleneck rapidly increases. When the bottleneck resources are fully utilized, all users compete for the same resources and increasing the number of parallel-access connections per user does not result in additional speedups. Though the benefits of a parallel-access in a shared bottleneck environment are limited, a parallel-access still provides a much better experience than selecting a single server since it avoids the risk of selecting a very slow server. Moreover, a parallel-access provides the same average performance to *all* users that share the same bottleneck since it prevents that some users are assigned to fast servers and experience low download times, while other users are assigned to very slow servers and experience high download times. Besides, a parallel-access hides sudden changes in server/link performance by shifting load from servers/links that are overloaded to other parts on the Internet where resources are spare.

One way to reduce the probability that all users share the same bottleneck and thus improve the speedup offered by a parallel-access is to use a dynamic parallel-access in the context of peer-to-peer applications or content distribution networks. In a peer-to-peer environment when a client downloads content from another peer-client, the new client becomes itself a server for future clients. As content becomes more and more popular, it gets replicated in new mirror sites throughout the network. These new mirror sites can be used as new sources to perform multiple parallel-accesses that avoid to compete for the same resources.

In summary, we see the greatest potential for dynamic parallel-access in the area of peer-to-peer applications such as Napster, where the size of content accessed is typically large and the machines storing these copies of the same content are distributed over a large geographical area, which helps improve path-disjointness. An application such as Napster also has the nice property that the more popular a content, the more mirror servers there are. This is due to the fact that as con-

tent becomes more and more popular, i.e. an increasing number of clients have been downloading this content, all these clients potentially become mirror servers.

C. Deployment Issues

These are a number of possible issues that should be considered when deploying a parallel-access system.

Concerning the discovery of mirror servers, the most frequent approach is to publish a list of mirror servers on the master Web server. Clients, manually select the server that they believe will offer the lowest retrieval time. Some search engines provide a full list of mirror servers and rate them in terms of loss rate and round-trip-time [4]. Several organizations who run mirror servers have modified DNS servers or delegate the DNS resolution to other modified DNS servers that return to the client the IP addresses of the administratively closest mirror servers/caches [10] [2]. Other recent studies suggest to extend DNS servers [19] or a central directory [17] to return a full list of all servers containing a copy of a certain document. Current cache-sharing protocols [29] [14] keep local information about the location of duplicated document copies in neighbor caches. When a client requests a certain document and the document is not found in the local cache, the local cache will re-direct the request to the best neighbor cache with a document copy.

To start a parallel-access, the client needs to know the document size. The document size could be given to the clients when they obtain the list of servers (from a central repository, a modified DNS, the origin server, etc). If this is not the case, the client can obtain the document size with a first initial request for a fixed small block.

The number of servers that a client connects to does not necessarily need to comprise all mirror-servers. In fact, most of the performance improvement offered by a parallel-access can be achieved using a few servers [25] [20]. Instead of using all mirror-servers, clients can obtain a reduced subset of mirror servers and perform a parallel-access in this subset. Obtaining a subset of servers requires some server selection process, however, this scoping process is still much easier than selecting the fastest server.

In the case where there are several slow servers and a few mirror-servers with relatively high transmissions rates, almost all blocks will be requested from these fast servers. In this case, a parallel-access may wish to drop the connections from the slow servers to conserve server resources without significantly affecting performance. Moreover, if a server happens to be fast enough to use all the bandwidth in the access link of the end user, a parallel-access may decide to continue just with this fast server or continue using multiple servers for load balancing reasons and to improve reliability and resilience against possible congestion.

Parallel-access to Web documents requires that clients and servers support range requests as specified in HTTP 1.1 and TCP persistent connections. However, current server implementations of range requests and TCP persistent connections have some unpredictable behavior that can affect the performance of a parallel-access. For instance, servers are not required to honor every range request and there may be cases where servers respond with data that does not cover the requested range. Also servers may choose not to honor range requests for certain types of files. Moreover, not all servers support persistent connections and may decide to close the connection after each request transaction. For a more detailed study of the peculiarities of HTTP 1.1 server implementations see [21]. One way to solve these problems is by placing a reverse proxy in front of a server or farm of servers that implements persistent connections as well as range requests.

Parallel-access only works when the replicated content is identically replicated. To make sure that different servers store the exact same version of a document, servers could provide a hashing tag that determines the current version of the document.

Parallel-access can be deployed in client browsers, cache sharing protocols, or content distribution networks. Moreover, parallel-access can be easily integrated in peer-to-peer schemes [7] [3].

VII. Related Work

Currently there exist several software packages that allow clients to dynamically pause, resume, and jump from one mirror server to another during a document download if the current mirror server is very slow [4] [8] [6]. Other software packages allow to open multiple parallel connections to a certain server to speed the download of a certain document [1].

Choosing the best mirror server has been subject of research during the last years. Several techniques have been proposed including multicast communication to poll all the mirror servers [11], dynamical probing [**?**], combining server push with client probes[15], and statistical record-keeping [31]. The work in [31], indicates that the choice of the best server is often not obvious and that the obtained performance can dramatically vary depending on the server selected.

Maxemchuk's work on dispersity routing [24] and Rabin's work on information dispersal [28] explored how to improve document delivery from a *single* server along *multiple paths*. Using erasure codes, the server takes the original document, breaks it into k blocks and generates h parity blocks with the property that *any* k out of the $k + h$ data/parity blocks can be used to reconstruct the original k blocks. By transmitting more than k blocks, the server reduces the download time and increases the probability that the receiver is able to reconstruct the original document even if some of the blocks are lost.

Byers et al. [12] proposed to access *multiple* servers

in parallel. They use an open-loop multicast distribution where different servers generate different sets of parity blocks and cyclically transmit parities and originals. Clients can recover the whole document as soon as they receive enough (k) different blocks, regardless of which servers the blocks came from [12]. To efficiently encode large documents with small encoding/decoding delays, special erasure codes [22], such as Tornado Codes[23], must be used. However, this approach has a major drawback compared to the dynamic parallel-access scheme discussed in this paper. It requires the servers to *encode all their documents* and the clients to *install decoders* to reconstruct the encoded documents. In addition, some problems still remain unresolved such as how to stop the servers or how to perform congestion control.

For these reasons, we propose a parallel access scheme where clients and servers communicate via unicast using TCP. To the best of our knowledge our implementation of dynamic parallel-access is the first parallel-access system that uses standard TCP and HTTP protocols to dynamically request different pieces of a document from the mirror servers, and does not require re-encoding of the content.

More recent experiments based on the parallel-access technique described in this paper were presented in [25]: Several tests were performed for a different implementation of dynamic parallel-access in an environment where the available bandwidth between the client and the servers is much higher (a factor of 3 to 7) than in our experiments. The results obtained indicate that the performance of a parallel-access in such scenarios can be smaller than the performance results presented in this paper. However, the implementation in [25] can benefit from a series of optimizations used in our paper to significantly improve the performance. The main optimization consists in keeping all servers busy at any point in time by avoiding the inter-block and termination idle times. Using these enhancements, a parallel-access continues to offer significant performance improvements compared to a single server selection even in a high bandwidth scenario for documents of several hundreds of KBytes.

Our implementation of dynamic parallel-access can be easily integrated in Web browsers without any modification of the mirror servers and no additional buffer requirements at the clients (since current Web browsers already support opening multiple parallel connections to the same server). It can also be included in cache sharing protocols [14] [29] to share the load among neighbor caches with the same document copy or content distribution networks [2] to speedup the download of popular documents into proxy caches.

The dynamic parallel-access scheme described in this paper has already been integrated into several peer-to-peer applications such as Morfeus [26] or OpenCola [9], which are used to download software, music, images, or video. Both applications provide a mechanism to identify all the peer servers that store the same copy of the content and they perform a parallel-access to speed up content delivery. More recently companies such as Kontiki [5] are using a content distribution network based on end-hosts to provide efficient content delivery. Kontiki uses a dynamic parallel-access scheme to eliminate the selection process while speeding up content delivery and offering a more consistent user experience.

VIII. Conclusions and Future Work

Given that popular documents are often replicated on multiple servers, we suggested that clients connect in parallel to several mirror servers for retrieving a document. We presented a dynamic parallel-access scheme that speeds up document downloads, balances automatically the load among servers, and avoids complex server selection.

We implemented a dynamic parallel-access scheme and evaluated its performance for different numbers of servers, document sizes, and network conditions. We showed that dynamic parallel-access achieves significant speedups and has a performance that comes very close to the optimum performance that can by achieved with any parallel-access scheme. Even when clients are connected through modem lines, a dynamic parallel-access offers a download time that is close to the download time of the fastest server without any server selection. A dynamic parallel-access scheme can be easily implemented and does not require modifications of the content in the mirror servers, in contrast with the digital fountain approach that requires re-encoding of the document [12].

Future versions of our implementation will include pipelining of several blocks to avoid idle times. However, the expected improvement will be modest since a dynamic parallel-access without pipelining already gives download times that are very close to the optimum ones. To reduce the number of negotiations between the client and the servers, clients could keep track of the fastest server during the download of the first blocks and instead of using a fixed block size, dynamically increase the block size for the fast servers. This approach would require some more complexity at the client, but seems a natural extension to our scheme.

The integration of dynamic parallel-access into several existing peer-to-peer applications and content distribution networks is an indication of the importance of dynamic parallel-access for speeding up the object download. Audio/video content is particularly well suited for parallel-access since it is usually static, large, and popular. Easy extensions of the parallel-access technique described in this paper could also be used to greatly improve audio/video streaming performance.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. Eurecom's research is

partially supported by its industrial partners: Ascom, Cegetel, France Telecom, Hitachi, Motorola, Swisscom, Texas Instruments, and Thomson CSF.

REFERENCES

[1] "Agiletp", http://www.daemon-info.com/Us/products.htm.

[2] "FreeFlow: How it Works. Akamai, Cambridge, MA, USA. Nov 1999".

[3] "Gnutella", http://gnutella.wego.com.

[4] "Go!zilla", http://www.gizmo.net/.

[5] "Kontiki", http://www.kontiki.com.

[6] "Leechftp", http://linux.fh-heilbronn.de/ debis/leechftp/.

[7] "Napster", http://www.napster.com.

[8] "Netscape Smart Download", http://www.netscape.com/.

[9] "OpenCola Swarmcast"", http://www.opencola.org/projects/swarmcast.shtml.

[10] "World Wide Web Consortium", http://www.w3c.org/.

[11] J. Bernabeu, M. Ammar, and M. Ahamad, "Optimizing a generalized polling protocol for resource finding over a multiple access channel", *Computer Networks and ISDN Systems*, 27:1429–1445, 1995.

[12] J. Byers, M. Luby, and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads", In *INFOCOM 99*, April 1999.

[13] R. Carter and M. Crovella, "Server Selection Using Dynamic Path Characterization in Wide-Area Networks", 1997.

[14] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", pp. 254–265, Feb 1998, SIGCOMM'98.

[15] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar, "A Novel Server Selection Technique for Improving the Response Time of a Replicated Service", In *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, March 1998.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al., "RFC 2068: Hypertext Transfer Protocol — HTTP/1.1", January 1997.

[17] S. Gadde, M. Rabinovich, and J. Chase, "Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches", In *The Sixth Workshop on Hot Topics in Operating Systems (HotOS-VI)*, May 1997.

[18] V. Jacobson, "Pathchar", http://www.caida.org/Pathchar/ Source: ftp://ftp.ee.lbl.gov/pa thchar.

[19] J. Kangasharju, K. W. Ross, and J. Roberts, "Locating Copies of Objects Using the Domain Name System", In *Proceedings of the 4th International Caching Workshop*, San Diego, March 1999.

[20] A. Kirpal, "Study of Parallel Access Schemes to Speed up the Internet", M.S. Thesis, University of Munich/Institut Eurécom, Sophia Antipolis, France, April 1999.

[21] B. Krishnamurthy, J. Mogul, and D. Kirstol, "Key differences between HTTP/1.0 and HTTP/1.1", Proceedings of WWW-8 Conference, Toronto, May 1999.

[22] M. Luby, "Information Additive Code Generator and Decoder for Communication Systems, US Patent No: 6,307,487 B1", October 2001.

[23] M. Luby et al., "Practical Loss-Resilient Codes", In *STOC*, 1997.

[24] N. F. Maxemchuk, "Dispersity Routing in Store-and-Forward Networks", PhD Thesis, University of Pennsylvania, 1975.

[25] A. Miu and E. Shih, "Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet", Term Paper, LCS MIT, Dec 1999.

[26] Musiccity, "http://www.musiccity.com".

[27] V. N. Padmanabhan and J. Mogul, "Improving HTTP Latency", In *Second World Wide Web Conference '94: Mosaic and the Web*, pp. 995–1005, October 1994.

[28] M. O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance", *Journal of the ACM*, 36(2):335–348, April 1989.

[29] A. Rousskov and D. Wessels, "Cache Digest", In *3rd International WWW Caching Workshop*, June 1998.

[30] M. Sayal, Y. Breibart, P. Scheuermann, and R. Vingralek, "Selection Algorithm for Replicated Web Servers", In *Workshop on Internet Server Performance, SIGMETRICS*, Madison, USA, June 1998.

[31] S. Sesham, M. Stemm, and R. Katz, "SPAND: Shared Passive Network Performance Discovery", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[32] D. Wessels, "Squid Internet Object Cache: http://www.nlanr.net/Squid/", 1996.