Parallel-Access for Mirror Sites in the Internet

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

Abstract—**Popular documents are frequently mirrored on multiple sites in an effort to share the load and reduce clients' retrieval latencies. However, choosing the best mirror site is a non-trivial task and a bad choice may give poor performance. We propose a scheme in which clients access multiple mirror sites in parallel to speedup document downloads while eliminating the problem of server selection. In our scheme, clients connect to mirror sites using unicast TCP connections and dynamically request different pieces of a document from different sites. The amount of data retrieved from a particular site varies depending on the network path/server conditions. Dynamic parallel-access can be easily implemented in the current Internet and does not require any modifications at the mirror sites. Using dynamic dynamic parallel-access, all clients experience dramatic speedups in downloading documents, and the load is shared among servers without the need for a server selection mechanism. Even in a situation where clients are connected through modem lines, dynamic parallel-access offers transmission rates at least as high as the fastest server.**

I. Introduction

The exponential growth of the Internet is overloading popular servers, increasing the demand for bandwidth, and increasing the clients' retrieval times. In order to alleviate these problems, multiple copies of popular documents are often stored in several locations. With network caching, geographically dispersed caches pull and store copies of a document. With mirror site replication, documents are replicated at secondary sites in an effort to both distribute the load of requests across servers and to decrease clients' retrieval latencies.

When a copy of the same document is placed at multiple sites, choosing the best site is not trivial and the obtained performance can dramatically vary depending on the selected site [1] [2] [3]. However, the fact that there are several copies of the same document in geographically dispersed servers allows clients to access several servers in parallel and obtain from every server a different portion of the document. Using a parallel-access eliminates the need for a complex selection process and performs load balancing among the different servers. Additionally, a parallel access can significantly speedup the transfer of a document. Clients experience a transfer rate close to the sum of the individual transfer rates of the servers contacted.

In this paper we develop a parallel-access scheme that uses application-level negotiations to schedule the transmission of different document parts from mirror servers. 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, server two sends the second half, etc. The size of the part delivered by one server is proportional to its 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 achieve the maximum possible speedup, all servers must finish transmitting their part at the same time, i.e., *all* servers must be delivering useful data to the client until the document is fully received. To calculate the size of every part, a history based parallel-access uses a database of previous server rates, which is refreshed periodically, e.g. every 10 minutes. We find that a history-based parallel-access scheme can speedup the transmission of a document when the network/server conditions do not change, since it is easy to predict the rates from the client to every server based on previous measured rates. However, in the case where the network/server conditions change rapidly, especially during day time, a history-based parallel-access scheme has poor performance since it is not able to correctly predict the rate of the different servers during the transmission of the document.

Dynamic parallel-access works as follows. A client partitions a document into small blocks and first requests one different block from each server. When a server finishes the transmission of one block, the client requests from this server another block that has not yet been requested from any other server. When the client receives all blocks it resembles them and reconstructs the whole document. Negotiations between the client and the servers indicating which block to get, are performed at the application-level using the HTTP1.1 byte-range header [4]. Multiple application-level negotiations for the same document and the same server server, use the same TCP persistent connection to avoid multiple slow-start phases [5]. For every negotiation between the client and each server, there is a round-trip-time (RTT), during which no data is transmitted (see Figure 1). To avoid *idle times*, requests for several blocks to the same server can be pipelined. Thus, before one server ends the transmission of one block, the client requests another block from the same server. The scheme implicitly adapts to changing network and server load. When the number of blocks is large, the degree of granularity is high and it is easy for all servers to deliver useful information until the complete reception of the document.

We have implemented a prototype of a dynamic parallel-access scheme as a JAVA client that receives the URL of the mirror servers as input parameters. We evaluated the dynamic parallel-access scheme for a different number of mirror sites, different document sizes and various network/server conditions.

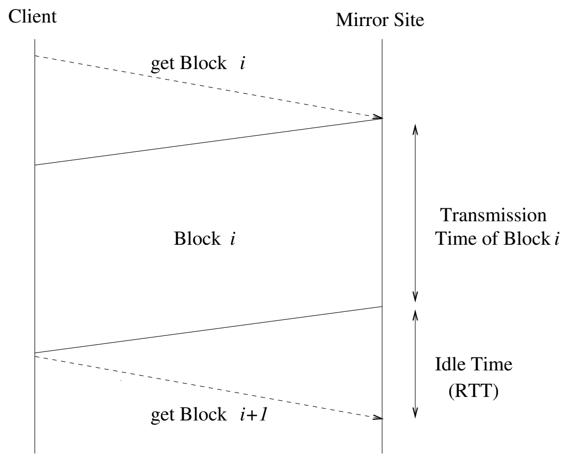


Fig. 1. Dynamic parallel-access : Block request.

Extensive experiments using the JAVA client implementation showed that dynamic parallel-access offers dramatic speedups in downloading a document, even when network/server conditions change rapidly. Very high speedups are obtained when all the servers contacted have similar performance. In the case when one of the servers is much faster than the others, the resulting speedup is not as significant when compared to the fastest server's performance. Even in this latter case, however, the parallel-access scheme offers response times that are at least as low as the ones provided by the fastest server contacted, while avoiding the complicated task of making a server selection.

A parallel-access scheme works efficiently in the case when there is no common bottleneck in the path from the client to the origin server. However, in the case when clients access information through a slow link, e.g. a modem link, connecting to several servers in parallel may not result in an additional speedup. For clients connecting through a modem link, a dynamic parallel-access provides transfer times that are as good as the ones offered by the fastest server. In the case when the mirror servers are very slow, the modem link is not a bottleneck and a dynamic parallel-access reduces the transfer time even more than the transfer time offered by the fastest server.

A dynamic parallel-access to multiple sites provides better speedups than a multiple parallel-connection to a single server, since parallel connections to a single server compete for the same server/network resources. Using a dynamic parallel-access to different mirror servers, parallel connections do not compete among them and the resulting speedup is very high. In addition, with a parallel-access to multiple servers, the number of TCP connections per server is kept low and every TCP connection lasts for a shorter period of time. The load is shared among the servers, and therefore, a higher number of receivers can experience high speedups.

A. Related Work

Choosing the best mirror site has been subject of research during the last years. Several techniques have been proposed including multicast communication to poll all the mirror sites [6], dynamically probing [2], combining server push with client probes[1], and statistical record-keeping [7]. The work in [7], and the work of others, indicates that the choice of the best server is not always obvious and that the obtained performance can dramatically vary depending on the server selected.

One of the most relevant related work in parallel access is Maxemchuk's work on dispersity routing [8] and Rabin's work on information dispersal [9], where a document is divided into several pieces and each piece also includes some redundant information. The receiver obtains different pieces of the document along different network paths and when the receiver has enough pieces the document is reconstructed. Currently there are several software packages that allow clients to dynamically pause, resume, and jump from one mirror site to another during a document transmission if the current mirror site is very slow [10] [11] [12]. Other software packages allow to open multiple parallel connections to a certain site to speed the download of a certain document [13]. The document is divided into several pieces and different pieces are delivered in different connections.

Byers et al. [14] proposed to access multiple servers in parallel using erasure codes [15] [16]. They proposed a parallel access scheme for open-loop multicast/broadcast distributions. Using erasure codes, servers take the original document, consisting on k packets, and generate h parity packets with the property that *any* k out of the $k + h$ data plus parity packets can be used to reconstruct the original k packets. Servers generate different sets of parity packets and cyclically transmit parities and originals. Clients can recover the whole document as soon as they receive k different packets, regardless of the which server the packets came from [14]. To efficiently encode large documents with small encoding/decoding delays, special erasure codes, such as Tornado Codes[16], must be used. Using Tornado codes, an open-loop parallel-access scheme can be scaled to a large number of clients and servers. However, this approach 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, e.g., how to stop the servers, or congestion control.

In the current Internet, most of the communications are performed via unicast between the clients and the servers. Clients and servers use close-loop communications, exchanging feedback messages at the transport-level (TCP) to implement reliability and congestion control. Thus, we propose to implement a parallel access scheme where clients and servers connect via unicast using TCP. A dynamic parallel-access uses application-level negotiations to dynamically request different pieces of a document from the mirror servers. Using the standard TCP and HTTP protocols, a dynamic parallel-access achieves very good speedups, without requiring to re-encode any document on the server. Our dynamic parallel-access implementation can be easily included in Web browsers without any modification of the mirror servers and no additional buffer requirements at the clients (since current Web browsers already support support opening multiple parallel connections to the same server). It can also be included in cache sharing protocols [17] [18] to speedup the download of popular documents and balance the load among neighbor caches with a document copy.

B. Assumptions

We consider *popular* documents that are identically (bit-by-bit) replicated on several mirror servers. We consider *large* documents of several hundreds of KBytes. For small documents, several documents should be grouped into a bigger document, before applying parallel-access to the bigger document (i.e. group all images and text from a Web page into a single document).

We assume that the path from the client to the mirror servers is *bottleneck-disjoint*, that is, packets from one mirror server are not slowed down or dropped due to packets from another mirror server. We consider that servers and clients implement the HTTP 1.1 protocol [4] to allow for persistent connections and application-level negotiations.

The rest of the paper is organized as follows. Section II presents and analyzes a history based parallel-access under different network/server conditions. 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 simulates a dynamic parallel-access with pipelining. Section VI concludes the paper and discusses some future work.

C. Mirror Site Discovery

Concerning the discovery of mirror sites, the most frequent approach is to publish a list of mirror sites on the master Web site. Clients, manually select the server that they believe will offer the lowest retrieval time. Some search engines provide a full list of mirror sites and rate them in terms of loss rate and round-trip-time [10]. Several organizations running mirror sites are modifying DNS servers to return to the client the IP address of the administratively closest mirror site [19]. Other recent studies suggest to extend DNS servers [20] or a central directory [21] to return a full list of all servers containing a copy of a certain document. Current cache-sharing protocols [18] [17] 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.

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 document part should be delivered by each server. The client divides the document into M disjoint blocks, one block for every mirror server. Let μ_i be the transmission rate for server i, $1 \leq i \leq M$. Let S be the document size. Let $\alpha_i S$ be the size of the block delivered by server i and let $T_{t,i} = \frac{\alpha_i S}{\mu_i}$ be the transmission time of this block. To achieve a maximum speedup, all servers must finish transmitting their block at the same time, thus, $T_{t,i} = T_{t,j}$ 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. To achieve a maximum speedup the size $\alpha_i S$ of the block sent by server i, must be equal to $\alpha_i S = \frac{\mu_i}{\sum_{j=1}^{M} \mu_j} S$. Fast servers send a bigger portion of the document, while slow servers send smaller portions. The parallel rate μ_p achieved when all servers keep sending useful data until the document is fully received, is equal to the sum of the individual rates to every server $\mu_p = \sum_{i=1}^{M} \mu_i$.

A history-based parallel-access needs to use a database with information about the previous rates from the different servers to the receiver to estimate the rate to every server, μ_i. 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 actualized periodically with an automated probing from the proxy.

A. Experimental Setup

To evaluate history-based parallel-access we have implemented a parallel-access 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 the document size, the parallel-access JAVA client polls the servers using a HTTP request at the beginning. The document size could also be prerecorded 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/) [22]. Figure 2 shows a network map with the mirror servers considered and the bandwidth of the slowest link in every path as given by pathchar [23]. 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 sites is likely to be bottleneck-disjoint.

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 of a history-based parallel-access where a client at EURECOM requests a 763 KByte doc-

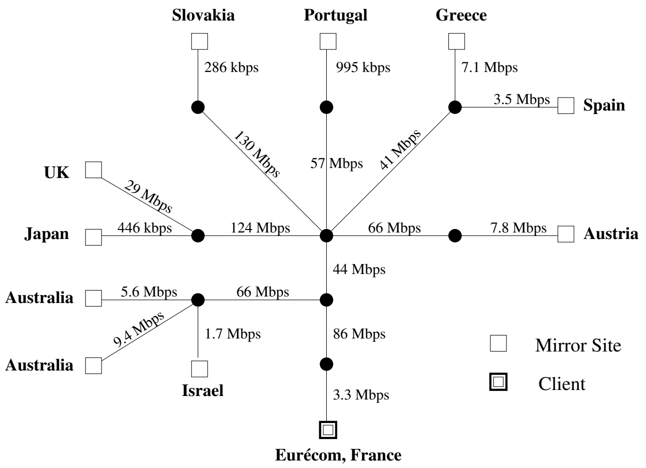
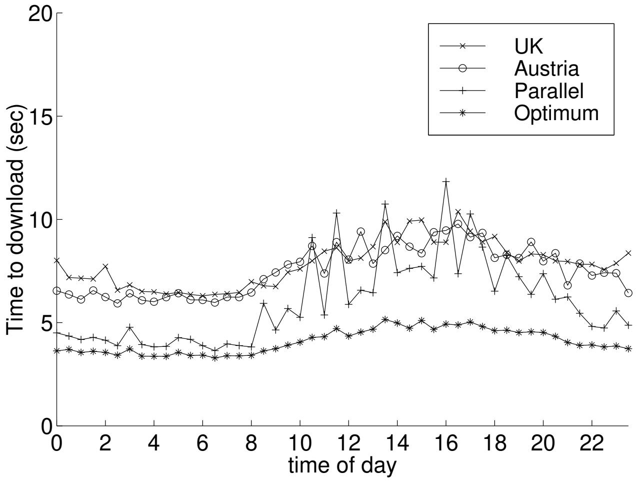


Fig. 2. Mirror sites for the Squid home page. Client is located at EURECOM, France.

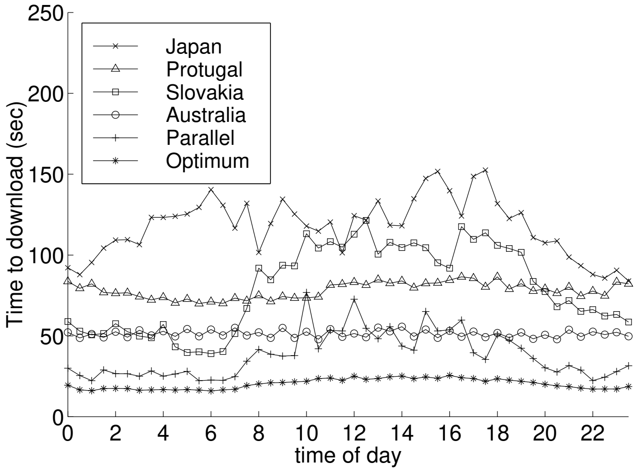
ument from two servers (Austria and UK), which have average transmission rates between $80-100$ Kbps. The actual document is the beta version of the SQUID 1.2 software [22], gzipped. The database with the previous rates from the client to every server is updated when the JAVA client performs a request for the document, that is every 15 minutes. The client assumes that the average rate offered by every server μ_i, will be equal to the rate obtained 15 minutes before.

In Figure 3 we show the transfer time offered by a history-based parallel-access , and the transmission time offered by an individual connection to every server. In addition, we also show the **optimum** transmission time. The optimum transmission time is the transmission time achieved by a parallel-access scheme where all servers send useful information until the document is fully received and there are no idle times between the reception of two consecutive blocks. To calculate the optimum transmission time, the average rates obtained from every server after the reception of a document are used. 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 significantly decreases the transmission 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 using the previous rates achieved, results in bad estimates. Thus, the obtained transmission times with a history-based parallel-access can be higher than the transmission times when clients access every server individually. When the rate to every server is wrongly estimated, some servers stop transmitting before the document is fully received, thus reducing the speedup.

Similar performance of a history-based parallel-access are also obtained for a different set of mirror servers (Figure 3(b)). During day times the database could be refreshed more frequently and other more sophisticated estimating 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 use any database and does not need to estimate the rates to the servers. Instead, this parallel-access uses dynamic negotiations between the clients and the servers to adapt to changing network conditions in real-time.



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



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

Fig. 3. History Based parallel-access .

III. Dynamic Parallel-Access

We consider a parallel-access scheme that uses dynamic negotiations between the clients and the servers as the transmission of the document progresses. With a dynamic parallel-access the document is divided by the client into B blocks of equal size. To request a certain block from a server, the HTTP byte-range header is used. Clients first request one block from every server. Every time the client has completely received one block from a server, the client requests from this server another block that has not yet been requested from another server. When the client receives all blocks it resembles them and reconstructs the whole document. The following points need to be considered when determining the size of the blocks requested:

- The number of blocks B should be larger than the number M of mirror sites that are accessed in parallel.
- Each block should be small enough to provide 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.
- Each block should also be sufficiently large as to keep the idle times between the transmission of consecutive blocks small compared to the transmission time of a block (see Figure 1).

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

Since clients need to perform several negotiations with the same server during the transmission of a document, TCP-persistent connections are used between the client and every server to avoid several slow start phases. When there are less than M blocks missing, idle servers may start transmitting in parallel a block that has already been 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. The maximum number of servers that can be transmitting the same block in parallel is limited to two, to avoid a possible bandwidth waste. The bandwidth wasted in the worst case, is equal to $(M-1)\frac{S}{B}$, where $\frac{S}{B}$ is the block size. However, the bandwidth wasted on average is much smaller since slow servers that did not complete the transmission of a block, are stopped after the document is fully received, and only those block-bytes already transmitted by the slow servers are wasted. Decreasing the block size $\frac{S}{B}$, the percentage of a document that may be received duplicated can be made very small. In addition, to avoid any bandwidth waste, clients could easily determine the fastest server during the transmission of the document. Only this server would transmit the missing bytes of the last block and the other servers would be stopped. Thus, slight modifications of a dynamic parallel-access can avoid most if not all the duplicated packets while hardly influencing the obtained speedup.

A. Analysis of the results

To evaluate the performance of a dynamic parallel-access we implemented the scheme as a JAVA client program. The JAVA program takes as input parameters the URLs of the mirror servers, performs a dynamic parallel-access , saves the document locally, and records the obtained transmission rate. We evaluated the dynamic parallel-access scheme using the experimental setup described in Section II-A. We first consider a dynamic parallel-access to download a 763 KByte document, which is replicated in $M = 4$ mirror sites (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. Our current implementation of dynamic parallel-access does not consider pipelining to reduce the idle times (see Section IV-A for a discussion on pipelining).

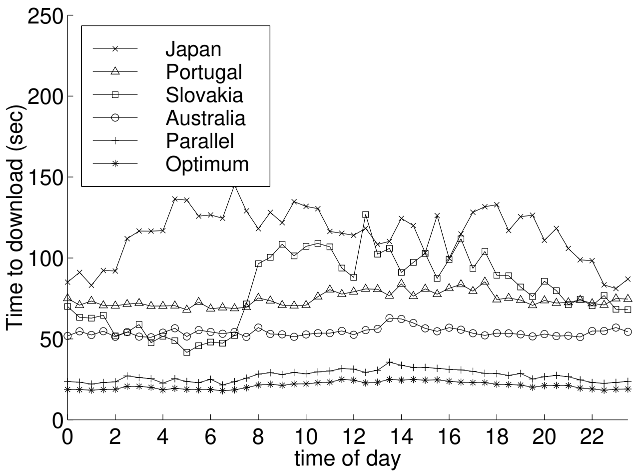


Fig. 4. Dynamic parallel-access . $S = 763$ Kbytes, $B = 30$, $M = 4$.

We 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 can be achieved if all servers keep transmitting useful data until the document is fully received and if there are no idle times. From Figure 4 we can see that dynamic parallel-access offers very high speedups compared to an individual document transfer from any 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 fluctuate, a dynamic parallel-access offers very small transfer times. We observe that the transfer time of a dynamic parallel-access is very close to the optimum transfer time which is an upper bound on the performance of a parallel-access scheme. A dynamic parallel-access is only a couple of seconds slower than the optimum since there are $B = 30$ idle times, which can be avoided using pipelining.

Next, we consider the situation where there are 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 therefore we have also reduced the number of blocks to $B = 20$ to avoid that idle times account for a high percentage of the total transfer time (the document is the FAQ from SQUID in postscript format [22]).

We can see that a dynamic parallel-access achieves a transfer time, that is almost half the transfer time of the fast servers (slow servers only contribute sending few blocks and decreasing 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

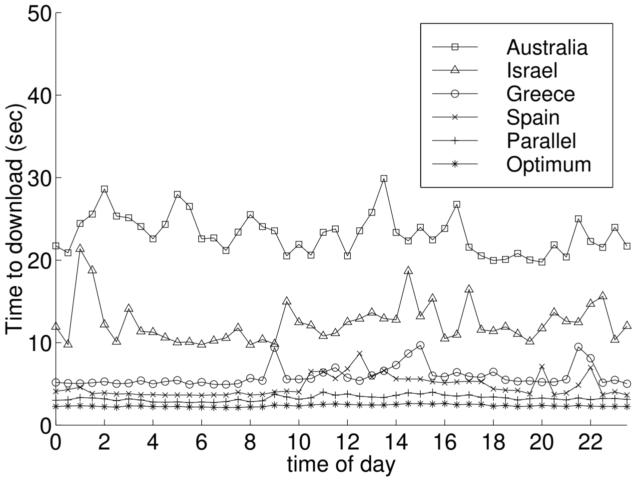


Fig. 5. Dynamic parallel-access . $S = 256$ Kbytes, $B = 20$, $M = 4$.

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 a 5 MBytes document from a single server can be up to 80 seconds. Using a dynamic parallel-access the transfer rate is less than 30 seconds. Due to the high number of blocks used, $B = 60$, the transfer time using the dynamic parallel-access scheme takes a few seconds longer than for the optimum parallel-access . This difference can be avoided by pipelining requests for several blocks (see Section IV-A).

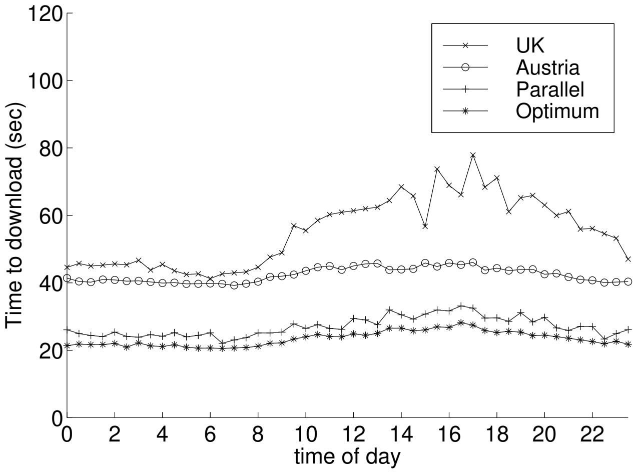


Fig. 6. Dynamic parallel-access . $S = 5$ Mbytes, $B = 40$, $M = 2$.

B. *Parallel Access for 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 Figure 7 we see the performance of a dynamic parallel-access scheme for a 10 KB document. We considered two mirror servers (Spain and Greece) and $B = 4$ blocks. We see that a dynamic parallel-access has a transmission time close to the transmission time of the fastest server, even though sometimes is slightly higher. Compared to the optimum transmission time, a dynamic parallel-access has a much higher transmission time since the idle times account for a high percentage of the total transmission time. To avoid idle times pipelining can be used.

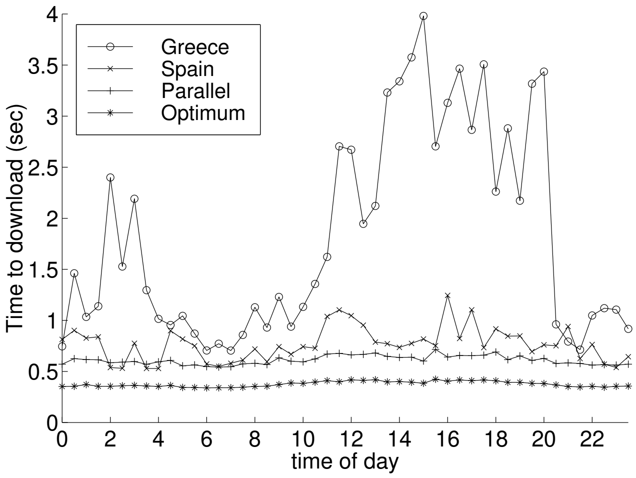
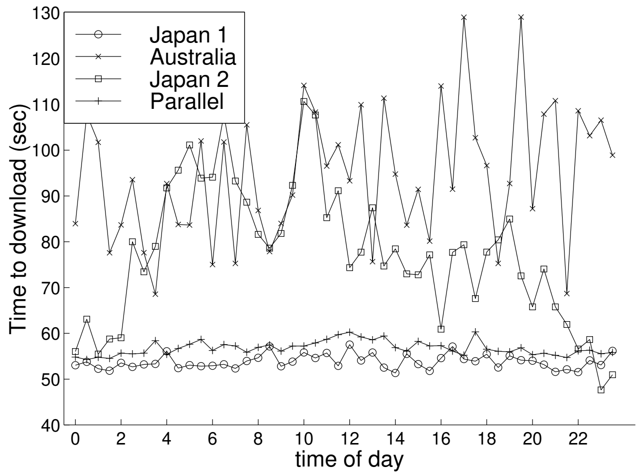


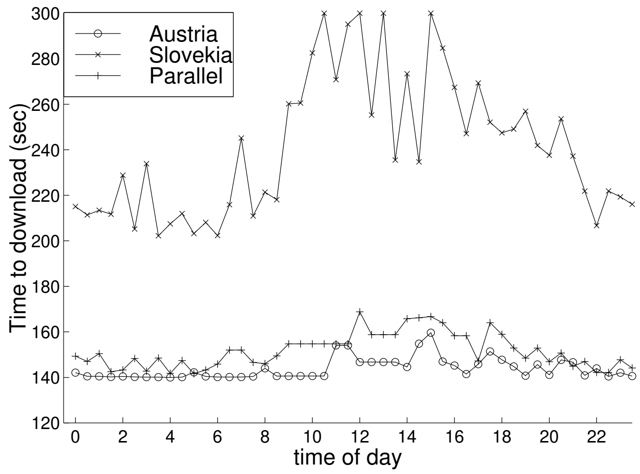
Fig. 7. Dynamic parallel-access . $S = 10KB$, $B = 4$, $M = 2$.

However, pipelining 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 $\frac{S}{B} > 1KB$. Thus, if the document size is $S = 10$ KB, and we choose a block size of 2 KBytes, the number of blocks is equal to 5. When the number of blocks is small, the degree of granularity is decreased and it is difficult that all servers keep sending useful information until the full reception of the document without wasting a lot of bandwidth. Thus, for small documents the need for pipelining results in a small number of blocks B, which does not allow to efficiently implement dynamic parallel-access .

In addition, with small documents the connection setup time may account for a significant portion of the total transmission time. A parallel-access scheme speeds up the transmission time of the document but can not do anything about the connection time. If the time to connect to the server is very high, the client may not experience any noticeable difference. To obtain better performances with a parallel-access , several small documents could be grouped together, i.e. all documents in a Web page, and perform a dynamic parallel-access to the bigger document.



(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 using a modem link.

IV. Dynamic parallel-access in a Bandwidth-Limited Environment

In this section we study the performance of a dynamic parallel-access where a client is connected through a low speed access link, i.e. a modem link. In this case the path from the client to the servers is not bottleneck-disjoint. A single server may 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.

In Figure 8 we consider dynamic parallel-access for a client connected through a modem line at 56 Kbps. We show the transmission 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 considered two slow servers (Japan 1 and Australia) and a fast server (Japan 2). For an individual access to the fast server, the modem bandwidth is fully utilized and the transmission time varies little during all the periods of the day. For an individual access to the other two slow servers, the rates obtained are about 24 Kbps, which is much lower than the maximum modem link rate (56 Kbps). In this situation the modem link is not fully utilized and the transmission time fluctuates depending 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 only two mirror-servers, a fast one and a slow one.

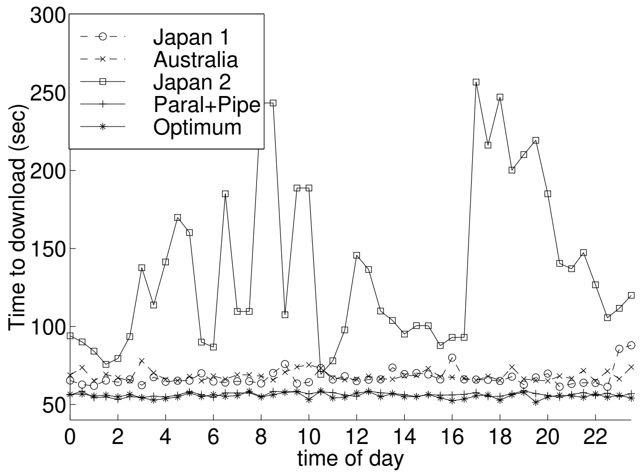
For the dynamic parallel-access , we see that the transmission rate achieved is close to the transmission rate of the fastest server, which is equal to the transmission rate of the modem link. The fact that with a dynamic parallel-access the transmission rate obtained is always slightly higher than the transmission rate offered by the fastest server is due to the idle times between block requests. Next, we study the performance of a dynamic parallel-access that uses pipelining to avoid idle times.

A. *Simulation of Pipelining*

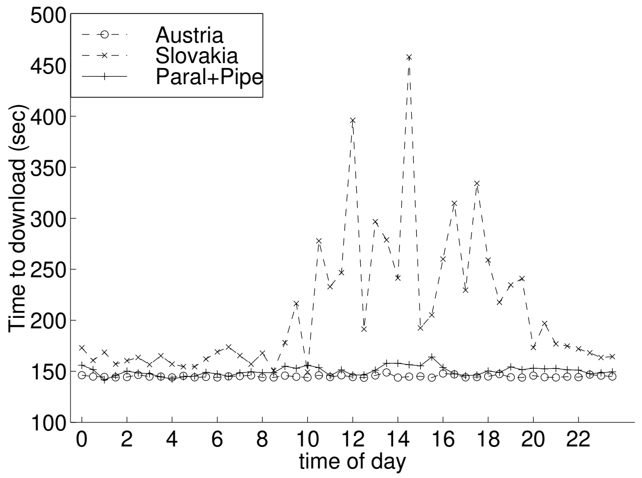
In this section we repeat the previous experiments (Figure 8(a) and 8(b)) simulating a dynamic parallel-access with pipelining. To estimate the transmission time with pipelining, we measured the RTT to every server and then recalculate the transmission time assuming that RTT=0. This simulation gives an upper bound on the performance of pipelining since it assumes that all RTTs can be fully suppressed.

In Figure 9 we see that the rate achieved by a parallel-access with pipelining through a modem link. We observe that the rate offered by a parallel-access with pipelining is smaller (Figure 9(a)) or equal (Figure 9(b)) than the rate achieved by a single connection to the fastest server. In Figure 9(a) the transmission rate of each server is much smaller than the maximum modem link rate, thus, a single server connection does not fully utilized the modem link. In this case, a parallel-access with pipelining can speedup the transfer of a document compare to a single server connection, and achieve transmission 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 with pipelining is almost equal to the optimum transmission time. Thus, the additional delay that the JAVA implementation may introduce is very small. The results achieved with pipelining in Figure 9(a) can also be applied to the other dynamic parallel-access experiments presented Section III.

Using pipelining is not so crucial since the performance of a parallel-access without pipelining is already very good, and the expected benefits are small. Implementing pipelining can be easily done and does not require to calculate the exact RTTs but simply estimate an upper bound on the RTTs. Using an overestimated RTT, pipelining could eliminate the idle times with no



(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. Pipelining simulation.

performance degradation.

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

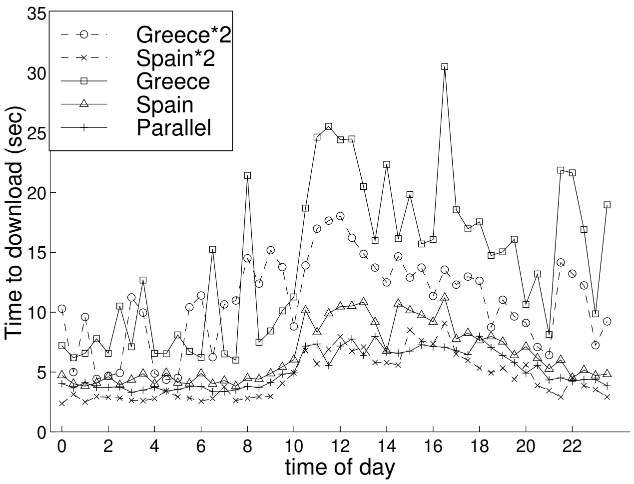
In this section we compare a dynamic parallel-access to multiple mirror-servers with a parallel-access to a single server. For a fair comparison, we consider the situation where a single client opens M TCP-parallel connections to the same server and compare this case to a dynamic parallel-access to M 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 in 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 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 with an individual connection to both servers, (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.

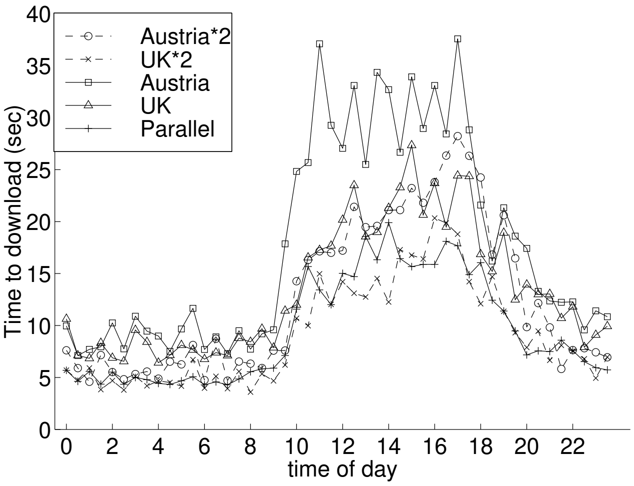
Figure 10 shows the transmission time obtained for the different schemes and several document sizes. For the fast server in Spain, the available resources from the client to the server are abundant, and therefore a two parallel connections to this server result in a reduction of the transmission time compared to a single connection to the same server. However, when two connections are simultaneously opened to the slow server in Greece, the resulting transmission time is frequently higher than the transmission time obtained if the client would open only one connection to this server. This is due to the fact that both TCP connections compete for the same resources (network bandwidth and server capacity), and packets from one connection may cause loss of drop packets from the other connection. For a parallel-access to both servers, connections use disjoint bottlenecks and do not compete for the same resources. Thus, the transmission time of a dynamic parallel-access connection to both servers is much smaller than the transmission time of a double connection to the slowest server and is very close to the transmission time of a double connection to the fastest server.

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. This is not a fair comparison for the dynamic parallel-access to both servers since a client only receives data in parallel from two connections and not four. In the case that the client opens four parallel connections to the fast server in Spain, the transmission time is smaller than a dynamic parallel-access to both servers. However, in the case that the client in France opens four connections with the server in Greece, the transmission time is equal or even higher than a dynamic parallel-access to both servers.

Therefore, even though parallel connections to the same server may result in small transmission times if the fastest server is selected, it may also result in no speedup if a slow server is selected, even for a high number of parallel connections to that server. On the other hand, a dynamic parallel-access to both servers automatically achieves very good speedups without any server selection. If many clients open parallel 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, the load is shared among the servers and there is a higher number of re-



(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.

ceivers that can experience significant speedups.

VI. Conclusions and Future Work

Given that popular documents are often replicated in multiple servers in the network, we suggest that clients connect in parallel to several mirror sites for retrieving a document. We presented a dynamic parallel-access that uses application-level negotiations to speedup document downloads, balance the load among servers, and avoid complex server selections. We evaluated the performance of dynamic parallel-access for different number of servers, document sizes, and network conditions. Dynamic parallel-access achieves significant speedups and has a

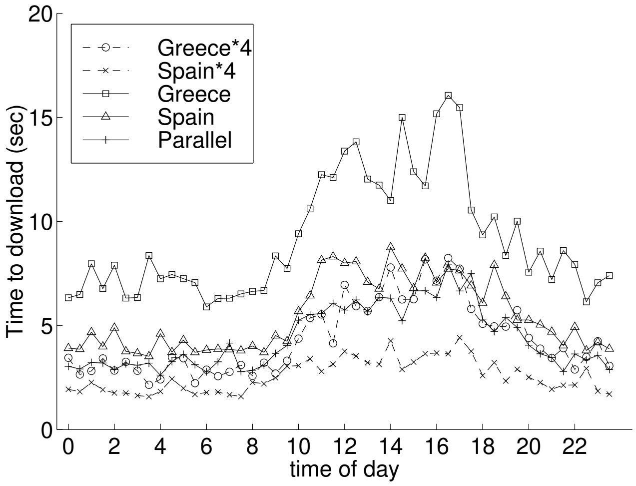


Fig. 11. Retrieval latency for a dynamic parallel-access scheme to $M = 2$ servers compared to a four parallel connections to the same server. $S = 256$ KBytes, $B = 20$.

performance that comes very close to the optimum performance of a parallel-access. Even when clients are connected through modem lines, a dynamic parallel-access offers a transmission time that is close to the transmission time of the fastest server without any server selection. A dynamic parallel-access can be easily implemented in the current Internet and does not require modifications of the content in the mirror sites, in contrast with the digital fountain approach [14].

Future versions of our parallel-access will include pipelining of several blocks to avoid idle times. However, the expected improvement will not very high since a dynamic parallel-access without pipelining already gives transmission 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 transmission 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 to track the fastest servers, but seems a natural extension to our scheme.

References

[1] Zongming Fei, S. Bhattacharjee, Ellen W. Zegura, and Mostafa 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.

[2] Mark Crovella and Robert Carter, “Dynamic server selection using bandwidth probing in wide-area networks,” in *Proceedings of IEEE INFOCOM*, 1997.

[3] Mehmet Sayal, Yuri Breibart, Peter Scheuermann, and Radek Vingralek, “Selection algorithm for replicated web servers,” in *Workshop on Internet Server Performance, SIGMETRICS*, Madison, USA, June 1998.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al., “RFC 2068: Hypertext transfer protocol — HTTP/1.1,” Jan. 1997.

[5] V. N. Padmanabhan and J. Mogul, “Improving http latency,” in *Second World Wide Web Conference ’94: Mosaic and the Web*, Oct. 1994, pp. 995–1005.

[6] 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*, 1995.

[7] 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.
[8] N. F. Maxemchuk, "Dispersity routing in store-and-forward networks," PhD Thesis, University of Pennsylvania, 1975.
[9] Michael O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, April 1989.
[10] "Go!zilla," http://www.gizmo.net/.
[11] "Netscape smart download," http://www.netscape.com/.
[12] "Leechftp," http://linux.fh-heilbronn.de/ debis/leechftp/.
[13] "Agiletp," http://www.daemon-info.com/Us/products.htm.
[14] John Byers, Michael Luby, and Michael Mitzenmacher, "Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads," in *INFOCOM 99*, Apr. 1999.
[15] Luigi Rizzo, "Effective erasure codes for reliable computer communication protocols," *Computer Communication Review*, vol. 27, no. 2, pp. 24–36, April 1997.
[16] M. Luby et al., "Practical loss-resilient codes," in *STOC*, 1997.
[17] Li Fan, Pei Cao, Jussara Almeida, and Andrei Broder, "Summary cache: A scalable wide-area web cache sharing protocol," Feb 1998, SIGCOMM'98.
[18] Alex Rousskov and Duane Wessels, "Cache digest," in *3rd International WWW Caching Workshop*, June 1998.
[19] "World wide web consortium," http://www.w3c.org/.
[20] J. Kangasharju, K. W. Ross, and J.W. Roberts, "Locating copies of objects using the domain name system," in *Proceedings of the 4th International Caching Workshop*, San Diego, March 1999.
[21] Syam Gadde, Michael Rabinovich, and Jeff 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.
[22] D. Wessels, "Squid internet object cache: http://www.nlanr.net/squid/," 1996.
[23] Van Jacobson, "Pathchar," http://www.caida.org/Pathchar/ Source: ftp://ftp.ee.lbl.gov/pa thchar.