On the Effect of Large-Scale Deployment of Parallel Downloading

Christos Gkantsidis, Mostafa Ammar, Ellen Zegura

College of Computing
Georgia Institute of Technology
Atlanta, GA
{gantsich, mostafa, ewz}@cc.gatech.edu

Abstract

In the presence of multiple mirror servers that can serve the same information, clients can improve their performance when downloading large files by conncurrently retrieving different parts of the file from different servers. In previous work, experiments with a single client have shown the potential for parallel downloading schemes to improve the performance perceived by this single client. In this paper we consider the question of what happens when parallel downloading is widely adopted *within the Internet. To that end we start with an experimental evaluation of various parallel downloading schemes and their performance from the perspective of a single client. We use these results as a benchmark for a set of simulations in which we investigate the performance of parallel downloading when performed by multiple clients. We find that, because parallel downloading incurs additional overhead as a result of its use of more network connections, large-scale use of this technique can actually lead to overall degradation of the performance experienced by clients. Further, we uncover a "fairness" tradeoff in that any performance improvement experienced by clients performing parallel downloading comes at the expense of clients who simply go to a single server to retrieve files.*

1. Introduction

The Internet has seen a substantial growth in traffic and popularity. As a result, popular web and ftp sites have to deal with higher loads. On the other hand, users' expectations have increased; they want to download the desired information in the shortest possible time.

Two main approaches have been proposed to deal with the increasing load and achieve short download times. The first is to use proxies to cache the information closer to the clients [19]. The other is to replicate the information in mirror servers and direct each client to the one that can provide the best performance. The client is directed either automatically (e.g., Akamai [1]) or must manually choose a server from the list of all mirror servers [17, 2].

In the presence of multiple mirror servers that can serve the same information, a novel approach to improve the performance is to connect to many servers in parallel [15]. "Parallel" in this context means that different parts of the information are requested from different servers (using, for example, the range request of HTTP [5]). This approach can greatly reduce the waiting time for the client.

In [15] the authors have studied the performance a single client would experience by using parallel downloading. An interesting and unresolved issue is to understand the impact that this idea might have in the network and the servers, if it is *widely adopted* by clients. Would the performance observed by each of the clients continue to be good? What about the performance of the system of the mirror servers? These questions are important; before the wide-scale deployment of this approach, we must understand the impact in the system. It is especially important to study the coexistence of users that employ parallel download techniques with simple users who do not.

In this paper we strive to understand the implications of large-scale adoption of parallel downloading as well as the question of service differentiation between users that use parallel downloading and those that do not. To that end we start with an experimental evaluation of various parallel downloading schemes and their performance from the perspective of a single client. Such an examination helps us understand how to best design and tune a parallel downloading scheme. It also serves to provide a performance benchmark for our evaluation. We next perform an extensive set of simulation experiments with the aim of understanding the full effect of large-scale deployment of parallel downloading by many clients. Our main conclusions can be summarized as follows:

- In general, clients that use parallel downloading can potentially experience shorter waiting times as compared to clients that download content from a single server. While this has been observed previously for single clients, our experiments confirm this possibility even when many clients use parallel downloading, but only when the overhead associated with each connection is low or moderate. When the per-connection overhead is high we have observed scenarios where parallel downloading can actually degrade performance when deployed widely.

- The performance improvement experienced by users is diminished as more of them employ parallel downloading. A large number of users using parallel downloading can introduce excessive amounts of overhead that, in turn, can result in reduced performance for all clients.

- The performance improvement that parallel-download clients experience comes at the cost of increased waiting times for clients downloading from a single server. The increased overhead that parallel-download clients introduce in the system has a more substantial effect on the clients that connect to only one server. In this sense parallel-download clients are unfair to the other users.

The rest of this paper is organized as follows. Section 2 briefly surveys related work. In Section 3 we describe several parallel download techniques and examine their performance using Internet experiments. By necessity these experiments deploy parallel downloading at a single client and examine the relative performance of the download algorithms from the perspective of this single client. We also study the question of how to best configure the algorithms to improve the performance experienced by this single client. In Section 4 we examine scenarios in which multiple clients use parallel downloading. Because of the need to perform controlled, multiple-client experiments our results in this section are based on simulations of wide-area networks as well as simulations of a simple queueing model. We conclude the paper in Section 5.

2. Related Work

The parallel download idea was first introduced by Rodriguez et al. [15]. In that work, the authors propose the history based TCP and the dynamic TCP approaches; these form the starting point for some of the techniques we describe in the next section. They have also analyzed experimentally the performance of these schemes in the Internet, but only from the perspective of a single client. The variability of their experimental configuration (number of servers, location, etc.) was quite limited. Moreover, they did not experimentally examine the impact of pipelined requests in minimizing the idle times and improving the performance of dynamic TCP clients.

Of most interest to us, this work did not study the effect that a large population of clients using parallel download can have on the performance of the network. As this technique becomes more popular and is used by commercial products like Speedbit [16], it becomes more pressing to understand how this approach will work in environments where a large percentage of users use it. This is the main target of our work.

Other proposals for performing parallel download include the use of Tornado codes as in [3] and the Information Dispersal Algorithm (IDA) [14]. These schemes require changes both in the operation of the servers and, perhaps more significantly, in how the content is encoded. In this paper we focus primarily on schemes that require changes only to the clients (not servers or content), although we expect many of our conclusions to carry over to these other schemes.

3. Single Client's Perspective of Parallel Download

The success of parallel downloading depends on the improvement in transfer times that it can give. We start in this section by examining a single-client's perspective. While this has been examined previously by [15], our experiments consider a larger set of configurations (e.g., number of servers, locations). This allows us to examine several new effects including the effect of the number of servers used for parallel downloading. Moreover, we experimentally examine the impact of pipelined requests in minimizing the idle times and improving the performance of dynamic downloading. Incorporating these features into our single-client experiments allows us to better configure the schemes for best performance and, more importantly, allows the results of the experiments to serve as an important validation as well as benchmark for our multiple-client evaluation in the next section.

3.1. Parallel downloading algorithms

We consider three parallel downloading schemes:

1. *Static-Equal*: In this approach the client downloads an equal portion of the desired file from each of the available servers. This approach is "static" since the decision of which portion of a file to retrieve from which server is made before the beginning of the download [1].

[1]Note that these schemes typically require knowledge about the file size which can be obtained external to the download operation (e.g., many sites

2. *Static-Unequal* The idea behind Static-Unequal is to make the amount of data requested from each server proportional to the server's throughput (in contrast to the Static-Equal scheme where the servers all serve equally sized portions of the file). Before the start of the download the client estimates the throughput of each server. The way this estimate is obtained is described in detail in Section 3.4. The client then requests from each server a portion of the file that is proportional to the server's throughput estimate. The name of the scheme stems from the fact that the file might be divided in unequal parts and the decision is made before the beginning of the download. A similar idea, although with the critical difference in the method used for estimating the throughout, is described in [15] as history-based TCP. The same principle is also used by commercial software such as Speedbit's Download Accelarator [16].

3. *Dynamic*: Another approach is to dynamically adjust the proportion of a file retrieved from each server during the download process to request more information from servers that are currently performing better. The basic scheme works as follows. The file is divided in many equal pieces and the client requests a piece from each of the servers. When a server finishes, then the client asks for another piece and so on. This is similar to the dynamic-TCP technique proposed in [15].

3.2. Measurement methodology

We have built a prototype client to evaluate the performance of the three schemes described above. The client is built on top of the libwww library [8] provided by the World Wide Web Consortium [18] and supports persistent connections and pipelined requests according to the HTTP v1.1 standard [5].

As our set of replicated servers we use the TuCows software library [17]. We picked twelve mirrors that are located in the United States [2]. The twelve servers representing a wide variety of distances from our site (both in terms of hop count, but also geographically). Table 1 shows the complete list of servers used in the experiments.

Our experiments involve only one client, located on our campus at Georgia Tech, Atlanta, GA. The client connects

Table 1. *List of TuCows servers that we have used in our experiments. (Locations as reported in the TuCows web site.)*

Server	Location
thecrossings.linux.tucows.com	Mountain View, CA
tde.linux.tucows.com	Wheat Ridge, CO
delaware.linux.tucows.com	Dover, DE
intermarine.linux.tucows.com	Jacksonville, FL
agentware.linux.tucows.com	Atlanta, GA
nitco.linuxberg.com	Indiana
linuxberg.mv.net	Londonderry, NH
epix.linux.tucows.com	Dallas, PA
config.linux.tucows.com	Ravenna, OH
emperor.linux.tucows.com	Austin, TX
datasync.linux.tucows.com	Biloxi, MS
infostreet.linux.tucows.com	Tarzana, VA

to nearly all possible subsets of the servers[3]. Using each possible subset we downloaded four files with sizes 144kB, 806kB, 2.5MB, and 7.1MB. We did not experiment with files of small sizes, since generally they do not benefit from parallel download. Each parallel download scheme was exercised 16328 times ($2^{12} - 1 - 1 - 12 = 4082$ valid sets, 4 objects per set) with different sets of servers and for files of different size.

3.3. Static Equal

The naive approach is to divide the file equally among the different servers. Assuming the file is of size L bytes and the number of servers is n, then each server is assigned L/n bytes.

The advantage of this approach is its simplicity, since it does not involve any decisions on the client side. The client opens n connections and makes n requests, then, it merges the replies into one file. On the other hand it is easy to see that this approach might be far from optimal.

Assume that b_{bad} is the throughput of the slowest server and that $b_{total} = \sum_i b_i$ is the sum of the throughputs from all servers. The client would hope to take advantage of all the throughput available and complete the download in time $t_{opt} = L/b_{total}$. Instead, the download finishes at time $t_{bad} = L/(nb_{bad})$. The static-equal approach is k times worse than the optimal one, where k is:

$$\begin{aligned} k &= \frac{t_{bad}}{t_{opt}} \\ &= \frac{\sum_i b_i}{nb_{bad}} \end{aligned}$$

or applications list the size of a file) or can be determined through an initial handshake with the servers.

[2]In a highly replicated system like TuCows, a client (even a parallel downloading one) is faced with the problem of selecting from among the large set of servers. Since we experimented with a client in the U.S. we presumed that the client went through a pre-selection process (which is outside the scope of this work) and ended up with reasonable selection of servers from within the U.S.

[3]We omitted the empty set, the set of all servers, and the sets of all servers but one.

This ratio depends on the ratio of the bandwidth of the best server over the worst one. Myers at al. have seen differences of over an order of magnitude in speeds of mirrored servers [12].

In our experiments we have indeed observed large values of k. For example in the case of a file of size 7.1MB, the best value of k when using two parallel servers is 1.01 and the worse is 11.84; whereas when using nine servers these values are 1.45 and 20.68 respectively. This example illustrates two general trends in experiments with Static-Equal. First, in many cases the performance achieved with Static-Equal is far from optimal. In fact, more than 60-70% of the experiments had download times more than twice the optimal times. Second, as we increase the number of servers to connect to, the gap between the optimal time and the achieved one increases.

From our experiments we have also observed that increasing the number of servers to connect to decreases the waiting time as expected. But we also observe diminishing returns: increasing the number of servers beyond 5-6 improves the performance only marginally. Further increase of the number of servers, typically more than 8 in our experiments, can even decrease the performance. This can be attributed to two factors. The first one is the increased probability of getting bad performance from a server as we increase the number of servers we use. The second reason is that by connecting to 5-6 servers our client uses all the available access bandwidth.

3.4. Static Unequal

In the Static-Unequal scheme, the goal is to assign each server a portion that is proportional to its throughput. If the throughput to each of the servers is b_i, then the goal is to have server i serve $L_i = Lb_i / \sum_j b_j$ bytes and hence achieve a completion time equal to $t_{opt} = L/\sum_i b_i$. Since the client has no way of knowing the exact b_i's, it has to use estimates b'_i instead. In this case the download time of the Static-Unequal method is k times worse than the optimal one, where:

$$\begin{aligned} k &= \frac{t_{bad}}{t_{opt}} \\ &= \frac{\sum_i b_i}{\sum_i b'_i} \cdot \max_i \frac{b'_i}{b_i} \end{aligned}$$

Obviously, the performance of this scheme depends critically on the quality of the throughput predictions. Unfortunately, there is no perfect way of predicting the future performance of a server. In this paper, we use previous measurements of the performance to each of the servers and compute the next estimate b_i^n of the throughput that server i can offer using the standard moving weighted average:

$$b_i^n = \alpha b_i^{n-1} + (1-\alpha)\, b_i^{real}$$

where b_i^{real} is the throughput measured during the experiment. The parameter α determines the weight of the previous predictions on the computation of the new one. For our experimental configuration values of α between 0.5 and 0.6 gave the best results. The value of α that we used is 0.59.

As in the Static-Equal method, we observe that by increasing the number of servers that our client connects to, the performance also increases. Again, though, increasing the servers beyond 5-6 servers yields only marginal additional improvements and increasing them more than 8 can even decrease performance. The reasons are the same as those given for the static-equal method.

We again compute the ratio k to compare the download times achieved under this method with the optimal download times. We find that for a file size of 7.1MB and for two servers, the best value of k was 1.00 and the worst value was 1.93. For nine servers, the values were 1.09 and 5.75, respectively. We also observe that 10-30% of the experiments had download times more than twice the optimal ones. This number suggests that the Static-Unequal method performs much better than the Static-Equal one, at least for our environment.

The above numbers show that even in this method there is room for improvement, especially in the case of larger numbers of servers. We have measured the ratio of the size of the job that was assigned to each of the servers and the size that would have being assigned to them, if we knew in advance their throughputs during the experiment. In many cases the ratio was more than 10. This suggests that large portions of the file were sometimes assigned to relatively slow servers, despite the fact that we always had a lot of recent measurements for each of the servers and that our estimation technique was particularly tailored for our testbed.

3.5. Dynamic

In the previous two techniques the partitioning decisions are made in advance and cannot be changed later to match the network and server conditions at the time of the download. The third scheme is dynamic, which means that the client makes decisions while in the process of downloading. It works as follows. The client asks each server for a piece of the file of size l bytes. l is fixed and typically $l << L$. When the server finishes, the client asks for another part (again of size l) from that server, and this process continues until the whole file has been requested. If all parts of the file have been requested and one of the servers finishes serving a request, while the others are still busy, then we pick the oldest pending request and re-send it to that server

There are two main advantages of this scheme. The total number of bytes downloaded from each of the servers is proportional to the actual throughput of the server at the time of download. Also, the scheme is adaptive to changes

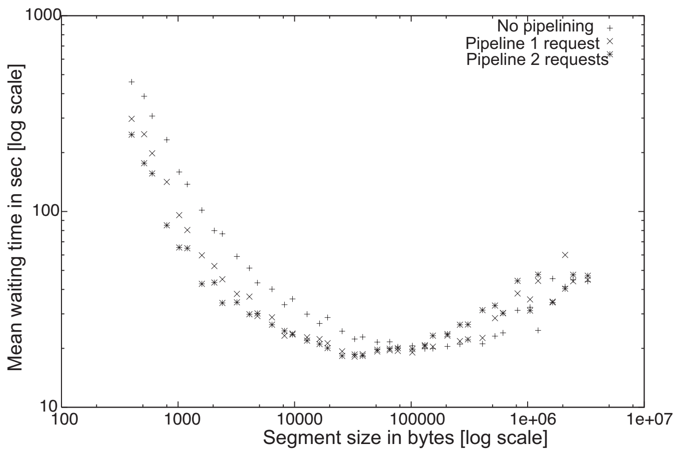

Figure 1. Effect of segment size and pipelining on download times for an object of size 7.1Mb. Eight parallel servers are used.

in the available bandwidth on the path from the server or change in the server's load.

A crucial parameter for the performance of this scheme is the choice of l. If the value of l is high, then the granularity of assigning work to servers is coarse, and the number of bytes requested from each of the servers will not be in direct proportion to the server throughput. A fine granularity is achieved with small values of l. But, in this case there is more overhead in sending requests to servers. Each request costs a round trip time in which the server does not send any new data to the client. Also, it is possible that the arrival of the next request at the server will be delayed and the TCP connection will move to the slow start phase [11]. More requests mean more signaling overhead and more resources used in the server to parse and start serving the request. Finally, very small values of l may also mean that the server's TCP connection never exits the slow start phase.

It is possible to eliminate the idle times between the finish of one request and the beginning of the next one by pipelining requests at the server. If there is a pending request in the server, it will be served immediately after the end of the previous one.

Figure 1 shows the performance of the Dynamic scheme as a function of the segment size and the number of pipelined requests. We observe that the optimal values for l, in our configuration are in the range of 50000-100000 bytes. We therefore use a value of $l = 100000$ in the experiments in the remainder of this section and in the simulation results presented in the next section.

As in the previous schemes an important configuration parameter is how many servers to connect to. We have experimented with different numbers and have discovered that the performance increases as more and more servers are used, but the additional improvement is very small if more than 5-6 servers are used. Moreover, using a very large number of clients, more than 8 in our experiments, can even decrease the performance. This trend is shown in Figure 2, which plots the best 25 download times of all the measurements for a varying number of parallel servers used.

4. Parallel Downloading with Multiple Clients

In the previous section we have seen that parallel download can significantly improve the mean waiting time of a single parallel download client. The natural question now is whether such an approach is good for the Internet as a whole and whether one would expect similar performance improvement if *all or a large number* of clients were to use parallel downloading.

Consider, for example, the following scenario where m clients are downloading the same file from a set of n replicated servers. We argue that the experimental results described in Section 3 and the results in [15] are only applicable to the scenario where one of the clients downloads the file in parallel from the n servers and the other $m - 1$ clients each downloads from one of the servers. This is true since in these experiments all clients (except our parallel downloading client and, possible, some few others) are downloading in the normal manner from the tucows servers (and the Squid servers in [15]) These single-parallel-client results clearly demonstrate the advantage that this client would have over other non-parallel clients. The obvious question is whether the same advantage would be observed if all the m clients were to parallel download from all the n servers. Clearly in such a scenario, the server load is considerably increased where each server would have to deal with m clients when all clients use parallel downloading as opposed to an average of m/n clients when all (or all but very few) clients download from a single server. So at least from the server load perspective there is the potential that large-scale use of parallel downloading can be burdensome.

From a network throughput point of view it is clear from the single-parallel-client results that path diversity afforded by parallel downloading can be a significant advantage in avoiding bandwidth bottlenecks. It is less clear that this will also be the case when all clients attempt to take advantage of path diversity.

Another important question is whether clients that are able to perform parallel downloading actually penalize clients that do not have this capability. This is really a fairness question. Note that we already know that a client that performs parallel downloading will experience better performance than the *same client* downloading from a single server. The question is whether the large-scale use of parallel downloading actually penalizes clients that do not

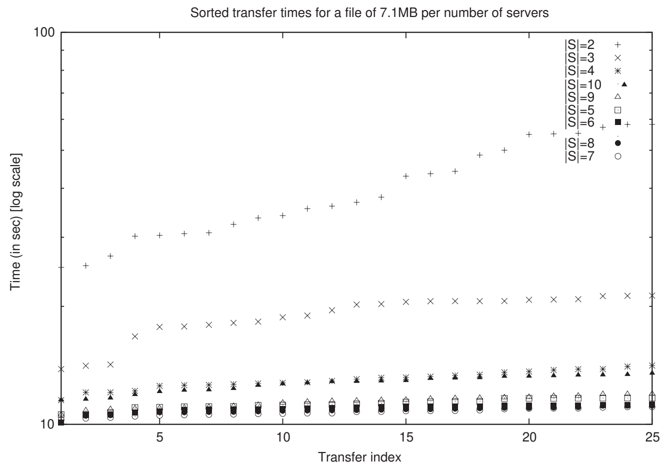

Figure 2. Waiting times for the best 25 downloads when connecting to n=2, 3, ..., 10 servers. Object size is 7.1Mb.

possess this capability. Is the performance improvement achieved by parallel downloading at the expense of clients without this capability?

Our goal in this section is to answer these questions. Ideally, we would like to perform Internet experiments similar to the ones conducted earlier. We are interested, however, in considering scenarios with a large number of clients. We further desire to provide more control and repeatability than would be possible using Internet experiments. We have, therefore, conducted a set of ns-2 simulations as described below.

What we learned from our evaluation in this section can be summarized in two points:

1. Parallel downloading when performed by all or a large proportion of clients can actually lead to a degradation of the performance experienced by clients.

2. In cases where parallel downloading clients do experience a performance improvement, such an advantage is typically at the expense of clients without the parallel downloading capability.

4.1. Methodology

Our ns-2 simulations run on a random network topology produced by GT-ITM [6] consisting of 100 nodes. Ten server nodes were added to our sample topology. Each of them was attached to one router, which was chosen uniformly at random from the set of all routers. A similar approach was followed to connect the clients to the network. We have experimented with three topologies that had 100, 200, and 300 clients respectively, representing low, medium and high load. Connections between clients and servers are always made using the FullTCP model of ns-2 which incorporates the overhead of connection establishment as well as the congestion control dynamics of TCP. We also inject some background load into the network using on-off UDP flows scattered on random paths in the network.

Each client is initially idle. At some random point in time it becomes active and picks a random set of servers used for a parallel download of a 1 MB file. The number of servers in the set is a configurable parameter. A *simple* client contacts only one server and sends only one request. If a client implements one of the static methods, it contacts more than one server, divides the file among them and sends one request to each. Static-Unequal clients record the throughput on the path from each of the servers at the end of the download and use it to estimate the performance on future requests. If a client implements the Dynamic method, all the requests and replies to the same server use the same TCP connection thus simulating persistent connections. After the end of the download, the client closes all TCP connections and returns to the idle mode, in which it will stay for a randomly chosen amount of time taken from an exponential distribution with mean 200sec. We use the results of the previous section to optimally configure the algorithms. All performance metrics are computed for events that finish after the system reaches steady state.

This type of simulation makes some assumptions about the underlying transport protocol that is used to provide this service. It is important to make sure that the results really reflect the effects of parallel downloading and are not skewed by artifacts of the simulation, or the combination of TCP and parallel download. In order to do so we use a very simple queueing model which is depicted in Figure 3. This type of queueing network is known in the lit-

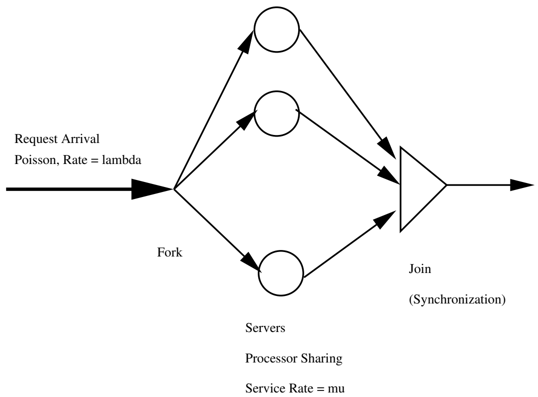

Figure 3. Queueing model

erature as a Fork-Join system and has been well studied ([13, 7, 9, 10, 4]). In this model clients arrive according to a Poisson process with rate λ and fork multiple requests that are sent to the N servers. Each of the servers implements the processor sharing service discipline and the dedicated (unshared) service rate is μ per file. Thus, if the requests are for $1/k$-th of a file, then the service rate is $k\mu$. A client waits in the join node for all the parallel requests it made to finish before it exits. This model captures some of the essential properties of parallel downloading and is agnostic to specifics of the protocols used and the network topology. Thus, the queueing network models different parts of the system than the ns simulation and thus the results we get with the queueing model should not be directly compared with the results we get with the ns simulation; rather, they should be viewed as complements. The queueing model is closer to scenarios where customers compete for resources in the servers and not in the network. We will find it useful to quote results from this model in some instances as it gives some insight into the performance of parallel downloading in general.

4.2. All parallel-download clients

Our first set of results are for scenarios where all the clients in our network perform parallel downloading from all the servers. We examine different schemes but assume that all clients implement the same scheme.

We first attempt to use our simulation to replicate the results from the single-client Internet experiments in Section 3. Figure 4A shows sample results from a set of Internet experiments showing the average client waiting time for varying number of servers and for the three download techniques we consider. We simulated a single client in conditions similar to those observed in the Internet experiments (including RTTs and packet drop probabilities) and the results are shown in Figure 4B. This figure also contains "simple" (i.e., single server) client results. While the results are not identical, they are close and more importantly provide roughly the same relative ordering (ignoring small differences) of the schemes with Static-Unequal and Dynamic schemes performing the best. Figure 4C shows the results when 100 clients perform parallel downloading from all the servers for a file of 1MB.

Comparing Figures 4A and B (the single client experiments) with Figure 4C (the all-clients experiment), we make the following important observations.

1. The difference among the various parallel downloading schemes is not as pronounced when all clients perform parallel downloading. The only exception is that the Dynamic scheme has a significant advantage when the number of servers is large.

2. The performance of the Static-Equal and Static-Unequal schemes is very similar when all clients perform parallel downloading. This is in contrast to the significant improvement achieved by the Static-Unequal scheme when only one client uses parallel downloading. The increased number of TCP flows in the network, in the case of a large number of parallel clients, makes the performance of the network more variable and more difficult to predict. Thus, the predictions made by the Static-Unequal scheme are not good resulting in a weaker performance of this scheme.

3. It is still true, in this case, that parallel downloading (even when performed by all clients) is better than when all clients simply go to a single server. As we will see shortly, however, this is not always the case.

These observations are in part due to the fact that parallel-download users introduce an extra overhead in the network, which degrades the performance observed by the clients and is not significant when there is a single client using parallel download. The source of this extra overhead is in opening and maintaining more TCP connections. This translates into more TCP flows competing in the network and in more packets injected in the network.

We should note, however, that these simulation results do not account for server load created by the maintenance of more TCP connections. We expect that if parallel downloading becomes popular, then the average number of connections at servers will increase, since clients will typically open many more connections. Moreover, the number of requests made will also increase, since instead of sending only one request to get a file, the clients will send many more. Server resources will, then, need to be increased in order to cope with the extra load introduced by the additional requests and connections. Without such an increase in server power, the effects observed here would be more pronounced.

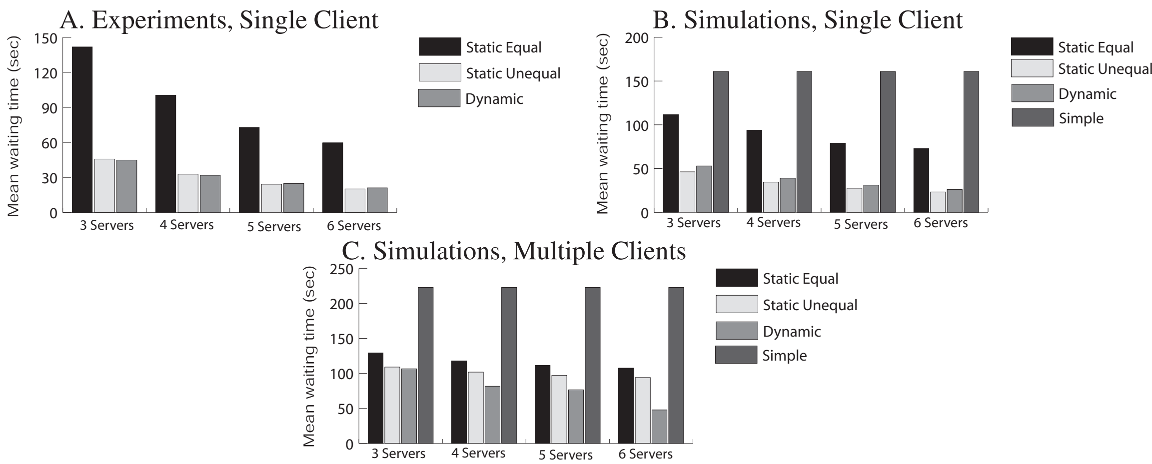

Figure 4. Experimental and simulation results for the average client waiting time as a function of number of parallel servers used and for different parallel download algorithms.

The amount of extra overhead introduced in the network increases proportionally to the number of clients that use parallel download. The results shown in Figure 5 help us examine the effect on the average performance of having more users doing parallel download. These results plot the average download time using the dynamic approach as a function of the number of servers for various client population sizes; other types of schemes give similar results. In the same figure we also plot for comparison the average waiting time for a population of simple clients.

When the load is low (100 clients), the performance gets better as the clients connect to more servers. This is in line with the results presented above. When 200 clients are accessing the servers, the performance initially improves as the number of parallel servers used increases, but later it degrades as we connect to more servers. In this case, the extra overhead, introduced to the system by the increased population of clients, plays an important role in the performance. When a small number of parallel servers is used, the per-request overhead is small and the performance improves by connecting to more servers. However, by connecting to more than 4-5 servers, additional overhead results in a drop in the performance. Again, there is an improvement by using parallel download instead of connecting to just one server. This improvement diminishes as we connect to more servers and actually the performance gets worse when performing parallel download and connecting to 7 servers.

When using 300 clients, the performance gets worse as the clients use more parallel servers. The main intuition for why the performance does not follow the same trends as in the case of 100 clients, is that the high number of clients introduces high per-request overhead in the network, which degrades average performance. It is interesting to observe that the performance with all clients parallel downloading

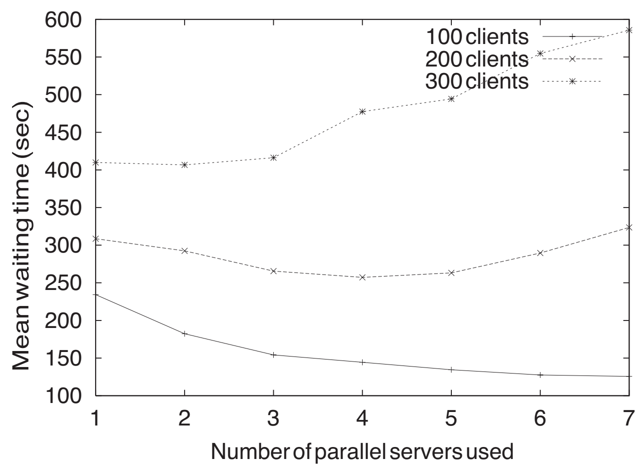

Figure 5. Simulation results for the average client waiting time as a function of number of parallel servers used. (Network composed of 10 servers and 100 routers.)

in this case is always worse than if they were to just connect one server each.

In our simulation study of parallel downloading we have experimented with schemes which connect to at most 7 servers. A natural question is whether the results would be different if we were using more servers. We have seen in the real experiments, that increasing the number of servers beyond 6 improves performance only marginally. This is not an artifact of our experimental setup, but it is inherent in the schemes. To understand why, we have performed simulations with a wider range of number of parallel servers using the queueing model. The results are shown in Figure

6. In Figure 6A, we plot the mean waiting times for both static and dynamic schemes when the system load is 0.6 and there is no constant overhead per request. We observe that increasing the number of parallel servers used to more than 7 did not improve the performance substantially. To understand the scaling characteristics of the two schemes better, we draw the ratio of the mean waiting time by connecting to only one server over the mean waiting time by connecting to n server for values of n between 2 and 15. The results are given in Figure 6B. If the schemes had perfect scaling characteristics, then we would expect that connecting to n servers, would improve the performance by a factor of n. In reality the performance achieved, even in this case in which we do not account for any other effects that can degrade the performance, is much worse.

4.3. Mix of clients

We now consider the question of how "simple" clients (i.e., clients that can only access one server for downloading) are affected by the existence of parallel-download clients. This is an important question because these simple clients will continue to exist in the network even if parallel downloading becomes prevalent. It is also important to answer whether parallel downloading is in some sense "unfair" and improves the performance for some clients at the expense of others.

It is, therefore, interesting to investigate what will happen in a mixed environment and more specifically how the users that connect to only one server will be affected by the more aggressive users that connect to many servers in parallel. Ideally, we would want the performance of the simple clients to be unaffected by the presence of the parallel ones. As we will see, this is not the case: the performance of the simple clients degrades as the fraction of parallel download users increases.

The results of our simulation experiments with mixed clients are shown in Figure 7 where the average download time is plotted against the proportion of clients using parallel downloading. We show the average performance for simple clients, parallel download clients and all clients. Interestingly, the performance of both types of clients degrades as the percentage of parallel download users increases. (Observe also that the overall performance becomes better as the proportion of parallel clients increases, since the average performance for all clients gets closer to the average performance of parallel clients.) This confirms our earlier observation that parallel downloading is most advantageous when very few clients are using it. It is interesting to observe that the average waiting time for the parallel download users increases by 27.5sec and of the simple ones by 62.6sec. In this case, the population of simple users pays a higher performance penalty for the increased load that the parallel download users have introduced in the network.

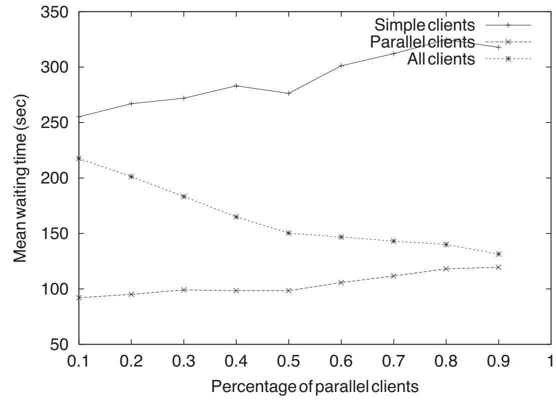

Figure 7. Simulation results for the average simple and parallel client waiting time as a function of the fraction of parallel clients. 100 clients used, 10 servers in total, 100 routers. Notice that the x-axis is the proportion of parallel clients. As the percentage of parallel clients increases the overall performance approaches the performance perceived by parallel clients.

The average performance over all clients improves as the number of parallel downloading clients increases. This seems to point to the effectiveness of parallel downloading from a global perspective. As we will see shortly this global improvement is very much dependent on the overall network load.

In our simulation the overhead is derived from TCP connection and slow-start overhead and the simulation does not model the overhead incurred at servers. Since this latter overhead can be variable we attempt to gain some insight into its effect using the simple queueing model described earlier.

To model overhead in the queueing model, we divide the mean service time for the entire file into two parts; one which does not depend on the size and one that does. The first represents the overhead. We can express the mean service time per file as $1/\mu = \alpha/\mu + (1-\alpha)/\mu$, where α is the portion of the service time which is size independent. If we are interested for $1/k$-th of the file, then, the mean service time would be $\alpha/\mu + (1-\alpha)/(k\mu)$. Thus, to compute the service time, we draw a random value from an exponential distribution with the above mean. The results for three different levels of overhead, which we call low ($\alpha = 0$), medium ($\alpha = 5\%$) and high ($\alpha = 10\%$), are shown in Figure 8. With low overhead, we observe that the performance of both types of users is not affected by the percentage of parallel users. With medium overhead, we observe that the

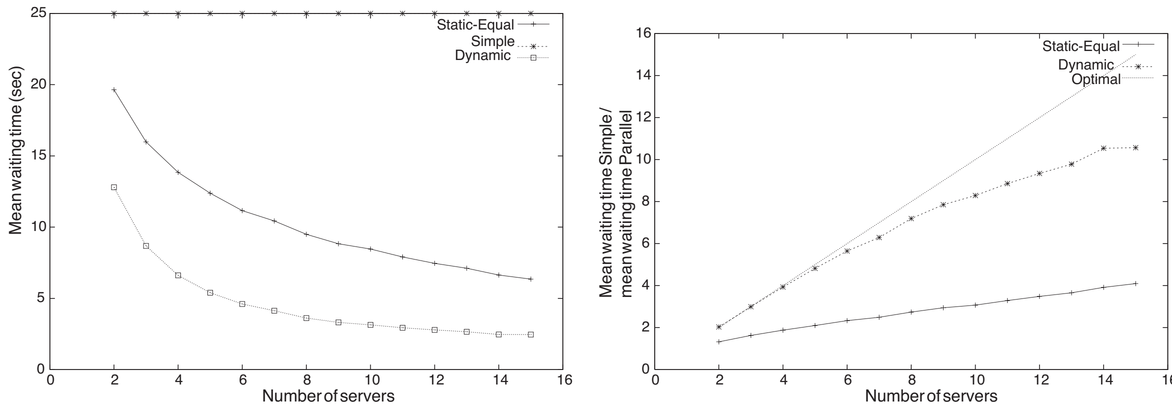

Figure 6. Mean waiting time for the simple queueing model with a system utilization of 0.6, no extra per-request overhead, and for various parallel download schemes. Left figure shows the average performance and the right figure the scaling as the number of parallel servers increases.

performance of both types of users degrades slightly and the overall performance improves as we increase the percentage of parallel users. This is basically the behavior we observed in our simulation which only accounted for TCP overhead. When the overhead is relatively high (at 10% of transmission time) we observe that the performance of both types of clients degrades substantially. The simple users are hurt most by the overhead even though they are not responsible for it. More importantly, perhaps, is the fact that overall performance remains unchanged as more clients use parallel downloading. It is clear that in this type of scenario, parallel downloading does not provide any performance improvement whatsoever while at the same time causing the network to be unfair to simple clients.

5. Concluding Remarks

The evaluation of new ideas and algorithms that are proposed to be used in Internet must not only rely on experiments with a single client, but also must try to understand what can happen if they are widely deployed. This work targets such a question focusing on the effects that parallel downloading can have, if widely deployed. Using experiments and simulations, we have studied the performance of three parallel download algorithms. We have investigated the performance of these schemes both from the perspective of a single user and in configurations that include a large number of parallel download users.

In some scenarios, we observe improved performance for the clients using parallel downloading as expected and in line with previous work. However, we have found a significant difference between the improvement predicted by experiments with only one user and those obtained when many users use parallel downloading techniques. One main conclusion is that the performance advantage of using parallel downloading techniques as seen by a client will drop as more and more users employ them. In other scenarios, it was interesting to discover that when a large number of clients use parallel download, the performance actually degrades. Finally, we have observed that the improvement of parallel downloading comes at the expense of clients that do not have such a capability. While this “unfairness” is undersireable in some cases, it can present an opportunity for providing a framework for client differentiation. We will explore this question in our future research.

References

[1] Akamai’s web site. http://www.akamai.com, February 2002.

[2] The apache software foundation. http://www.apache.org, February 2002.

[3] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *INFOCOM (1)*, pages 275–283, 1999.

[4] A. Duda and T. Czachórski. Performance evaluation of fork and join synchronization primitives. *Acta Informatica*, (24):525–553, 1987.

[5] J. G. et al. Hypertext transfer protocol — http/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616.html, June 1999.

[6] Modeling topology of large internetworks — gt-itm. http://www.cc.gatech.edu/projects/gtitm/, April 2002.

[7] P. Heidelberger and K. S. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Transactions on Computers*, 32(1):73–82, 1983.

[8] Libwww - the w3c protocol library. http://www.w3.org/Library/, April 2002.

[9] Y. C. Liu and H. G. Perros. A decomposition procedure for the analysis of a closed fork/join queueing system. *IEEE Transactions on Computers*, 40(3):365–370, 1991.

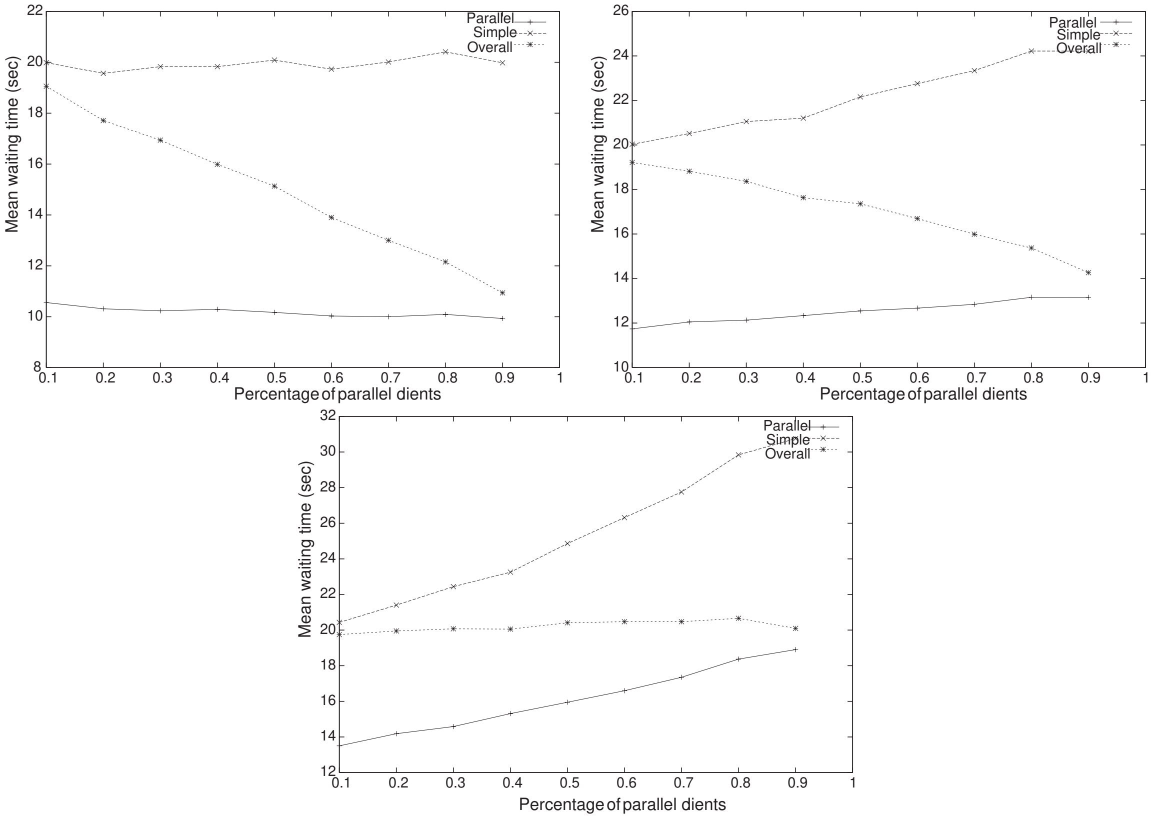

Figure 8. Performance of static equal parallel and simple clients as a function of the fraction of parallel users and as a function of overhead. Top left figure is for the case of no overhead, top right figure is for medium overhead, and bottom figure for high overhead. System utilization is 0.5. 5 servers in total. Notice that the x-axis is the proportion of parallel clients. As the percentage of parallel clients increases the overall performance approaches the performance perceived by parallel clients.

[10] J. C. S. Lui, R. R. Muntz, and D. Towsley. Computing performance bounds for fork-join queueing models. Technical Report CS-TR-94-09, 1994.

[11] W. S. M. Allman, V. Paxson. TCP congestion control. IETF RFC 2581, April 1999.

[12] A. Myers, P. A. Dinda, and H. Zhang. Performance characteristics of mirror servers on the internet. In *INFOCOM (1)*, pages 304–312, 1999.

[13] R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, 1988.

[14] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[15] P. Rodriguez, A. Kirpal, and E. Biersack. Parallel-access for mirror sites in the internet. In *INFOCOM (2)*, pages 864–873, 2000.

[16] Speedbit's download accelarator. http://www.speedbit.com, February 2002.

[17] Tucows's web site. http://www.tucows.com, February 2002.

[18] World wide web consortium. http://www.w3.org, April 2002.

[19] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.