A Top-10 Approach to Prefetching on the Web

Evangelos P. Markatos and Catherine E. Chronaki
Institute of Computer Science (ICS)
Foundation for Research & Technology – Hellas (FORTH)
P.O.Box 1385
Heraklio, Crete, GR-711-10 GREECE
tel: +30 81 391 655, fax: +30 81 391 661
markatos@csi.forth.gr

August 1996

Technical Report 173, ICS-FORTH. Available from http://www.ics.forth.gr/proj/arch-vlsi/www.html

Abstract

In the World Wide Web bottlenecks close to popular servers are very common. These bottlenecks can be attributed to the servers' lack of computing power and the network traffic induced by the increased number of access requests. One way to eliminate these bottlenecks is through the use of caching. However, several recent studies suggest that the maximum hit rate achievable by any caching algorithm is just 40% to 50%. Prefetching techniques may be employed to further increase the cache hit rate, by anticipating and prefetching *future* client requests.

This paper proposes a Top-10 approach to prefetching, which combines the servers' active knowledge of their most popular documents (their Top-10) with client access profiles. Based on these profiles, clients request and servers forward to them, regularly, their most popular documents. The scalability of the approach lays in that a web server's clients may be proxy servers, which in turn forward their Top-10 to their frequent clients which may be proxies as well, resulting in a dynamic hierarchical scheme, responsive to users access patterns as they evolve over time. We use trace driven simulation based on access logs from various servers to evaluate Top-10 prefetching. Performance results suggest that the proposed policy can anticipate more than 40% of a client's requests while increasing network traffic by no more than 10% in most cases.

1 Introduction

Recent results suggest that the World Wide Web traffic continues to increase at exponential rates [9]. One way to reduce web traffic and speed up web accesses is through the use of caching. Caching documents *close* to clients that need them, reduces the number of server requests and the traffic associated with them. Unfortunately, recent results suggest that the maximum cache hit rate that can be achieved by any caching algorithm is usually no more than 40% to 50% - that is, regardless of the caching scheme in use, one out of two documents can not be found in the cache [1]. The reason is simple: Most people *browse* and *explore* the web, trying to find new information. Thus, caching *old* documents is of limited use in an environment where users want to explore *new* information.

One way to further increase the caching hit ratio is to *anticipate* future document requests and *preload* or *prefetch* these documents in a local cache. When the client requests these documents, the documents will be available in the local cache, and it won't be necessary to fetch them from the remote web server. Thus, successful prefetching reduces the web latency observed by clients, and lowers both server and network load. In addition, off-line prefetching activated after-hours when there is plenty of bandwidth at low rates, may reduce overall cost and improve performance. At the same time, prefetching facilitates off-line browsing, since *new* documents that the user will most probably be interested in, are automatically prefetched.

Unfortunately, it is difficult, if not impossible, to guess future user needs, since no program can predict the future. In this paper we propose Top-10, an approach to prefetching that addresses successfully the above concerns through the use of *server knowledge* to locate the most popular documents, *proxies* to aggregate requests to a server and amortize the costs of prefetching over a large number of clients, and *adaptation* to each client's evolving access patterns and needs.

Server Knowledge: In the web, it is well known, that "popular documents are very popular" [9]. Thus, for each server, a small set of its files amounts for the largest percentage of web requests to that server. We call this set of documents Top-10, for the most popular documents of a server. Only documents that are members of the Top-10 are considered for prefetching by the clients. The actual number of documents in the Top-10 is fine-tuned based on client profiles which reflect the volume of requests initiated in the recent past by the client and the amount of disk space allocated for prefetched documents.

This idea of prefetching only popular items is not new: it has been followed for several years now by music shops. A significant percentage of a music store's stock contains the most popular LPs of each week. Both music store owners (proxies) and music store customers (users) make their purchases based on the Top-10. The actual purchases themselves determine next week's Top-10, which will determine future purchases and so on. The whole process of creating a Top-10 chart, distributing it to popular music magazines and channels, and making purchases based on the Top-10 of the current week, is being successfully used by millions of people each week all over the world.

Proxying: The average client makes a small number of requests to each server. Arlitt and Williamson [2] report that at least 30% of a server's clients make only one request and never request anything from the same server again. If a client is going to request only one document from a server, prefetching makes no sense. If, however, clients go though a proxy, prefetching documents at the proxy may improve performance significantly, since documents prefetched on behalf of one client may also be used by other clients as well.

Adaptation: The volume of requests generated by various web clients differs. Some of them (esp. proxies) generate large numbers of requests, while others request only a few documents. Top-10 prefetching has been designed to adapt to these cases, and allow frequent users to prefetch lots of documents, while occasional users are allowed to prefetch few documents if any at all. Prefetching is controlled by the access profile of the client. This profile contains the number of documents the client has requested from each server in the recent past, in order to determine how many documents should be prefetched from each server in the future. This way, if the access patterns of the client change, Top-10 prefetching will follow the trend and start prefetching from the currently popular servers.

In the rest of the paper we will describe Top-10 prefetching in more detail and present performance results based on trace-driven simulation. In section 2 we outline client-proxy-server framework on which Top-10 prefetching can be applied. In section 3 we use trace-driven simulation based on server traces from five different universities, research institutes, and Internet providers to quantify the performance of prefetching documents using Top-10. The performance results are very encouraging suggesting that when using Top-10, web proxies are able to successfully prefetch up to 60% of their future requests, with a corresponding increase in network traffic only in the order of 20%. To put it simply, more than half of the future requests can be predicted and prefetched. Section 4 surveys related work, and places Top-10 in the appropriate context. Section 5 discusses various aspects of Top-10. Finally, section 6 summarizes and concludes the paper.

2 Top-10 Prefetching

The Top-10 approach to prefetching is based on the cooperation of clients and servers to make successful prefetch operations. The server side is responsible for periodically calculating a list with its most popular documents (the Top-10) and serving it to its clients. Actually, quite a few servers today calculate their 10 most popular documents among other statistics regularly (e.g. see `http://www.csd.uch.gr/usage/`). Calculating beyond the 10 most popular documents is an obvious extension to the existing functionality.

To make sure that documents are prefetched only to clients that can potentially use them, Top-10 does not treat all clients equally. Time is divided in intervals and prefetching from any server is activated

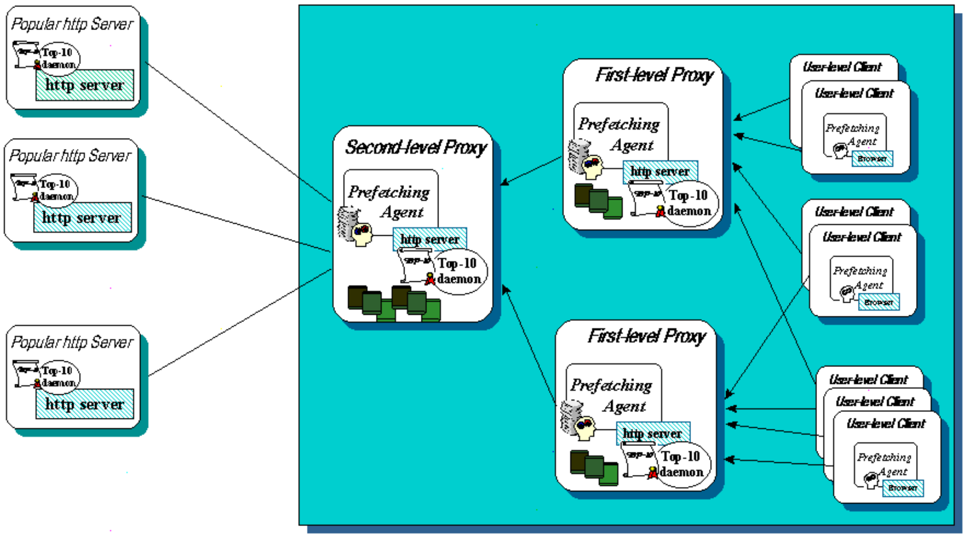

Figure 1: Top-10 prefetching operates in a client-proxy-server framework.

only after the client has made sufficient number of requests to that server (>`ACCESS_THRESHOLD`). Thus, no documents are prefetched to *occasional* clients, while *frequent* clients are identified and considered for prefetching.

Some clients, i.e. proxies, make much more requests than others, and a correctly designed algorithm should prefetch different number of documents to different clients. For example, it makes no sense to prefetch the 500 most popular documents to a client that made 10 requests during the previous time interval. Taking the recent past as an indication of the near future, that client will make around 10 requests in the next time interval and at most $10/500 = 2\%$ of the prefetched documents will be used. On the other hand, a client that made 20,000 requests during the previous interval will benefit from prefetching the 500 most popular documents, and even more than those. Top-10 prefetching adjusts the amount of prefetching to various clients based on the amount of requests made in the recent past. Along those lines, a client may not prefetch more than the number of documents it accessed during the previous time interval.

Finally, to make sure that Top-10 policy can be more or less aggressive when needed, the `TOP-10` parameter defines the maximum number of documents that can be prefetched during *any* time interval from *any* server. Thus, at any point, a client can not prefetch more than `TOP-10` documents even if it accessed lots of documents during the previous interval. By choosing a large value for `TOP-10`, prefetching can be very aggressive. On the other hand, small values of `TOP-10` limit the extend of prefetching.

Summarizing, the Top-10 approach to prefetching has two safeguards against letting prefetching getting out of control: (i) `ACCESS_THRESHOLD` which identifies *occasional* clients and does not prefetch documents to them, and (ii) `TOP-10` which can practically deny prefetching even to *very frequent* clients. We believe these safeguards are enough to control the extent of prefetching.

2.1 Client-Proxy-Server Prefetching Framework

We envision the operation of Top-10 prefetching in a client-proxy-server framework (see fig. 1). Prefetching occurs both at the client and the proxy level. User-level clients prefetch from first-level proxies to cater the needs of particular users. The benefits and costs of prefetching on user-level clients are discussed in section 3.2. First and second-level proxies play both the client and the server role. First-level proxies are clients to second-level proxies and prefetch and cache documents for user-level clients (ie. browsers). Second-level proxies are clients to various popular servers from which they prefetch and cache documents to be served to their own clients. The performance results of Top-10 prefetching at first and second level proxies are discussed in sections 3.3 and 3.4 respectively.

We picture first-level proxies at the department level of companies or institutions and second-level proxies at the level of organizations or universities. Eventhough this framework implies a structure, this structure is dynamic and may support dynamic proxy configuration schemes. In any case, Top-10

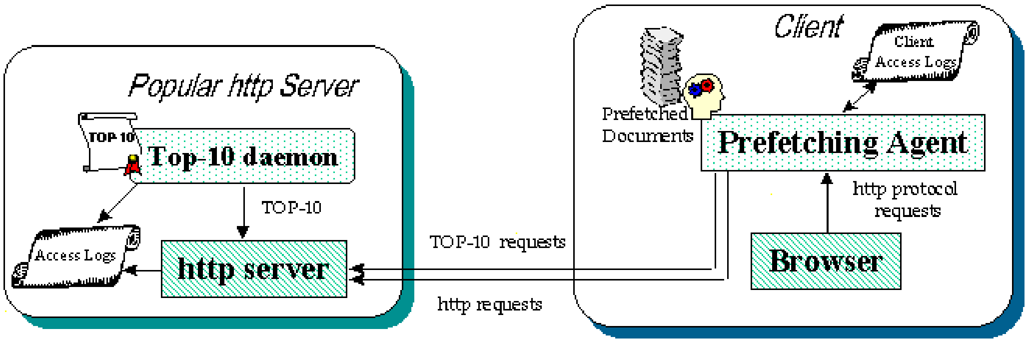

Figure 2: Top-10 prefetching depends on the cooperation of the various http servers and a client-side prefetching agent

prefetching may be *transparent* to the user and cooperate with the caching mechanisms of the browser or the proxy.

The implementation of Top-10 prefetching is based on the cooperation of server and client-side entities (see fig. 2). On the server-side, the `Top-10 daemon` processes the access logs of the server, and compiles the Top-10, the list of the most popular documents on that server. Then, it updates a web page presenting this information and the Top-10 is served as yet another document by the http server. The frequency of evaluating the Top-10 depends on how frequently the content on the server changes. In section 3.5 we investigate this issue further.

On the client side, the `prefetching agent` logs all http requests of the client and adapts its prefetching activity based on them. The prefetching agent co-operates with a proxy that filters all http requests initiated by the client. If an http request can be served from the local cache of prefetched documents, the proxy serves the document from the cache. Otherwise, it forwards the request to the web server or the next level proxy. Daily or weekly, depending on the configuration, the prefetching agent goes through the client access logs which contain all http requests made by the client and creates the prefetching profile of the client, that is, the list of servers from which prefetching should be activated. The number of documents requested from any of those servers during the previous time interval exceeds the `ACCESS_THRESHOLD`. Finally, based on the prefetching profile of the client, the prefetching agent requests the most popular documents from the servers which have been activated for prefetching. The number of documents prefetched from each server is equal to the number of requests to that server during the last time interval, or the `TOP_10` whichever is less.

Although the details of prefetching Top-10 documents can be fine-tuned to suit each client, the underlying principle of prefetching only popular documents is powerful enough to lead in successful prefetching. An advanced prefetching agent may request and take into account additional parameters like document size, percentile document popularity, and client resources, to make a more informed decision on what should be prefetched.

3 Experiments

3.1 Experimental Environment

Server Traces: To evaluate the performance benefits of prefetching we use trace-driven simulation. We have gathered server traces from several Web servers from a variety of environments that include universities, research institutions, and Internet providers both from Europe and the States. All traces total more than four million of requests. Specifically the traces are from:

- **Parallab (www.ii.uib.no)**: A Supercomputing Center associated with the University of Bergen, Norway. These traces represent heavy traffic, especially from all over Norway.
- **FORTH (www.ics.forth.gr)**: Traces from the the Institute of Computer Science, FORTH, one of the largest Research Organization in Greece.

- **Rochester (www.cs.rochester.edu)**: Traces from the Computer Science Department of the University of Rochester, NY, USA.
- **NASA**: Traces from the Web Server at NASA's Kennedy Space Center.
- **FORTHnet (www.forthnet.gr)**: Traces from the Web server of the first and largest internet provider in Greece.

We believe that it is very important to use traces from a variety of sources, since different server traces display access patterns from different client bases.

The characteristics of the traces from our servers are summarized in the following table:

Name	Duration	Total Requests
FORTH	3 Nov 95 - 28 Dec 95, 6 Jun 96 - 17 Jul 96	328,070
Rochester	18 Nov 95 - 23 Dec 95	499,073
Parallab	24 Feb 96 - 22 Apr 96	843,442
NASA	1 Jul 95 - 31 Jul 95	1,697,7364
FORTHnet	26 May 96 - 28 Jul 96	1,330,413

We preprocessed the original traces and removed requests to "cgi" scripts. We have also removed requests from local (within the same domain) clients, since local users tend to reload their pages frequently (e.g. while changing them), thereby creating an artificial popularity for some pages.

Performance Metrics: The performance metrics we use in our experimental evaluation are the *Hit Ratio*, and *Traffic Increase*. The Hit Ratio is the ratio of the requests that are serviced from prefetched documents, to the total number of requests. It represents the "guessing ability" of our algorithm. The higher this ratio is, the lower the client latency and the server load.

The *Traffic Increase* is the increase in traffic due to *unsuccessfully* prefetched documents. Since no program can predict the future, some of the prefetched documents will not be actually requested, and thus, they should not have been prefetched in the first place.

The design of a prefetching policy, is the art of balancing the conflicting factors of *Hit Ratio* and *Traffic Increase*, while trying to guess future requests. At one extreme, if an algorithm never prefetches any documents, it will not suffer any traffic increase, but it will also have zero hit ratio. At the other extreme, a very aggressive prefetching algorithm may prefetch the entire Web (assuming enough resources), but this would saturate the network with prefetched documents that will never be requested.

In all our experiments we assume that a prefetch document stays in the cache for the entire duration of a time interval, so that a small cache will not distort our results. This assumption is not unrealistic, however, since the cost of magnetic disks is low, usually lower than network bandwidth cost.

3.2 The benefits of Prefetching

In this first set of experiments we investigate the costs and benefits of our Top-10 prefetching approach. Figures 3 and 4 plot the hit ratio and the traffic increase as a function of the `TOP-10`: the maximum number of documents that any client, no matter what its access history is, can prefetch within a time interval. The time intervals in these experiments are chosen to be 50,000 client accesses long. You may observe that for all servers, as the size of the `TOP-10` increases, the hit ratio increases as well; which is as expected, since the more documents a client is allowed to prefetch, the better its hit ratio will be.

FORTHnet has the best hit ratio of all servers. Therefore, prefetching from FORTHnet results in high performance. To understand this performance advantage we need to grasp the dimensions that influence prefetching in general, and the hit ratio in particular. The hit ratio is high when (i) lots of prefetching is being done, and (ii) this prefetching is successful. Top-10 prefetches documents to *repeated* clients which are those that visit a server during successive time intervals. Additionally, Top-10 prefetches large volumes of documents to *heavy* clients which are clients that access lots of documents during a time interval. Thus, the more repeated and heavy clients a Web server has, the higher the hit ratio is going to be.

It turns out that FORTHnet has the largest percentage of repeated clients (23.5%) as figure 5 suggests. Effectively, one out of four FORTHnet clients visit for at least two successive time intervals. Since

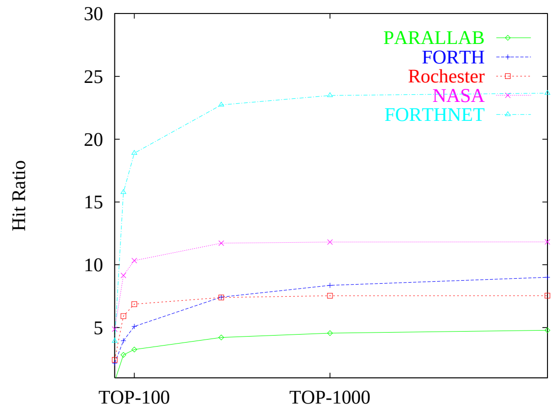

Figure 3: Successfully Prefetched documents as a function of the size of the `TOP-10`.

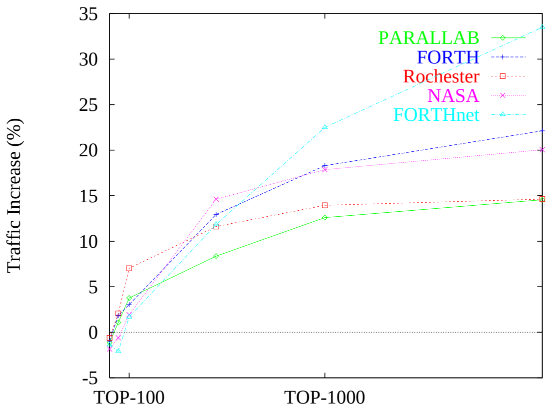

Figure 4: Traffic Increase as a function of the size of the `TOP-10`.

Server	Repeated Clients
FORTH	5.4%
Rochester	16%
Parallab	15.6%
NASA	22.6%
FORTHnet	23.5%

Figure 5: Percentage of repeated clients of each server. Observe that 23.5% of FORTHnet's clients (vs. 5.4% of FORTH's) are repeated i.e. visit during two successive time intervals.

FORTHnet has more repeated clients than any other server, it has the potential for prefetching to more clients. Moreover, FORTHnet (almost) has the most heavy clients as well as figure 6 suggests. Actually, the 10 best FORTHnet clients amount for 12% of FORTHnet's requests, the largest percentage in any of the servers we studied. As FORTHnet has both heavy and repeated clients, its clients benefit by Top-10 prefetching.

Going back to the hit ratio in figure 3, we see that the performance of the NASA server and the FORTH server follow that of FORTHnet. This is as expected, since, NASA has lots of repeated clients, but few heavy clients, and FORTH has lots heavy clients, but few repeated clients. Finally, Rochester and Parallab follow with lower hit rates, since neither of them has particularly large numbers of repeated or heavy clients. It is interesting to note however, that although Parallab has more heavy clients than Rochester, and comparable number of repeated clients to Rochester, Rochester's hit ratio is better. This can be explained by looking at the documents each server serves to its clients. Figure 7 shows the cumulative percentage of requests for a server's documents. We see that Rochester has significantly more popular documents than Parallab. For example, the 10 most popular Rochester's documents amount for 30% of the total Rochester's requests, while the 10 most popular Parallab's documents amount only for 10% of Parallab's requests. Thus, prefetching the 10 most popular Rochester's document is going to result in higher hit ratio than prefetching the 10 most popular Parallab's documents.

From the above discussion it is clear that the performance of prefetching depends on several factors. The most important ones seem to be the client base of a server, and the popularity of the documents a server provides. Frequent clients that access lots of documents form a very good basis for successful prefetching.

Although prefetching reduces the number of requests made to a web server, it may also increase traffic, since the prefetched documents may not be needed by the client that prefetched them. Figure 4 plots the traffic increase as a function of the `TOP-10` for all servers simulated. We see that the traffic increase is small for almost all servers for low (< 500) value of `TOP-10`. For example, prefetching up to 500 documents results in less than 12%, traffic increase for any server. Actually, the traffic increase for Parallab is only 5%.

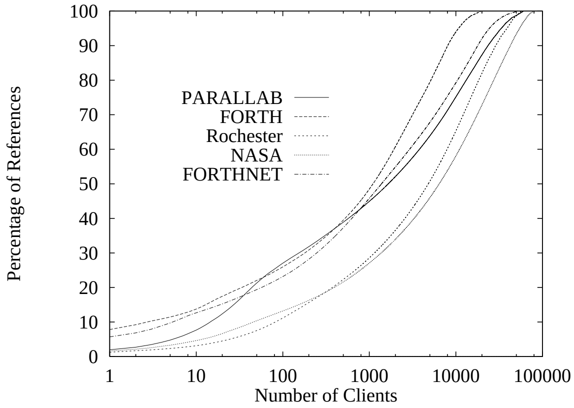

Figure 6: Cumulative percentage of requests as a function of the number of clients that make these requests.

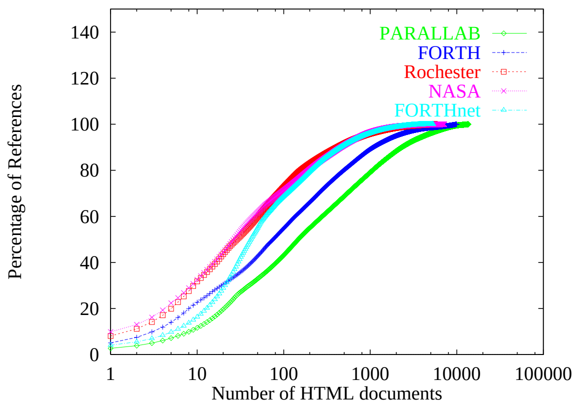

Figure 7: Cumulative percentage of requests as a function of the documents requested. Top-10 documents are *very* popular on each server, but the degree of their popularity depends on the server.

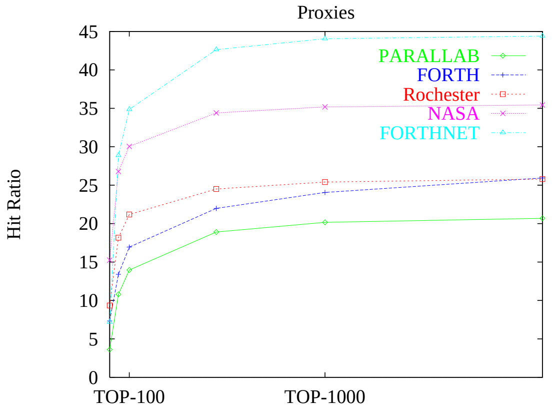

Figure 8: Successfully Prefetched documents as a function of the size of the `TOP-10`.

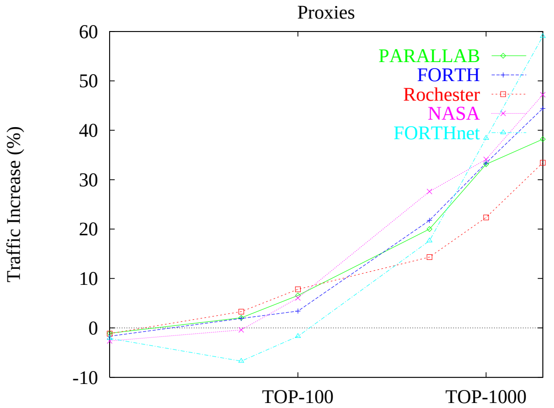

Figure 9: Traffic Increase as a function of the size of the `TOP-10`.

3.3 The effect of Proxies

In figure 3 we notice that, with the exception of FORTHnet, the hit ratio of Top-10 prefetching is between 3% and 12% which is rather low. Although it could be increased by making a more aggressive prefetching (e.g. by increasing `ACCESS_THRESHOLD`), aggressiveness will significantly increase the traffic. Recall, that the essence of prefetching is in keeping a good balance between high hit ratio and low network traffic increase. Thus, we should find other ways to improve the performance of prefetching.

One way to improve the performance of prefetching is through the use of proxies. Proxies are being extensively used for caching and firewall purposes by intervening all requests from a domain [15]. We advocate that prefetching can benefit from the use of proxies. In the current traces, several clients even from the same domain make distinct requests to a specific server. Thus, each server ends up with lots of clients, few of which qualify for prefetching. If, however, all these clients access the server through a proxy, the proxy would aggregate all the client's requests and qualify for prefetching as a repeated and heavy client. Thus, the proxy would prefetch documents that could be used to reduce the latency of *any* of its clients, and thus improve performance. For example, if a document is prefetched on behalf of one client in a proxy's cache, the document may be served locally to all other clients that use the same proxy. Thus a client will be able to make use of a document that was prefetched on behalf of another client.

To show the benefits of proxying in prefetching, we use trace-driven simulation, and introduce artificial proxies that gather client requests, and distribute the benefits of prefetching onto a larger number of

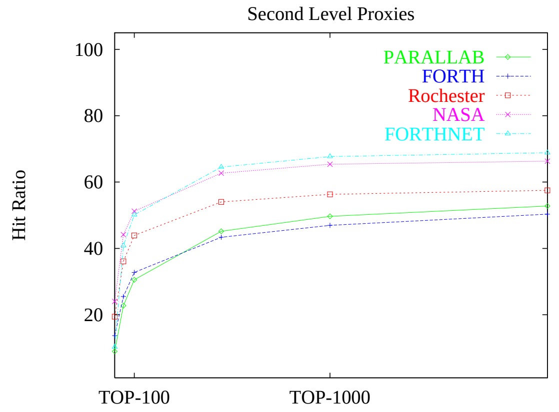

Figure 10: Successfully Prefetched documents as a function of the size of the `TOP-10`.

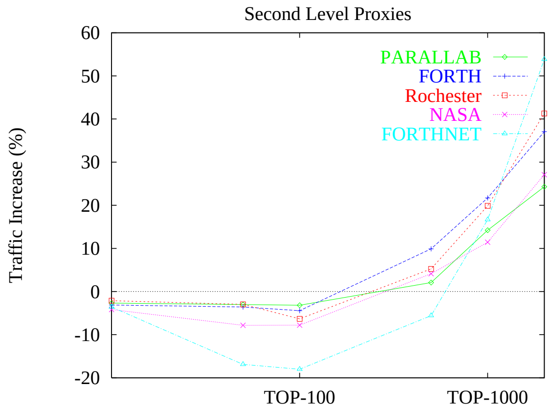

Figure 11: Traffic Increase as a function of the size of the `TOP-10`.

clients. Artificial proxies are generated by grouping clients into larger groups, and considering the whole group as a proxy for all the group's clients. Although several grouping algorithms can be designed, we use a straightforward one, which is very close to the proxying schemes used in practice. The grouping algorithm is as follows:

> A request coming from a client will be considered as coming from a proxy that has the same name as the client, with the first part of the client's name striped off.

For example, all requests coming from clients `saronis.ics.forth.gr`, `mykonos.ics.forth.gr`, and `pandora.ics.forth.gr`, will be considered as requests all coming from proxy `ics.forth.gr`. As another example, all requests coming from `vein.cs.rochester.edu`, and `athena.cs.rochester.edu` are grouped into requests coming from proxy `cs.rochester.edu`. That is, all requests that originate from any computer of the computer science department of the University of Rochester appears as coming from a single computer from that department, which is what most reasonable proxying schemes do.

Figure 8 plots the hit ratio (for proxies) due to prefetching as a function of the `TOP-10`. We see that the hit ratio of prefetching using proxy servers has doubled or even tripled compared to figure 3. For example, the hit ratio of prefetched documents from FORTHnet is close to 45%, for Parallab 18%, and for the other server's in between. Fortunately, this increase in hit rate comes at almost no increase in network traffic as figure 9 suggests. For low (< 500) `TOP-10` values, the traffic increase is less than 20%, and sometimes there is even a traffic decrease! The reason for the observed traffic decrease is simple: A prefetched document that will be used by *two* clients of the same proxy, results in traffic *decrease*, since the document is fetched into the proxy only *once*, thus saving the second request that the second client would make if there were no proxy.

We should note however, that for aggressive prefetching the network traffic increases as high as 60%, which starts to get significant. Fortunately, when `TOP-10` is less than 500, the hit ratio is almost the same with the cases for higher values of `TOP-10`, and the traffic increase is down to at most 20%, which seems reasonable. Interestingly enough, this observation holds for figures 3 and 4 where no proxies are used: increasing `TOP-10` more than 500 does not noticeably increase hit rate.

3.4 Second-level Proxies

To carry the idea of proxying one step further in this section we study two-level proxies: First-level proxies aggregate requests from user-level clients, while second-level proxies aggregate requests of first-level proxies. The algorithm we use to group the first-level proxies into second-level proxies is as previously:

> Second level proxies are found by stripping off the first two parts of a client's name.

For example, all requests coming from clients `saronis.ics.forth.gr`, and `athos.iesl.forth.gr`, will appear as coming from second-level proxy `forth.gr`. Similarly requests from `vein.cs.rochester.edu`, and `uhura.cc.rochester.edu` will appear as coming from second-level proxy `rochester.edu`. Effectively, first-level proxies aggregate requests from within a department while second-level proxies aggregate

How often should we calculate the Top-10?

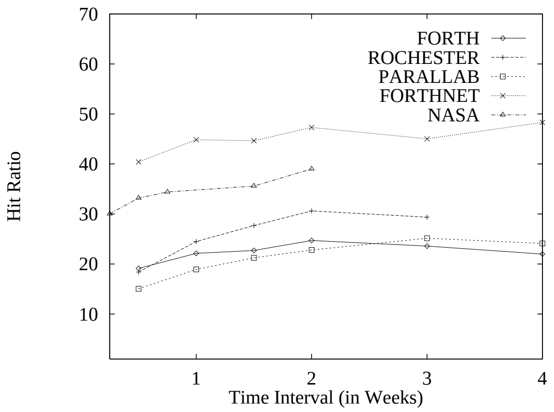

Figure 12: Hit Ratio.

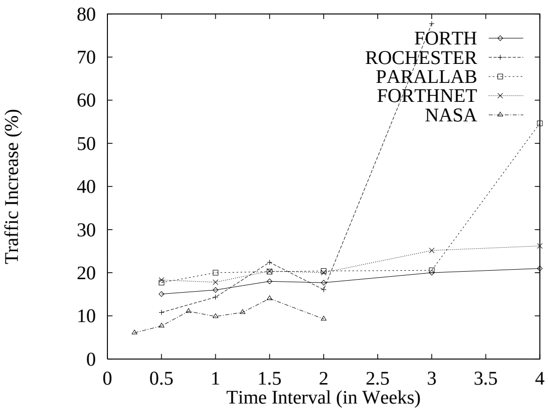

Figure 13: Traffic Increase.

requests from within an institution. In the specific example of the University of Rochester, our method assigns a first-level proxy at each department, and a second-level proxy for the whole University.

Figure 10 plots the hit ratio (for second-level proxies) as a function of the `TOP-10`. We see an even higher performance improvement, compared to figure 8. FORTHnet reaches a hit ratio of more than 60%, while even the server with the worst performance (Parallab) reaches a hit ratio close to 45%. Even better, this performance improvement is usually accompanied by a noticeable traffic decrease, as figure 10 suggests, since a document prefetched on behalf of one client will be used by lots of other clients as well. Although for very aggressive prefetching, traffic increase may go as high as 60%, prefetching up to 500 documents results in good performance and unoticeable traffic increase.

3.5 Frequency of Top-10 Release

In this section, we investigate how often a new Top-10 should be released. The Top-10 music charts have been released every week for several decades now, without any significant problems. It would be interesting to see if the same one-week time interval should apply to the calculation of the Top-10 of each Web server as well.

Intuitively we believe that the time interval for the Top-10 calculation should neither be too large, nor too small. For example, if a new Top-10 is released every several months, then it may be out of date, and all clients that prefetch it, may not use it. If, on the other hand, a new Top-10 is released every few minutes, then it will probably not be credible, and would imply a significant overhead for clients that would prefetch probably the same Top-10 every few minutes.

To find what values of the time interval would be appropriate, we conducted a trace-driven simulation, where we vary the time interval from half a week up to a month, and plotted the performance results in figures 12 and 13 (`TOP-10` is fixed at 500, with the exception of NASA where it as fixed at 100). We see that for most servers, Hit Ratio improves slowly with the time interval and then declines. [1] Interestingly enough, we see that for all servers the best interval seems to be between one and two weeks. Thus, every one to two weeks, a new Top-10 should be released.

Figure 13 shows the traffic increase as a function of the time interval. We see that traffic increases slowly with the time interval, and sometimes it fluctuates around a value. For Parallab and Rochester, traffic increases sharply after two and three weeks, respectively. The reason is that after that time interval, lots of clients start to qualify for prefetching, and lots of prefetching operations start to clients that do not really need the prefetched data. Fortunately, for time intervals less or equal to 2 weeks, all servers' traffic increase is lower then 20%. Summarizing, the best balance between hit ratio and traffic

[1] The interested reader will notice that the NASA and Rochester curves stop in 2 and 3 weeks respectively. The reason is that in order to conduct a meaningful prefetching experiment, we need at least two time intervals: one interval to find the Top-10, and a second interval, to give the Top-10 to clients and measure their hit ratio. Since we had only one month's traces for NASA, the highest time interval we could simulate was 2 weeks.

FORTH

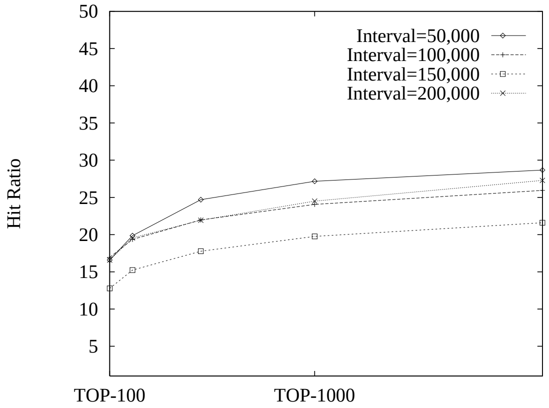

Figure 14: Hit Ratio as a function of the size of the `TOP-10`.

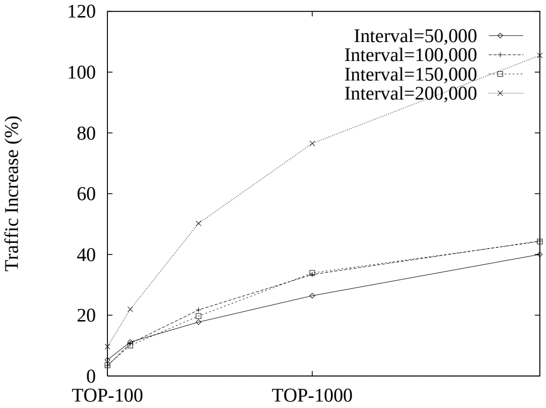

Figure 15: Traffic Increase as a function of the size of the `TOP-10`.

Rochester

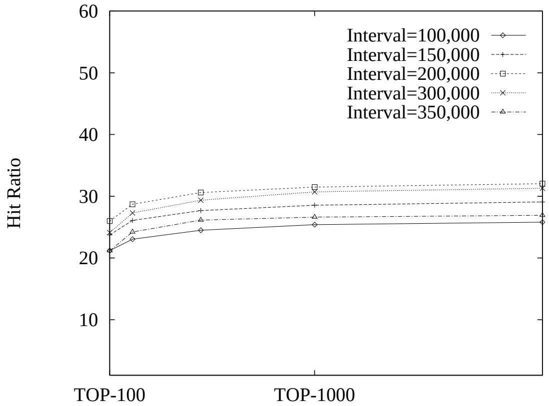

Figure 16: Hit Ratio as a function of the size of the `TOP-10`.

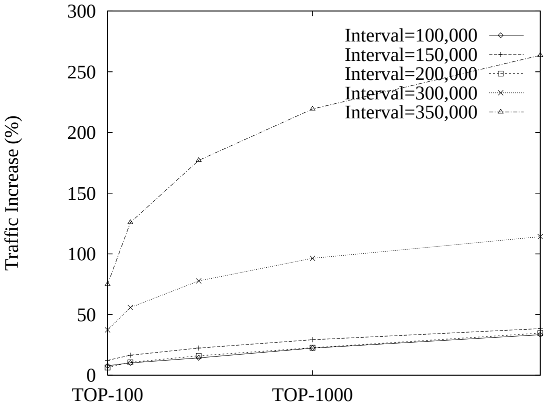

Figure 17: Traffic Increase as a function of the size of the `TOP-10`.

increase seems to be achieved when a new Top-10 is released every couple of weeks.

3.6 Changing the Time Interval

In this set of experiments we investigate further the effect that the size of the time interval will have on the performance of Top-10 prefetching. Recall that a client has to make a certain number of references within the time interval before it is enabled to start prefetching from a server. Intuitively, we believe that a very small time interval will result in low hit ratio, since few clients will make enough references to qualify for prefetching. On the other hand, a very long interval will imply that more clients would make enough references to qualify for prefetching, thus increasing hit ratio and traffic as well. However, a very long time interval may result in dis-proportionate increase in traffic, since the documents that are popular at the beginning of the interval may not be popular at the end of it.

To investigate the influence of interval size, we run the simulations again for intervals ranging from 50,000 accesses to 350,000 accesses, and plotted the results for each server in figure 14 to 23. For each server we plot both the hit ratio and the traffic increase. We see that different servers achieve the best hit rates for different time intervals. For example, FORTH achieves best hit rate for time interval of 50,000 references, while FORTHnet achieves best hit rate for 350,000 references. The other servers achieve their best performance for time intervals of 200,000 references and larger. However, we should

Parallab

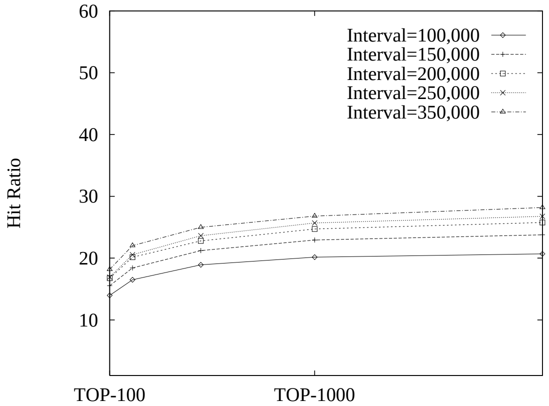

Figure 18: Hit Ratio as a function of the size of the `TOP-10`.

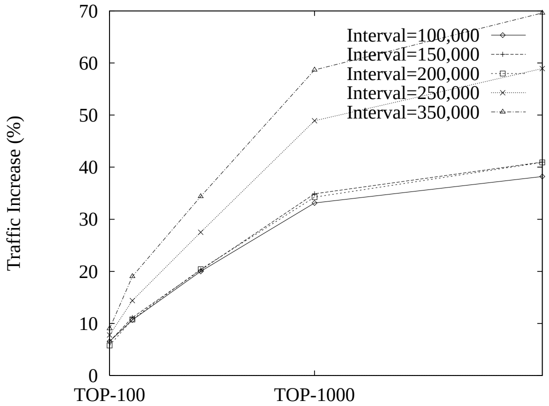

Figure 19: Traffic Increase as a function of the size of the `TOP-10`.

FORTHnet

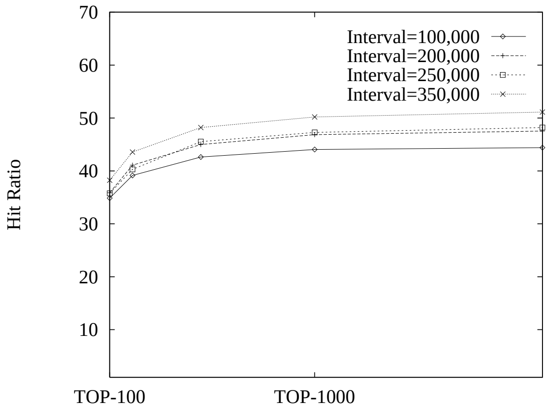

Figure 20: Hit Ratio as a function of the size of the `TOP-10`.

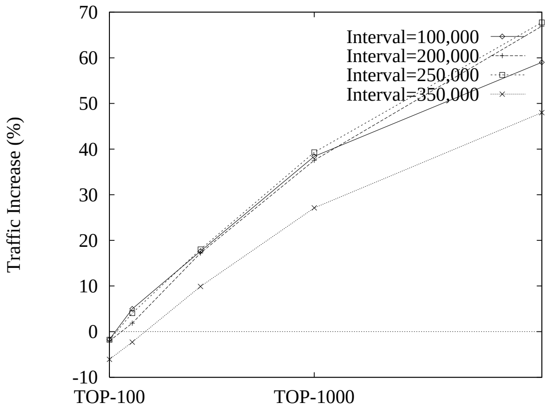

Figure 21: Traffic Increase as a function of the size of the `TOP-10`.

NASA

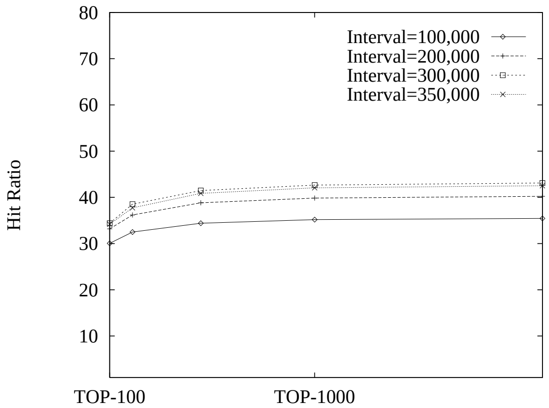

Figure 22: Hit Ratio as a function of the size of the `TOP-10`.

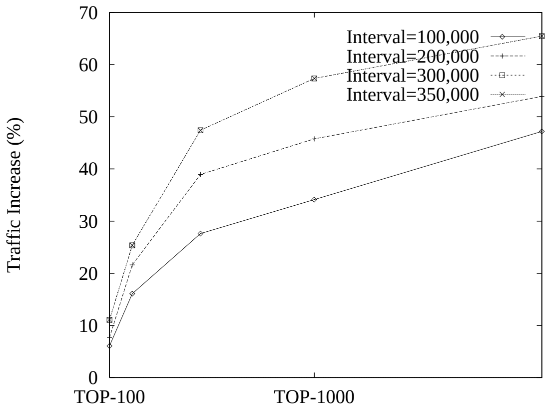

Figure 23: Traffic Increase as a function of the size of the `TOP-10`.

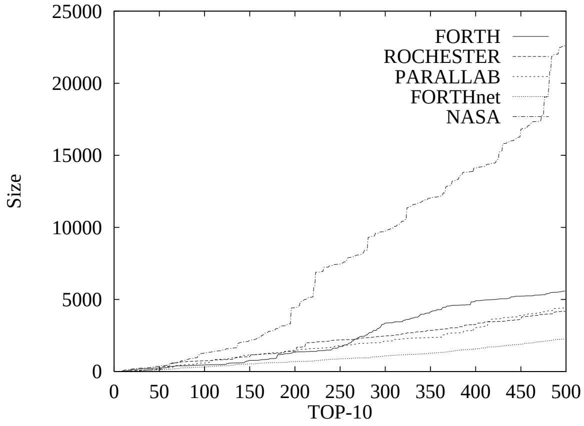

Figure 24: Size of the most popular documents.

closely observe the large hit ratio may imply significant traffic increase as well. In all figures we see that for large values of the time interval and the `TOP-10`, the traffic increase may be unacceptably high. Fortunately, when `TOP-10` is less than 500, the traffic increase is always low, while the hit ratio is close to the hit ratio achieved for much higher values of `TOP-10`. Thus, a value of `TOP-10` equal to 500 seems a reasonable choice in all cases. The only exception to the rule seems to be NASA, that seems to achieve a good balance between hit ratio and traffic increase for values of `TOP-10` around 100. The reason for this traffic increase lies in the *size* of the documents NASA provides to its clients: popular NASA documents are much bigger than popular documents of other servers. Figure 24 plots the size of the most popular files of all servers. We can easily see that NASA serves the largest files. For example, the 500 most popular documents of NASA are 22 MBytes long, while the 500 most popular documents of FORTHnet are only 2.2 Mbytes long. The reason is that NASA provides lots of large images that are very popular among many people. Thus, prefetching lots of documents from NASA may result in high traffic increase. Moreover unsuccessfully prefetched documents from NASA result in much higher traffic than unsuccessfully prefetched documents from any other server.

4 Previous Work

The area of web prefetching is rather new and is currently being actively explored. Padmanabhan [13] suggested a server-initiated prefetching algorithm. He observed that web document requests have *interdependencies*, that is, if a document D_i is requested from a web server by some client, then probably document D_j will also be requested within a small time interval T_w, by the same client. Each web server keeps this dependency information in the form of a graph. The nodes of the graph are documents; an edge between documents D_i and D_j, represents how probable is to access document D_j after document D_i has been accessed. When a client requests document D_i, the server along with document D_i, sends all the documents D_j, that are likely to be accessed next. Alternatively, the server along with document D_i sends only the names of the documents D_j that are likely to be accessed, leaving the initiative for prefetching to the client. Padmanabhan validated his approach using trace-driven simulations that gave very encouraging results: e.g. a 36% reduction in network latency can be achieved at the cost of 40% increase in network traffic.

Bestavros has also proposed a similar approach for server-initiated prefetching [4, 3]. For all pairs of documents D_i and D_j, Bestavros calculates the probability $p[i, j]$ with which document D_j will be requested within time interval T_w after document D_i is requested. Based on those probabilities, a server can *advice* clients on which documents to prefetch. Bestavros conducted trace-driven simulations which provided very encouraging results. For example, his approach using 10% extra bandwidth only, can result in 23% reduction in document miss rate.

Contrary to the previously proposed sophisticated prefetching algorithms, our Top-10 approach uses a simple and easy-to-calculate metric. Most web servers routinely calculate their most popular documents, among other statistics. Thus, no extra work is needed on behalf of the server in order to calculate which documents should be prefetched. Interestingly enough, Top-10 achieves similar (and sometimes better) results than other prefetching heuristics. For example, Top-10 has shown to achieve close to 60% hit rate, at only 10% traffic increase (see figure 10), because `TOP-10` capitalizes on two web invariants:

- Popular documents are *very* popular,
- The use of proxies increases the hit ratio and reduces network traffic.

Gwertzman [10, 9] proposed a geographical push-cashing approach to reduce a web server's load: when the load of a web server exceeds some limit, the server replicates (pushes) the most popular of its documents to other cooperating servers that have reduced load, so that clients will be able to make future requests for these documents from the other servers. Push-cashing replicates only popular documents (much like Top-10) in some server, while Top-10 replicates them in some proxy close to the client. In push-cashing, the client still needs to request the replicated documents from some, usually non-local server, while in Top-10 the clients request the documents from a local proxy. To make matters worse, in push-cashing clients need to know which server to ask the documents from, that may involve a request to the original server, which adds even more to the client's latency. Thus, although push-cashing may off-load a busy server, by replicating its most popular documents, it still requires the client to make one or more requests to non-local servers in order to find the replicated data. On the contrary, Top-10 replicates popular documents only to local proxies, so that clients always make local accesses to replicated data.

Wachsberg *et al.*[17] propose the use of prefetching as a way to improve performance of web browsing over low-bandwidth links. They propose a client-based approach where each proxy will keep a list of documents needed by its clients, and it will decide which of them to prefetch. However, they have reported no performance results yet.

The benefits of prefetching are going to be investigated within the LowLat [16] project. In the LowLat approach, a *preloader* will be responsible for communicating with the web servers and prefetching documents from them. The prefetching process will take into account several factors including bandwidth, cache load and space, server load, etc. Unfortunately, the prefetching policies and their associated benefits have not been reported as of the time of this writing (summer 1996).

Recently, there has been considerable work on web Caching, that is, caching of popular documents close to clients [1, 5, 7, 6, 8, 11, 12, 14]. All this work aims at reducing both network traffic and server load by keeping data close to clients that re-use them. Most web servers and proxies today support some form of caching. Our work complements the research in caching, since all benefits of prefetching are *in addition* to those of caching. While caching attempts to provide fast access to a document the *second* time it of accessed, prefetching provides fast access to a document, the *first* time it is accessed. Moreover, the already existing infrastructure for caching (proxies, etc.) can also be exploited for prefetching.

Summarizing, we believe that out Top-10 approach is an easy to calculate algorithm that achieves effective prefetching of documents in the web.

5 Discussion

In this paper we present a systematic approach towards the reduction of the web latency experienced by web clients, by *prefetching* documents, before they are actually requested by the users.

Prefetching has not been employed in the Web so far mainly for several reasons: (i) a prefetching robot can easily get out of control and start prefetching everything that it is out there, (ii) prefetching may be ineffective, since nobody knows what a client will want to access, (iii) proxies may delay clients, and finally (iv) prefetching over high-speed interconnection networks may result in minor performance improvements.

We believe our prefetching approach addresses all previous concerns about prefetching for the following reasons:

- The Top-10 approach to prefetching uses several well defined thresholds to make sure that only a small number of useful documents is prefetched. The prefetched documents do not increase the total traffic by more than 10%-20%. Our prefetching approach sometimes even *reduces* the total network traffic by aggregating several clients requests. Thus, it can not get out of control and lead to traffic chaos.

- We prefetch only the proven *popular* documents that most clients have accessed, and that future clients will probably want to access. Thus, the risk of bringing useless documents is minimized. Essentially, by prefetching only popular documents the risk of guessing the future is significantly reduced.

- Although sometimes a user "feels" that it is faster to retrieve a document directly from a server instead going through a proxy, this is because most of the other users think that they should go through a proxy and avoid putting unnecessary load to a server. If everybody starts accessing the servers without intervening proxies, most servers will collapse, and most interconnection lines close to the servers will saturate. The only way to avoid the collapse is to use proxies somewhere in the traffic route from a client to a server. One could argue that aggregating several clients through a proxy will slow down all clients and eventually saturate the proxy. Although this may be true to some extent, it can not easily happen. Proxies are usually powerful computers capable of handling hundred of requests per second. If each interactive user that browses the Web makes one request every few seconds, then the proxy will be able to handle up to a few thousand users before being unable to cope with more requests. Realistically, most departments do not have that many users to generate the load needed to saturate a proxy.

- Although prefetching over a high-speed interconnection may result in minor performance improvements, certainly it does not hurt the clients, and it may benefit the servers by taking some of the load off the server and putting it on the proxies.

In addition to addressing previous concerns, we believe that prefetching in general, and Top-10 in particular has several advantages:

- Prefetching can be used during off-peak periods to downlod useful documents at low transfer costs. By transferring documents during low-rate periods, prefetching pays for itself, and may even result in profit.

- Prefetching reduces client latency by spreading the load from busy servers to idle clients/proxies. Thus, it improves user turnaround time and user productivity.

- Prefetching can also be used to organize up-to-date digital repositories of related documents. For example, a prefetch agent may download all documents related to a specific research topic, update them regularly and make them available to local users.

Although several people have concerns about prefetching, we believe that the Top-10 approach has been specifically designed to address these concerns, and bring out the benefits of prefetching. Top-10 takes the risks out of prefetching by doing it in a controlled way that results in significant performance improvements with only a minor traffic increase.

6 Conclusions

In this paper we present a Top-10 approach for prefetching World Wide Web documents. Top-10 prefetches only the most popular documents (that is where the name comes from) and only to clients that will be able to use them. We use trace-driven simulations of server traces to evaluate the costs and benefits of our approach. Based on our experimental observations we conclude:

- The Top-10 approach to prefetching may result in significant performance improvements. Our experimental results suggest that Top-10 manages to prefetch (up to) 60% of future requests, with little (less than 20%) corresponding increase in traffic (see fig. 10).

- Top-10 has shown to be robust over a wide variety of parameters and server loads. We have used traces from 5 different servers both in Europe and the States. Top-10 always managed to result in good performance for all traces without significant traffic increase.

- Prefetching the most popular documents is a simple but effective prefetch heuristic. It requires very little effort to be computed on the server side, while it provides performance comparable to (if not better than) previously proposed sophisticated prefetching heuristics.

References

[1] M. Abrams, C.R.Standridge, G. Abdulla, S. Williams, and E.A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of the Fourth International WWW Conference*, 1995.

[2] M.F. Arlitt and C.L. Williamson. Web Server Workload Characterization: The search for Invariants. In *Proc. of the 1996 ACM SIGMETRICS Conference*, May 1996.

[3] Azer Bestavros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems. In *Proceedings of ICDE'96: The 1996 International Conference on Data Engineering*, March 1996.

[4] Azer Bestavros. Using speculation to reduce server load and service time on the WWW. In *roceedings of CIKM'95: The Fourth ACM International Conference on Information and Knowledge Management*, November 1995.

[5] Azer Bestavros. Demand-based document dissemination to reduce traffic and balance load in distributed information systems. In *Proceedings of the 1995 Seventh IEEE Symposium on Parallel and Distributed Processing*, October 1995.

[6] J.-C. Bolot and P. Hoschka. Performance Engineering of the World-Wide Web. In *Proceedings of the Fifth International WWW Conference*, 1996. Paris, France.

[7] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A Hierarchical Internet Object Cache. Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, March 1995.

[8] S. Glassman. A Caching Relay for the World Wide Web. In *Proceedings of the First International WWW Conference*, 1994.

[9] J. Gwertzman. Autonomous Replication in Wide-Area Networks. Technical Report 17-95, Harvard University, 1995.

[10] J. Gwertzman and M. Seltzer. The Case for Geographical Pushcaching. In *Proceedings of the 1995 Workshop on Hot Operating Systems*, 1995.

[11] Radhika Malpani, Jacob Lorch, and David Berge. Making World Wide Web Caching Servers cooperate. In *Proceedings of the Fourth International WWW Conference*, 1995.

[12] E.P. Markatos. Main Memory Caching of Web Documents. In *Proceedings of the Fifth International WWW Conference*, 1996. Paris, France.

[13] V.N. Padmanabhan. Improving Word Wide Web Latency. Technical Report 95-875, University of California at Berkeley/CSD, 1995.

[14] J.E. Pitkow and M. Recker. A Simple, Yet Robust Caching Algorithm Based on Dynamic Access Patterns. In *Proceedings of the Second International WWW Conference*, 1994.

[15] Neil G. Smith. The U.K. National Web Cache: A state of the Art Report. *Web Journal*, 1(3), Summer 1996.

[16] Joe Touch. About the LowLat Project, 1996. http://www.isi.edu/lowlat/about-ll.html.

[17] Stuart Wachsberg, Thomas Kunz, and Johnny Wong. Fast World-Wide Web Browsing Over Low-Bandwidth Links, 1996. http://ccnga.uwaterloo.ca/ sbwachsb/paper.html.