Scalable Parallel-Access for Mirrored Servers

Amgad Zeitoun Hani Jamjoom Mohamed El-Gendy
Department of Electrical Engineering and Computer Science,
The University of Michigan
1301 Beal Ave.
Ann Arbor, MI 48109-2122, USA
Tel: 1-734-936-0393
Fax: 1-734-763-8094
{*azeitoun, jamjoom, mgendy*}*@eecs.umich.edu*

Paper ID: 351-436

Keywords: Computer Networks, Internet Tools, Applications, Parallel Downloading.

Principle Author: Amgad Zeitoun

Email: azeitoun@eecs.umich.edu

Address:

3327 EECS Building
The University of Michigan
1301 Beal Ave.
Ann Arbor, MI 48109-2122
USA

Tel: 1-734-936-0393

Fax: 1-734-763-8094

We confirm that if the paper is accepted in the Applied Informatics 2002 (AI 2002) conference, one of the authors will attend the conference to present the paper.

Scalable Parallel-Access for Mirrored Servers

Amgad Zeitoun Hani Jamjoom Mohamed El-Gendy
Department of Electrical Engineering and Computer Science,
The University of Michigan
Ann Arbor, MI 48109-2122, USA
{*azeitoun, jamjoom, mgendy*}*@eecs.umich.edu*

Abstract— **To increase a client's perceived throughput while transferring large multimedia files, a parallel-access approach can be used to simultaneously transfer different pieces of a file from different mirrors. Previously proposed approaches, however, either employ very aggressive implementations with poor scalability or suffer from deployment difficulties on the current Internet. Using trace-driven and real Internet experiments, we show the effect of poor scalability and performance degradation for some of the previous parallel-access implementations. In this paper, we propose a hybrid scheme that (1) minimizes the added load on all mirrors while (2) maximizing the client-perceived throughput. Our scheme uses informed mirror selection based on round-trip time (RTT) measurements as well as dynamic monitoring. While using less than 10% of the available mirrors, we show a decrease in transfer time by more than 85% over traditional single-server transfers and 35% over other parallel-access implementations.**

Keywords— **Internet Tools, Applications, Parallel Downloading, File Transfer, Mirror Selection.**

I. Introduction

Improving client-perceived throughput is an ongoing effort in today's Internet. Currently, files are replicated across different mirrors to increase availability and decrease service time. Simple mirroring services, like Tucows [1], have users select from a list of available mirrors. In contrast, more complex services, such as Akamai [2], automatically direct clients to the closest mirror, represented by a caching proxy. While either approach is proven sufficient for well-provisioned servers, bulk data transfers, such as multimedia files, have rendered this model insufficient for typical servers as it exacerbates the effects of heterogeneous link speeds and server reliability.

To minimize these effects, a parallel-access technique, named *paraloading* [3], is presented in [3, 4] where an application opens separate connections to all or a randomly chosen subset of the mirrored servers. It then transfers different blocks, representing different portions of the file, from each mirror[1]. When the application finishes transferring a data block, it immediately issues another request to the server to get another block. Once all data blocks are received, they are reassembled into the target file.

Parallel-access schemes are shown to have increased resilience to route and link failures and traffic fluctuations while maximizing the effective transfer throughput of the requested file. Furthermore, the authors of [3, 4] have argued that parallel-access can be used to eliminate the mirror selection problem, which is deciding the most appropriate mirror. While their results are valid from a single client's perspective, we show that large-scale deployment will render their scheme useless.

The heart of the problem is scalability. A parallel-access scheme must balance two conflicting criteria. On one hand, it should scale the number of connections for a single client to maximize the perceived throughput (we call this the *performance criterion*). On the other hand, it must also minimize the interference between the aggressiveness of the parallelism and the performance of all clients and mirrors (we call this the *scalability criterion*).

In this paper, we present a scalable parallel-access mechanism that maximizes client throughput by dynamically selecting a small subset of mirrors based on the round-trip times (RTTs) between the client and the servers. Our scheme also uses dynamic monitoring to converge the selection to the set of optimal servers. We show that this scheme has two primary advantages. First, ranking mirrors based on RTT values prioritizes mirrors with larger available bandwidths; as we will show, choosing the closest mirrors satisfies the performance criterion. Second, in most cases, only a small subset of mirrors, which varies across different clients, is needed to reach this performance thus satisfying the scalability criterion.

This paper is organized as follows. Section II briefly reviews related work. We motivate our server selection algorithm in Section III and discuss the merits of informed mirror selection on scalability and performance in Section IV. We present our implementation for a parallel-access client in Section V. In Section VI, we empirically evaluate some performance issues. Finally, the paper ends with concluding remarks in Section VII.

II. Related Work

The scalability of parallel-access was originally studied in the context of improving a client's throughput by initiating multiple connections to the same server [5], typical of most web browsers [6]. It was shown that the added aggressiveness may lead to unfair allocation of network resources. While naive implementations of parallel-access do incur similar problems, our scheme is different for two reasons. First, by minimizing the number of selected mirrors and converging to an even smaller number of fast servers our scheme minimizes aggressiveness. Second, because a file is transferred using small chunks from different servers, it is easy to see that the transfer time is reduced resulting in an expected number of simultaneous connections that is close to its single connection counter-

[1] "mirror" and "server" are used interchangeably in this paper

part.

In [7], the authors use multicast to ensure the scalability of accessing multiple mirrors in parallel. In their scheme, each server uses Tornado Codes to split and encode each document into N disjoint blocks such that a client can reconstruct the document by only retrieving $k < N$ blocks. By using a separate multicast channel for each mirror, any client can subscribe to multiple channels and retrieve the document in parallel. This scheme has two drawbacks. First, it requires large modifications to existing non-Tornado aware servers and clients. Second, even if such technology is universally adopted, it has limited scalability as it requires multiple multicast channels for different documents. Similar to [3, 4], our approach integrates well with the existing Internet infrastructure as it requires minimal changes to existing applications.

Our approach relies heavily on appropriate mirror selection and mirror ranking, which has been extensively studied in the literature [8–15]. Previous work on mirror selection can be classified into three categories: network-layer [10], application-layer [8, 15], or measurement-based server selection [8, 9, 12, 13]. Each selection algorithm has focused on one or more performance metrics (e.g., RTT, routers hop count, response time, and available bandwidth). The effectiveness of client-side mirror selection and provider-side mirror selection was studied in [9] and [14, 16], respectively. Both studies have concluded that dynamic selection (e.g., RTT, server load, and available bandwidth) performs better than static selection (e.g., geographical location and hop count).

III. Rationale

Ensuring the scalability of a parallel-access scheme is a design challenge that must be given careful consideration. Naive implementations will, at best, render their approaches useless. In the worst case, the added aggressiveness may overload or even crash the target servers. As mentioned in Section I, two criteria must be considered when designing a parallel-access scheme, namely the performance and scalability criteria. More specifically, the performance criterion should minimize the average transfer time for all clients. On the other hand, the scalability criterion should minimize the degree of parallelism while uniformly distributing the load over all mirrors.

Meeting these requirements directly translates to an optimal selection algorithm where a set of N clients must choose from M mirrors with limited capacity (mainly bandwidth) subject to the "minimize average transfer time" constraint. Unfortunately, this can be mapped to the knapsack problem, an NP-complete problem [17], which is almost impossible to solve in real-world deployment scenarios. We, thus, try to obtain a near-optimal solution by proposing a dynamic selection algorithm. In our approach, each client acts independently and chooses the target mirrors based on local measurements.

Designing a dynamic selection algorithm is not a straightforward task especially without server-side collaboration, e.g., reporting system load and available bandwidth. Alternatively, we focus on client-side measurements, namely using measured RTT values to all mirrors to improve mirror selection. Parallel-access scheme presented in [4] does not elaborate any selection scheme, instead it uses all available mirrors in the parallel download. We do not analyze their scheme in this paper since it clearly has poor scalability and high overhead. On the other hand, Paraloading [3] selects a random subset from all available mirrors based on the *Degree of Parallelism* (DoP). The DoP is the number of opened parallel connections to different servers. Hence, we compare two selection algorithms in this paper:

- **Random selection:** chooses a number of servers randomly from a list of all available mirrors. The number of servers chosen depends on the required DoP. When the DoP is one, server selection is analogous to the *conventional* single server selection exists in typical mirroring service (e.g., Tucows), where a user selects a mirror at random.
- **RTT-based selection:** dynamically chooses a subset of all available mirrors based on measured RTT values to all mirrors. For a DoP equal to k connections, RTT-based selection chooses the k servers with the smallest RTT values.

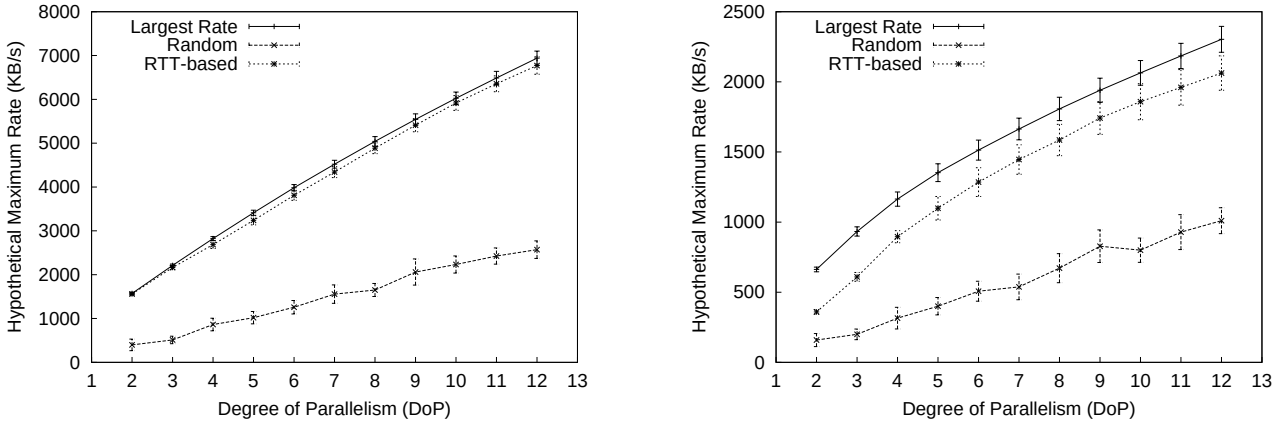
Unfortunately, it is intractable to perform "real" measurements to evaluate the scalability and performance of these selection algorithms at both client-side and server-side. We also believe that simulation alone will not capture the true behavior of clients and servers. We therefore use a combination of trace-driven analysis and client-side measurements to determine the load-balancing and improvements in the transfer-time of the selection algorithms. These are presented in the following two experiments. At the end of this section, we show that the combination of the results makes RTT-based selection the ideal candidate for large-scale deployment scenarios.

TABLE I
Mirrors used in the client perceived throughput experiment

Mirrors	Number	File Size	Coverage
Akamai	262	2.76MB	Worldwide
FreeBSD	52	2.8MB	Worldwide
Tucows	148	2.0MB	USA

A. Experiment 1: Client Perceived Throughput

In this experiment, we compare the effectiveness of both Random and RTT-based selections in satisfying the performance criterion. We based our analysis on measurements from three different mirroring services on the Internet: Akamai [2], FreeBSD [18], and Tucows [1]. From each mirror service, a reasonable size subset of all available mirrors is chosen (see Table I). We used several machines to act as clients requesting files from each server after measuring the corresponding RTT. Each request was repeated 20 times over a period of one week and the client perceived transfer rate for each server, $\frac{file\ size}{transfer\ time}$, was averaged over the 20 measurements. Due to space limitation, we only present results collected from a machine located at the University of Michigan in the US.



(a) Akamai (b) FreeBSD

Fig. 1. The average hypothetical maximum rate perceived when using Largest Rate selection, Random selection, or RTT-based selection.

To estimate the expected performance improvement at various DoP, a *hypothetical maximum rate (HMR)* is calculated by summing up the transfer rates of the corresponding servers at a given DoP. The HMR is an overestimate for the actual perceived aggregate rate by the client, because it assumes that links are bottleneck free. Yet, it is still a good indication of how well the set of selected servers can perform. Fig. 1 shows the average HMR, with 95% confidence interval, perceived by our client when using Random selection or RTT-based selection for Akamai and FreeBSD mirrors (Tucows exhibits similar performance and is not shown due to space limit). The figure also compares both selection algorithms against Largest-Rate selection, the optimum by definition, representing the upper bound of throughput that can be perceived by the client at a given DoP. Largest-Rate selection simply chooses the fastest servers and, in our case, is computed *a posteriori* to data collection.

We can clearly see that Random selection performs the worst (on average 50% lower). This is simply because for any client, there are only a small subset of available mirrors that represent the fastest servers. This decreases the probability of selecting the fastest servers and is represented by the smaller slope of Random selection compared with the other two algorithms in Fig. 1. For example, in the FreeBSD mirrors, the HMR achieved by the Random selection with DoP of 11 can be achieved with DoP of 3 for the Largest-Rate selection. This compares with DoP of 4 in the case of RTT-based selection. The informed nature of the RTT-based selection has clearly produced favorable results, especially when the set of mirrors is well-provisioned (such as Akamai). In the next experiment, we show that this improved performance of RTT-based selection is crucial for server-side scalability.

B. Experiment 2: Server-side Scalability

A selection algorithm has a direct impact on the server load; for bulk data transfer it can be reflected by *the clients' workload* on each server. The clients' workload represents the expected number of connections perceived by a server. Thus, we define a scalable selection algorithm as one that will increase the clients' workload on any server by only a constant factor.

The combination of improved performance and good load balancing determines the true scalability of the underlying algorithm. The basic rule is that if, for instance, the number of parallel connections was doubled from k to $2k$ connections and was uniformly distributed across all servers, then in order to keep the expected workload constant on all servers, the transfer time should be reduced by 50%. On the other hand, if the new k connections are directed to slow servers, the performance of the selection algorithm only improves marginally, but, the workload will increase proportionally with the number of simultaneous connections. This is exactly the problem with the Random algorithm.

Clearly, a Random selection algorithm uniformly distributes the clients' workload across all servers. However, experiment 1 showed that in order for Random selection to match the optimal performance of Largest-Rate selection, a larger number of servers must be chosen (i.e., larger DoP). Therefore, increasing the DoP for the Random selection improves the throughput marginally while increasing the workload.

Focusing on RTT-based selection, we use trace-driven analysis to give a general insight into its expected load balancing behavior. We use, as servers, 15 Traceroute Gateways (TGs) located in the US. TGs [19] are distributed servers that perform `traceroute` — on-demand — to different hosts on the Internet. We collect 11,814 unique IPs, distributed worldwide, from the Web log of a popular Web server. Viewing these IPs as clients, we are able to determine the RTT values from the 15 TG servers to these 11,814 clients. Since, the RTT reflects the total delay of the forward and backward paths between hosts, it is independent of the side that originated the measurement. Therefore, we can accurately view our measurements as RTT values from 11,814 clients to the 15 TG servers.

From these measurements, we analyze the expected clients' workload on each server. For each client, we extract the closest k servers based on RTT selection ($DoP = k$) and calculate the clients' workload on each server by summing up the number

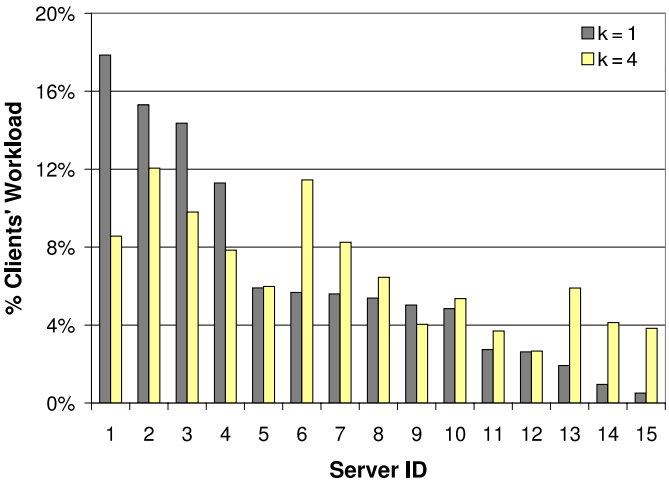


Fig. 2. The percentage of clients' workload perceived by different servers when using RTT-based selection to select the best k servers.

of clients directed to it. Fig. 2 shows the percentage of clients' workload on each server for $k = 1$ and $k = 4$. For $k = 1$, there are 3 regions in the figure, the region of most popular servers (first 4 servers), the region of uniformly distributed workloads for different clients (servers 5 through 10), and the region of bad servers (servers 11 through 15). When the DoP increases to 4 ($k = 4$), RTT-based selection does not proportionally increase the load on the most popular servers, instead the clients' workload is distributed to other less popular servers.

We conclude that since clients are perceiving varying RTT measurements depending on their locations with respect to the set of mirrors, RTT-based selection does not increase the number of connections on maximally-loaded servers (i.e., popular servers). Instead, RTT-based selection will distribute clients' workload on different servers. Briefly, RTT-based selection balances our two design criteria for the following reasons:

1. *Performance criterion.* TCP, which is used by FTP and HTTP, is biased against connections with large RTT [20], thus when two connections have the same bandwidth but different RTTs, the transfer rate of the larger RTT connection will be worse than that of the smaller RTT connection due to slow returning acknowledgments. RTT-based selection thus selects those servers with potentially faster transfer rates.
2. *Scalability criterion.* The selected sets of servers vary across clients since clients are scattered across the Internet and measure different RTT values to all mirrors. This behavior balances clients' workload across all servers. Furthermore, because of the small DoP that is required to maximize client performance, RTT-based selection is expected to minimally increase the clients' workload on all servers.

The previous two experiments have shown that RTT-based selection is the ideal candidate that combines both good performance and load-balancing. Nevertheless, there are two points that must be considered. First, while Random selection has consistently performed worse than its counterpart, it may still be sufficient for clients with slow connections (e.g., dial-up modems [4]). Second, Random selection has the least overhead, since it does not require any client-side or server-side measurements.

IV. Informed Mirror Selection

Ideally, clients should be able to accurately select the exact number of servers that are necessary to optimize their performance goals (mainly minimize transfer time). We loosely define a client performance criterion based on the number of fastest servers (ones with highest throughput) that he/she must select. A mirror selection algorithm is then viewed as a function that *maps* the fastest servers, for a given client, from a large set into a smaller subset. The size of the selected subset, is determined by the underlying selection algorithm and the mirrored servers. For example, in Fig. 1(b), RTT-based selection requires a larger subset of selected servers to realize an aggregated rate equivalent to the fastest k servers.

More formally, let I_k be the smallest set of selected servers that includes the fastest k servers. Since, the RTT ranking of k servers does not necessarily provide a one-to-one mapping to the corresponding fastest k servers, the size of this set ($|I_k| \geq k$) is modeled as a random variable that depends on the selection algorithm and the mirrored servers. Therefore, a probability value is associated with $|I_k|$ to indicate the likelihood that the fastest k servers are included in the set I_k. For a given mirror service, we can construct a graph that relates k and $|I_k|$ with a given probability p. Fig. 3 shows $|I_k|$ that maps the best k servers with probability $p \geq 0.9$, where $1 \leq k \leq 12$. From the figure, we clearly see that the relation between k and $|I_k|$ is dependent on the mirroring services. For example, to get the best 2 servers, we need to select the first 2, 4, and 9 servers from the RTT-ranked list for Akamai, FreeBSD, and Tucows, respectively.

Constructing a graph like the one in Fig. 3 requires both measuring the RTT as well as transferring the actual file from each server. This is obviously an impractical approach and, unfortunately, has no alternative. These measurements, however, can be performed by a separate service on the Internet. Unfortunately, measuring RTT to all mirrors, while substantially less than transferring the file, introduces overhead. This overhead can further be minimized by using other services that provide end-to-end delay estimation on the Internet, such as IDMaps infrastructure [21].

V. Implementation

We implemented an FTP client that supports parallel-access (PA-FTP). Fig. 4 shows our general design as well as the six basic steps that are taken when transferring a file.

1. A user submits a request to the PA-FTP client.
2. The manager thread measures the RTTs to all available servers. It then selects a smaller subset defined by the maximum DoP. In our example, it selects servers *b* and *c*.
3. The manger partitions the file into two blocks proportional to the corresponding RTT values. It assigns each block to a worker thread.

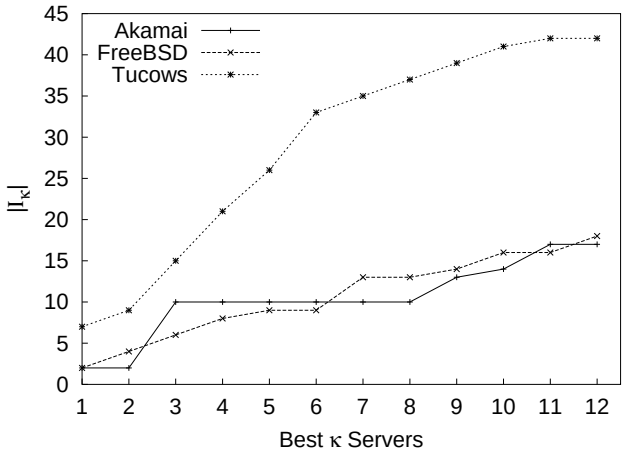


Fig. 3. The minimum size of $|I_k|$ for RTT-based selection required to map the best k servers with probability ≥ 0.9, for Akamai, FreeBSD, and Tucows mirrors.

4. Each thread independently connects to the assigned server and requests its block using FTP [22]. FTP allows servers to transfer a file starting from a specific offset by using the restart (REST) command. When a client receives the last byte in the current requested block, it sends an abort (ABOR) command to the remote server to stop sending any more bytes.
5. Periodically, as the file is being transferred, the monitor thread will dynamically adjust the block allocation to maximize the client's throughput (explained later).
6. Finally, once all blocks are retrieved, the manager reassembles the file and returns it to the user.

Theoretically, the total time to transfer a chunk of data from a specific server is bounded by the following Equation:

$$t_i = \frac{\sum_{j=1}^{N_i} B_j}{\mu_i} + N_i t_i^{request} + t_i^{connection} \quad (1)$$

where N_i is the number of data blocks requested from mirror i. B_j and μ_i represent the block size and the server's speed, respectively. $t_i^{request}$ represents the time between sending the request command to the server and receiving the first byte of the data. $t_i^{connection}$ represents the time needed to establish a connection with the remote server.

To minimize the total transfer time, the amount of data transferred from each server (the first term in (1)) is adjusted such that t_i is equal for all servers that are accessed in parallel. The overhead of requesting all blocks must also be minimized (second term in (1)). We use dynamic monitoring to perform these optimizations. By monitoring the progress of all connections, the data block of the slowest connection is reassigned to a faster one that is about to finish. This algorithm is depicted in Fig. 5 and has three properties. First, it closes a connection only if it finds an alternative that is faster. Second, fast servers will always be maximally utilized. Third, by using large blocks, it minimizes the term $N_i t_i^{request}$ in (1).

Besides improving clients' throughput, dynamic monitoring also converges to the set of fastest servers. This effectively reduces the DoP used by each client and, thus, improves the scalability of our clients. Another way of characterizing dynamic monitoring is that it minimizes most inefficiencies that were introduced by RTT-based selection (mainly having the size of the selected server set $|I_k| > k$).

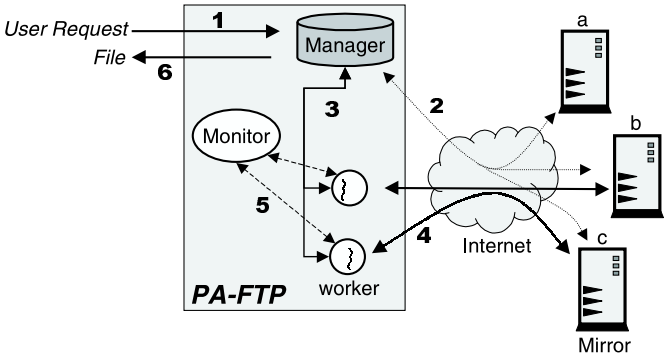


Fig. 4. Architecture of our Parallel-Access FTP client

Algorithm 1 (Block Assignment Optimization)

1. for ($i \in \{all\ connections\}$)
2. TOC(i) := estimated time of completion for connection i
3. if($TOC(i) < next\ time\ to\ perform\ optimization$)
4. Find slowest connection (max TOC)
5. if ($reallocation\ will\ improve\ transfer\ time$)
6. Kill slowest connection and assign its block to i

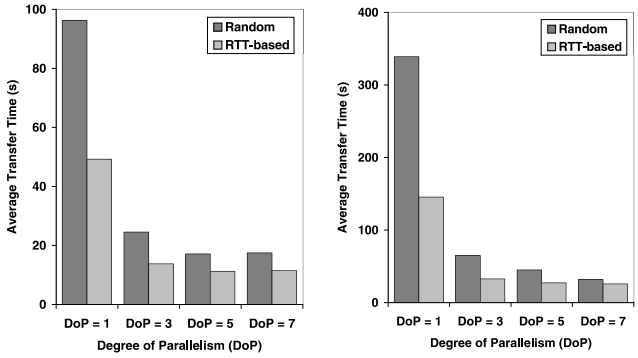
Fig. 5. Dynamic monitoring algorithm

VI. Evaluation

We verified the effectiveness of the performance criterion of our design on the FreeBSD mirrors. Using our PA-FTP client, files ranging from 100KB to 10MB are transferred 20 times while varying the DoP from one to seven. Because our clients sits behind a fast network link, files that are less than 500KB in size showed marginal improvements. Due to space limitation, we only show results for two file sizes 2.8MB and 10MB. We compared our results to that of Random selection as well as to the traditional single-server approach. Fig. 6 shows the average transfer time for each file at different DoPs for Random and RTT-based selections.

The general advantages of parallel-access are indicated by the dramatic decrease in transfer time (85%) as the DoP increases beyond 1 (DoP=1 indicates transferring from a single-server). Furthermore, the average number of failures for transfers from a single server is more than 5% while for the parallel-access approach is 0%, which shows the advantage of parallel-access scheme in masking server failures.

RTT-based selection outperforms the Random selection even at higher DoP: 50% improvement at DoP = 3 and about 35% at DoP > 3 (Fig. 6(a) and (b)). The average transfer time, however, only improves marginally when the DoP is increased from 5 to 7, especially for the RTT-based selection. This shows the property of diminishing returns due to the following rea-



(a) 2.8MB file size (b) 10MB file size

Fig. 6. The average transfer time at DoP of 1, 3, 5, and 7 for Random and RTT-based mirror selections for (a) 2.8MB and (b) 10MB file sizes.

sons:

- Requesting a smaller amount of data from each server (because we are splitting the same pie into a larger number of small pieces) will decrease the transmission time and make the overhead time significant.
- Because the connection time and request time overheads depend on the server's RTT and load, a fast server may finish transferring a block before a slow server even sends the first byte of another block. The dynamic monitoring will reassign the block from the slow server to the fast server, rendering the slow server useless.
- The down-link connection of the client is saturated, so increasing the number of servers will not increase the perceived rate.

VII. Conclusions

Using a Parallel-access approach for transferring files reduces transfer time and increases resilience to server and link failures. In general, increasing the DoP to a reasonable limit will improve the client-perceived throughput. We have showed that mirror selection plays a crucial role in determining the scalability of the parallel-access technique and the performance perceived by the end users. We also argued that RTT-based selection solves these problems by providing two key advantages. First, it correctly maps some of the fast servers into a small subset of mirrors that can be accessed in parallel. Second, it limits the probability of overlapping sets. Our implementation of the PA-FTP client that combines RTT-based selection with dynamic monitoring showed the effectiveness of our new technique in improving the transfer time and enhancing the scalability of the parallel-access.

Although, we have setup our experiments to mimic the characteristics of real scenarios as closely as possible, we believe that large scale deployment of our parallel-access technique will further provide more insight into the performance of the new approach. Nevertheless, our final conclusions are still valid.

References

[1] "Tucows inc.," http://www.tucows.com.
[2] "Akamai technologies, inc.," http://www.akamai.com.
[3] A. Miu and E. Shih, "Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet," url: http://nms.lcs.mit.edu/~aklmiu/comet/paraload.html, Dec. 1999.
[4] Pablo Rodriguez, Andreas Kirpal, and Ernst Biersack, "Parallel-Access for Mirror Sites in the Internet," *Proc. of IEEE INFOCOM '00*, pp. 864–873, March 2000.
[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements," *Proc. of IEEE INFOCOM '98*, March 1998.
[6] "Netscape Navigator," http://www.netscape.com/.
[7] J. Byers, M. Luby, and M. Mitzenmacher, "Accessing Multiple Mirror sites in Parallel: Using Tornado Codes to Speed Up Downloads," *Proc. of IEEE INFOCOM '99*, Apr. 1999.
[8] Robert L. Carter and Mark E. Crovella, "Server Selection using Dynamic Path Characterization in Wide-Area Networks," *Proc. of IEEE INFOCOM '97*, April 1997.
[9] Sandra G. Dykes, Clinton L. Jeffery, and Kay A. Robbins, "An Empirical Evaluation of Client-Side Server Selection Algorithms," *Proc. of IEEE INFOCOM '00*, 2000.
[10] P.R. McManus, "A Passive System for Server Selection within Mirrored Resource Environments Using AS Path Length Heuristics," http://proximate.appliedtheory.com/, Jun. 1999.
[11] A. Myers, P. Dinda, and H. Zhang, "Performance Characteristics of Mirror Servers on the Internet," *Proc. of IEEE INFOCOM '99*, Mar. 1999.
[12] Katia Obraczka and Fabio Silva, "Looking at Network Latency for Server Proximity," Tech. Rep. USC-CS-99-714, University of Southern California, 1999.
[13] M. Sayal, Y. Breibart, P. Scheuermann, and R. Vingralek, "Selection Algorithms for Replicated Web Servers," *Performance Evaluation Review*, vol. 26, no. 3, pp. 44–50, Dec 1998.
[14] Anees Shaikh, Renu Tewari, and Mukesh Agrawal, "On the Effectiveness of DNS-based Server Selection," *Proc. of IEEE INFOCOM '01*, 2001.
[15] M. Stemm, R. Katz, and S. Seshan, "A Network Measurement Architecture for Adaptive Applications," *Proc. of IEEE INFOCOM '00*, pp. 2C–3, Mar. 2000.
[16] V. Cardellini, M. Colajanni, and P.S. Yu, "Dynamic Load Balancing on Web Server Systems," *IEEE Internet Computing*, pp. 28–39, May-June 1999.
[17] Michael R. Garey and David S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York, 1979.
[18] "FreeBSD," http://www.freebsd.com.
[19] "Traceroute gateways," http://www.tracert.com.
[20] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks," *ACM Computer Communication Review*, vol. 21, no. 5, pp. 30–47, October 1991.
[21] P. Francis et al., "An Architecture for a Global Internet Host Distance Estimation Service," *Proc. of IEEE INFOCOM '99*, pp. 2B–1, Mar. 1999.
[22] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)," RFC 959, Internet Engineering Task Force, Oct. 1985.