2000 IEEE International Symposium on Performance Analysis of Systems and Software.
April, 2000. Austin, Texas.

WEB LATENCY REDUCTION VIA CLIENT-SIDE PREFETCHING

Avinoam N. Eden *Brian W. Joh* *Trevor Mudge*

Electrical Engineering and Computer Science Department
The University of Michigan, Ann Arbor
ane@engin.umich.edu, tnm@eecs.umich.edu, bwj@umich.edu

ABSTRACT

The rapid growth of the WWW has inspired numerous techniques to reduce web latency. While some of these techniques have not been implemented because they either increase network traffic or require cooperation between tiers, recent studies cast a shadow on techniques already in use (e.g. proxy caching) as a result of the increasingly dynamic aspects of the WWW. In particular, the proliferation of dynamically generated web pages (i.e. cgi, ASP), which are either linked to a database, or extract information from cookies, reduces the effectiveness of cacheing techniques. Most techniques attempt to improve on part of the overall latency, and often neglect to address the internal latency, which can be a serious bottleneck in heterogeneous environments.

We propose a client-side prefetching mechanism, where the decision of what to prefetch is left to the user. We found it has the potential of reducing latency by up to 81% in a homogeneous environment and 63% in a heterogeneous environment. In data taken on the client, the technique depicted the potential to decrease latency by three-fold. Client-side prefetching does not increase network traffic, it attempt to improve on all parts of latency, and it can be implemented on the client side, without the cooperation of any other tier. Moreover, it can work seamlessly with any other latency reduction technique. We advocate the inclusion of a suitable mechanism in future web browsers to support client-side prefetching.

I. INTRODUCTION

The rapid growth of the WWW inspired numerous techniques to reduce web latency. In this section, we first describe the possible WWW environments, which those techniques might work within. We than proceed to describe the different techniques in the context of those environments and highlight the advantages and disadvantages. Finally we go on to describe client-side prefetching, which we believe can overcome most of the disadvantages of earlier techniques.

A Environment

Figure 1 depicts a classic WWW architecture. A web browser residing on the client machine is used to request web pages from the server employing the HTTP protocol. The request is channeled through a proxy server, which might be used as a firewall, for caching, prefetching or/and filtering requests. If the proxy server cannot service the request, it forwards it to the content provider, which uses the web server to service the request. Depending on the request, the server might do one of two things: If the request is static object (i.e. HTML, JPEG, GIF, Applet, etc.), the server transfers it back to the proxy server, which delivers it to the client. On the other hand if the request is for a dynamic object (i.e. ASP, CGI, etc.), the server first constructs it, and only then sends it to the proxy. The construction of the object might first require querying a database server, then constructing the object according to the result set. In another example, a dynamic object might be constructed by first requesting a cookie from the client and then constructing the page. As can be seen in figure 1, we divided the latency into three components: internal, external, and object creation latencies. The internal latency consist of the time it takes the proxy server to transfer the object to the client. The external latency is the time it takes the server to respond and transfer the object to the proxy server. Object creation latency applies only to objects, which are created dynamically. It includes the time it takes to the server to construct the object, either by querying a database, a cookie, or simply processing a script.

Figure 2 exhibits the three most common configurations in use on the WWW. The first describes as the homogeneous three-tier architecture (2a), where the proxy server is connected via a fast connection to both the server and the client. This kind of architecture is widely used in corporate settings, where the middle tier (proxy) might be used to provide security (a firewall), reduce

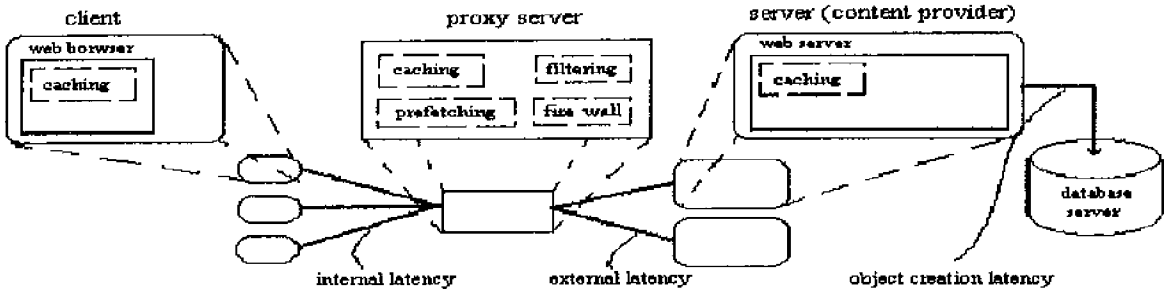

Figure 1 - General WWW Architecture

latency (proxy caching), or monitor and filter content. In the heterogeneous three-tier (2b) architecture the proxy is connected to the server via fast connection while connected to the client via a slow connection (i.e. modem). This architecture is used by most Internet home-users, where the middle tier is usually a modem pool. While this is its main role, it can also be used for caching, prefetching, etc. In the third architecture (2c), the proxy is not present and the client is connected to the server via fast connection. A fourth architecture, where the client is connected to the server via a modem falls under a category of a Bulletin Board Service (BBS) and will not be described here.

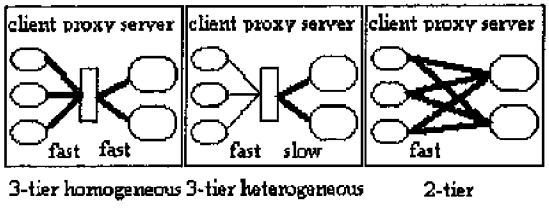

Figure 2 - General WWW Architectures

B Latency Reduction Techniques

The following section provides a brief overview of latency reduction techniques.

B.1 Cacheing

Cacheing in the WWW can be divided into three categories: proxy, clients, and server side caching. In proxy caching a computer (the proxy server) serves as a cache of objects for a set of WWW clients. Ideally, the proxy server is physically located closer to the clients and has fewer clients to serve than the content provider (the WWW server). Clearly, if the proxy server has an object cached, it will be able to service a request for this document faster then the content provider. Due to the proximity to the client and smaller number of clients the proxy has to service. Early studies suggested a significant latency reduction if proxy caching is employed.

Gribble and Brewer showed that proxy cache hit ratios can approach 60% for a pool of 8000 clients [8]. It was shown that the hit ratio is a function of the number of clients the proxy server services. Therefore, if the proxy services more than 8000 clients, the hit ratio is likely to increase. A later study by Kroger et al. accounted not only the cache's hit ratio, but also the reduction in the client's perceived latency [10]. In this scenario it was shown that the latency reduction could only reach 26% at best. Another setback to proxy caching was shown by Caceres et al: by looking at more details such the presence of cookies in HTTP requests they observed a hit ratio of only 35.2% [7]. Furthermore, by considering aborted connection they showed that the network traffic between the proxy server and the service provider actually increased. None of the studies above took in consideration client-side cacheing. It is probable that some of the objects, cached and serviced by the proxy server, are already cached on the client's side. For example, a heavy use of the BACK button on the web browser might register a request at the proxy server (depending on the browser configuration), but the objects requested are already in the client's cache and therefore can be serviced by that cache. We suspect that not considering client-side caching while examining proxy caching, results in an optimistic evaluation of proxy caching, and we advocate further studies to consider client-side caching.

In spite of the pessimistic results from recent studies, proxy cacheing is employed. We speculate that as the number of dynamically web pages constructed increases, the benefit of employing a cache on a proxy server will diminish.

In client side cacheing the objects (i.e. HTML, GIF and JPEG files etc.) are stored for a limited period of time on the local hard drive of the client. Only request made by the client, are cached, therefore, the benefit of sharing

cached object brought by other clients does not apply. We speculate that a large portion of the benefit from client side caching is limited to the use of the BACK button in the web browser (i.e. the client visited the page in the current session and return to it using the BACK button). Client side caching is employed in most web browsers on the market and cacheing is activated by default.

Server side cacheing caches the content of web pages, which are generated dynamically. Downloading dynamically generated web pages (i.e. ASP and CGI files) can incur large object creation latency. For example the construction of a page cannot be completed before the query is done for if querying data in a database generates a web page, the construction of the page cannot be completed before the query is. If the query takes long to complete the user will notice a large latency when downloading this page. Since temporal locality exists in access pattern to the web server (i.e. if a page is accessed by one client, it is likely to be accessed by other clients in the near future), caching those pages on the server is beneficial. With server-side caching only few clients will incur the latency associated with generating the dynamic web page. This technique was successfully used by IBM in the 1996 Atlanta Olympic games' official web site [2].

B.2 Prefetching

Prefetching on a proxy server has also been explored. The knowledge whether to fetch a page or not can come from the prefetching side or the server side. Since the server has more information regarding request patterns, it is beneficial to use server hints to determine what to prefetch. However, there is some overhead associated with transferring the prefetching hints from the server side to the prefetching proxy.

Prefetching from the proxy side is a well studies concept [10], [11], [1]. While a reduction latency of up to 50% has been seen, it only came at the expense of significantly increasing the amount of traffic on the network. It is hard to determine the effect of a clogged network on latency; clearly this will adversely affect it.

While we are not aware of any literature on server side prefetching, it should be possible to reduce latency by prefetching a dynamic web page on the server. If there is knowledge that a web page will be requested in the near future, the server can prefetch this page so when the request is made the web page is already constructed and the client does not have to incur the latency associated with constructing it. However, the only time when this scheme will be cost effective is when a rapid change of data, on which the page construction depends, is present. Without this, prefetching on the server side will not yield better results than a simple caching scheme on the server and might adversely effect the web server performance by taking valuable CPU time.

B.3 Other schemes

In most cases caching and prefetching schemes do not attempt to cache/prefetch a dynamic web page. The assumption is that dynamically constructed web page might constantly change, and therefore any form of caching and prefetching is prohibited. Delta encoding [5][6] reduces this problem by only sending the portion of the binary file, which changed since the last version stored on the proxy side. HTML Pre-Processing (HPP) [4] is an HTML extension, which distinguish a static and a dynamic portion. While the static portion can be cached, the dynamic portion is generated for each request. Since a large portion of dynamic web pages is static, such a scheme can alleviate the perceived latency.

C Client-Side Prefetching

Network bandwidth is the biggest limitation to the WWW expansion. With unlimited bandwidth, video conferencing, Internet broadcasting, and Internet phone would be widely in use. To alleviate the bandwidth limitation, methods to reduce the clients perceived latency were revised and borrowed from classic I/O. The first ones were proxy caching and client side caching, which are widely implemented.

However, the rise in dynamic and specialized web content has limited these caching methods, because a growing number of web pages cannot be cached. Moreover there is an extra latency associated with generating those objects. Confronting those problems are methods such as HPP and delta encoding. Unfortunately they were never widely implemented because they require cooperation between the proxy and the server. It is unlikely that these methods will ever be widely accepted unless they become part of the HTTP protocol or some other standard. The only widely implemented method to specifically reduce the object creation latency, is server-side caching. Proxy-side prefetching was shown to yield good reduction in the client's perceived latency, but only at the expense of large increase in network traffic. The lesson learned from PUSH technology will probably prevent proxy prefetching from being implemented until the increase in network traffic can be tamed. We observe that for a latency reduction technique to be implemented it must not yield increase in network traffic and it must be

an independent component: one that does not need the cooperation of another tier.

Server-side caching is trying to reduce only the object creation latency. Proxy-side caching and prefetching, on the other hand, are aimed at reducing external latency. A secondary effect to reducing external latency, is the reduction of object creation latency. Attempting to reduce internal latency will result in targeting external and object creation latencies as well. Targeting internal latency is critical in heterogeneous environments, because the slow connection between the client and the proxy-server is often the bottleneck. Unfortunately the only scheme that tries to reduce the internal latency is client-side caching, and the limits of this scheme have been discussed above.

In this paper we consider client-side prefetching, where the user is in control of the prefetch process. By prefetching from the client side we target all portions of the latency; by having the user decide what and when to prefetch, we eliminate any extra network traffic usually created by prefetching. Such a prefetching mechanism can be implemented on the client machine with out any cooperating from the proxy or the server. As discussed above, the ease of implementation, and the lack of extra network traffic, makes such mechanism a strong candidate for wide spread implementation. Such mechanism can be implemented as part of current web browsers or as a plug-in for a browser.

The mechanism can be simple. Its main function is to provide a click to prefetch mechanism (i.e. by pressing Shift and left clicking on a hyperlink), in addition to the click for fetch mechanism present in current browsers (left click on a hyperlink). If the click and prefetch mechanism is used instead of bringing the page and displaying it, the browser brings to page and keep it internally. The user can access the prefetched page via a list similar to a favorite list. When the user chooses, a page from the prefetch list, it is displayed. Any attempt

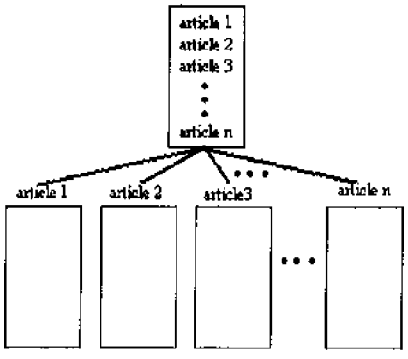

Figure 3 - Web Site's Tree Structure

to fetch a page via the click and fetch mechanism will take precedence over the prefetching mechanism. This idea can be taken further with precedence set among the prefetched pages, prefetch and display mechanism (where the pages are displayed only when they are fully fetched), and so on.

Prefetching in I/O systems has three important metrics: coverage, accuracy and timeliness. Coverage determines the fraction of web pages prefetched before requested by the client. Accuracy measures what fraction of the web pages were prefetched and actually used by the client. Timeliness measures what fraction of the web pages prefetched before being requested. Since prefetching is left to the user, we assume 100% accuracy, or in other words, every web page requested by the prefetching mechanism is used. This is in contrast to other prefetching mechanisms, where there is a tradeoff between coverage and accuracy; client side prefetching will not overload the network. The coverage and timeliness are dependent upon user training and the specific web site. All simulations assume ideal user, who, given a proper prefetching mechanism, will fully utilize it. In order to prefetch an object, the user needs to know ahead of time, that she would like to use it in a short while. As we will show, the tree like structure, used by most web sites, can be employed to advantage prefetching.

Take for example a web site that provides news. Figure 3 shows the structure of such a web site in which, the main page contains the headlines with hyperlinks to different articles. Once the end-user reads the headlines, she knows which article she would like to read. Assume that she would like to read articles 1,2 and 6. Figure 4a shows a typical web session over time for this sequence of pages. From the main page the user will request the first article (i.e. article 1) and once the page is downloaded the user will spend time reading it. The user then hits the BACK button to go back to the main page, this time the main page takes very little time to download because it is cached on the client. Next the user chooses to download the second article she is interested in (i.e. article 2). This process continues for the rest of the session with article 6. An important observation is that the user knew she would like to read articles 1,2 and 6 after reading the main page. Figure 4b shows the same web session, depicted in 4a, when the client prefetching mechanism is employed. While article 1 is being read article 2 can be prefetched. Once the user demands article 2 for viewing it already resided in the client machine and therefore the perceived latency is zero. The potential of improving performance is obvious, the question is how much opportunity for such prefetching the structure of the

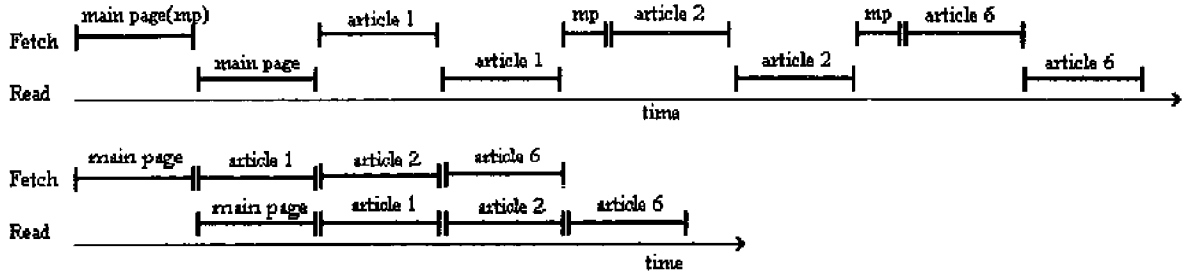

Figure 4 - Web Sessions: a) No Prefetching b) With Prefetching

web sites provides.

II. SIMULATION ENVIRONMENT

In our simulations we used traces obtained within a firewall web proxy used by Digital Equipment Corporation [10] to simulate a homogeneous environments. All HTTP requests from client within Digital to servers on the Internet made between, August 29th and September 22nd, were recorded. 24,659,182 such requests were made coming from 17,354 different clients to 140,506 distinct servers. The proxy server provided no latency reduction methods; it served as a method for crossing the corporate firewall.

A Simulation Details

As in [10] we focused on the latency seen when retrieving a web page and therefore concentrated on requests using the HTTP protocol and GET method. Failed events such as aborted connections were part of the calculation, since the user still waited for those requested until she aborted them. Since the client is initiating the prefetching, requests for cgi-bin object were included.

The Digital traces were taken in a three-tier homogeneous environment. We used the trace's external latency (e) and total latency (t) to calculate the internal latency (i = t-e). Notice that the external latency (e) in the traces encompasses both the object creation and external latency as discussed before.

In order to mimic a heterogeneous environment, we recalculate the various latencies, and the time when a request was initiated. The internal latency was obtained by dividing the object size by 56,000 bits per second (bps). The value, 56.000 bps, serves as an upper bound on data transfer through regular phone lines. The external latency was calculated by assuming a T1 connection. The overall latency was calculated by adding the internal and external latencies calculated. The time a request was initiated was adjusted by adding the time the page was used (the difference between two requests minus the total latency of the first request) to the calculated heterogeneous latency and then adding it to the time the previous request was initiated.

For both environments, the latency after prefetching was calculated by subtracting the amount of time the client had to prefetch a web page from the total time (l=t-pf). If the result turned out negative, the web page was considered to have zero latency.

The simulations were done on a web page granularity (as oppose to an object granularity). Since the traces were base on objects we used a heuristic to construct and equivalent page based trace. A web page was considered to be a HTML or cgi-bin file and all the following objects requested by the same user (GIF, JPEG files, etc.) until the next HTML or cgi-bin file was encountered.

Using the heuristics described next, the simulations stepped through the traces constructing the web pages, and determining whether the client was able to prefetch this web page. If it was able to prefetch a page, the amount of time it had to prefetch the page was calculated, and the perceived latency was calculated by subtracting the prefetch time from the latency. The overall latency for the traces is determined by adding the user's perceived latency for each web page. The total perceived latency is compared to the total latency when using no prefetching.

B Simulation Heuristics

Since the traces do not indicate whether a user is able to prefetch the next page, nor provide the data to construct the web site structure (i.e. what other web pages does the current web page have a hyperlink to), we used three heuristics. The first one, which is termed "vanilla", assumes that a user can always prefetch the next web

page. The second one, termed "same-server", assumes that a user can only prefetch a web page if the web page resides on the same server as the current page. The last heuristic assumes that a web page can only be prefetched if in a tree structure, and it is a leaf of this tree. Since the traces do not provide the information required to know weather a web page is in a tree structure, we attempted to construct it. A web page was assumed to be in a tree structure if it showed the following pattern: a, b, a, c, a, d... In this case we assumed pages b, c and d are leaves and therefore prefetchable. The tree heuristic is an under estimation since web browsers can be configured to use the client-side caching. Although pages b, c and d are leafs of page a, the pattern which will be seen on the server will be a, b, c, d, and the tree structure is therefore lost.

C Sources of Inaccuracy

We note several sources of inaccuracy for the presented simulations. Our result only shows the potential in client-side prefetching, and by no mean it shows the improvement, which will be obtained by using the client-side prefetching mechanism.

To get the true benefit from the client-side prefetching, the user must be trained, browsing a prefetch friendly web site, and an easy prefetching mechanism. No page rendering time is included and we did not consider any overlapping of the external and internal latencies.

III. RESULTS

The purpose of the simulations is to show the potential for client-side prefetching with respect to client-side and proxy-side caching for both homogeneous and heterogeneous environments. We also show the effect of client-side prefetching when used in conjunction with those methods.

A Traces Collected on the Proxy

Table 1 presents the percentage of external latency with respect to total latency for homogeneous and heterogeneous environments. The external latency constitutes about 50% in the homogeneous environment. In the heterogeneous environment, on the other hand, it constitutes only about 35% of the total latency. The external latency puts a bound on most latency reduction schemes, which address only the external latency portion. Proxy caching, and prefetching are examples for such techniques. A perfect proxy cache (hit ratio of 100%) will reduce the latency only by 50% for homogeneous environment, and 35% for a heterogeneous environment.

	Week1	Week2	Week3
Homogeneous			
Total latency *t*	0.21	0.20	0.21
External latency	0.11	0.11	0.11
Percentage of external latency e/t	0.50	0.53	0.50
Heterogeneous			
Total latency *t*	2.96	2.84	2.87
External latency	1.03	0.98	1.00
Percentage of external latency e/t	0.35	0.35	0.35

Table 1: External Latency in Homogeneous & Heterogeneous Environment

Table 2 depicts the latency reduction when using client-side prefetching in a homogeneous environment. When using no latency reduction techniques the latency is 0.21 seconds for the first week. Using client-side prefetching, the latency reduction for vanilla, same-server, and tree schemes is 81%, 76 % and 52 % respectively. An important observation is that the average length of tree prefetching is 3.2. This suggests the tree heuristic provide pessimistic results, for the reasons discussed above. Week 2 and 3 depicts similar behavior.

When using proxy caching, the latency reduces from 0.21 seconds to 0.17 seconds - a 19% improvement. Client-side caching reduces the latency to 0.16 seconds, and when using both client-side and proxy-side caching, the latency reduces to 0.16 seconds - a 24% improvement. Using client-side prefetching yields improvement of 37% for tree prefetching, and up to 75% improvement for vanilla prefetching.

Table 2 show that an advantage can be gained by using client-side prefetching in conjunction with other latency reduction methods. When using client-side prefetching in conjunction with both client-side and server-side caching the latency reduction runs from 62% for the tree scheme up to 86% reduction in latency for the

vanilla scheme.

		Week1	Week2	Week3
	Total	0.21	0.20	0.20
	Proxy caching	0.17	0.17	0.17
	Client caching	0.20	0.20	0.20
	Both caching	0.16	0.16	0.16
Client Prefetching	Vanilla	0.04	0.04	0.05
	Same server	0.05	0.06	0.05
	Tree	0.10	0.10	0.10
With Proxy Cache	Vanilla	0.03	0.03	0.04
	Same server	0.04	0.05	0.05
	Tree	0.08	0.09	0.09
With Client Cache	Vanilla	0.04	0.04	0.04
	Same server	0.05	0.06	0.05
	Tree	0.09	0.10	0.09
With Both Caches	Vanilla	0.03	0.03	0.04
	Same server	0.04	0.05	0.05
	Tree	0.08	0.08	0.08

Table 2: Latencies in Homogeneous Environment (avg. in seconds)

Table 3 depicts the simulation results in a heterogeneous environment. The improvement in latency is from 40% for the tree scheme up to 63% for vanilla scheme. The improvement shown for the homogeneous environment is larger. This is due to the limitation in viable prefetching time. Prefetching is done while the user is observing other pages. However, in a heterogeneous environment the observation time with respect to the total session time is lower than in the homogeneous environment, there is a smaller percentage of the time to fetch pages. Therefore, the improvement due to client-side prefetching is lower in percentage for the heterogeneous environment. Notice however, that the user in the heterogeneous environment ends up saving 1.78 seconds due to client-side prefetching, compare to a user in a homogeneous environment. Since the latency is much more noticeable in the heterogeneous environment, client-side prefetching might be more beneficial in a heterogeneous environment, or for large objects in the homogeneous environment.

B Results Using Data Collected on the Client

The DEC traces have a few shortcomings: To preserve anonymity sake the information provided by the DEC traces regarding the URLs, is limited. This makes it hard to evaluate the client-side prefetching technique. The DEC traces were taken on the proxy side, which make it hard to estimate the overlap between the internal and external latencies. For those reasons we collected our own traces on the client side. Code was embedded in web pages to collect, using cookies, when a page was requested, fully fetched, and left by the user. This information gave us a true picture of the latency reduction potential inherent by the client-side prefetching by a mix of users and environments on the WWW.

		Week 1	Week 2	Week 3
	Total	2.95	2.89	2.88
	Proxy caching	2.92	2.85	2.85
	Client caching	2.75	2.67	2.67
	Both caching	2.72	2.63	2.64
Client Prefetching	Vanilla	1.12	1.16	1.17
	Same server	1.12	1.17	1.18
	Tree	1.79	1.81	1.80
With Proxy Cache	Vanilla	1.10	1.15	1.16
	Same server	1.10	1.15	1.16
	Tree	1.77	1.79	1.78
With Client cache	Vanilla	1.06	1.09	1.10
	Same server	1.12	1.17	1.18
	Tree	1.68	1.68	1.66
With Both Caches	Vanilla	1.05	1.08	1.08
	Same server	1.10	1.15	1.16
	Tree	1.65	1.65	1.64

Table 3: Latencies in Heterogeneous Environment (avg. in seconds)

The traces were obtained constructing our own simple web site. The structure of the web site was kept simple and consists of on index page pointing to 15 different pictorial web pages. We collected data between

August 15th 1999 and November 15th 1999. 144,649 different users made 1,345,235 different requests for web pages.

The size of the web pages ranged from 113K to 384K. We acknowledge that the structure of the web site is tailored for the client-side prefetching technique but the usage of pictorial pages is a limiting factor because the user is observing the pages, while they are downloading. This reduces the observation time, which can be used for prefetching.

While the average latency per page was 12.53 seconds without prefetching, the usage of client-side prefetching was able to reduce it to 4.32 seconds. This is almost a three-fold decrease in latency. The average consecutive number of pages to be prefetched was 4.92 pages.

IV. SUMMARY

Using trace driven simulations we have explored the potential of a client-side prefetching mechanism. Such mechanism can be implemented on the client-side without the cooperation from other tiers; it will not increase overall network traffic. The positioning of the mechanism on the client side result in attempting to reduce all portions of latency, so in the best case it will reduce the latency to zero. Most latency reduction techniques cannot reduce internal latency, which can be a big factor of overall latency, especially in heterogeneous environments.

Our simulations shows that client-side prefetching outperforms both proxy, and client-side caching, furthermore, it can work in conjunction with other latency reduction techniques. We advocate the inclusion of such mechanism in future web browsers.

V. REFERENCES

[1] A. Bestavros and C. Cunha. *A prefetching protocol using client speculation for the WWW.* Tec. Rep. TR-95-011, Boston University, Department of Computer Science, Boston, MA 02215, April 1995.

[2] A. Iyengar and J. Challenger. Improving web server performance by cacheing dynamic data. *Proceedings of the USENIX Symposium on Internet Technologies and Systems.*

[3] F. Douglis, A. Feldmann, B. Krishnamurthy, and J.Mogul. Rate of change and other metrics: A live study of the World Wide Web. *Proceedings o the Symposium on Internetworking Systems and Technologies.* USENIX, December 1997

[4] F. Douglis, A. Haro, and M. Rabinovitch.. HPP: HTML macro-preprocessing to support dynamic document caching. *Proceedings of the Symposium on Internetworking Systems and Technologies*, pages 83-94. USENIX, December 1997.

[5] G. Baga, F. Douglis, and M. Rabinovitch. Optimistic deltas for WWW latency reduction. *Proceedings of 1997 USENIX Technical Conference*, pages 289-303, Anaheim, CA January 1997.

[6] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnanmurthy. Potential benefits of delta encoding and data compression for HTTP. In Proceedings of SIGCOMM'97, pages 191-194, Cannes, France, September 1997.

[7] R. Caceres, F.Douglis, and A.Feldmann. Web proxy cacheing: The devil is in the details.

[8] S.D.Gribble and E.A. Brewer. System design issues for internet middleware services: Deduction from a large client trace. *Proceedings of the Symposium on Internetworking Systems and Technologies*, pages 207-218. USENIX, December 1997.

[9] S.Williams, M. Abrams, C.R. Standridge and C. Abdulla. Removal policies in network caches for world wide web documents. *Proceedings of the 1996 SIGCOMM.* pp. 293-305, ACM, July 1996.

[10] Thomas M. Kroger, D. E. Long, and J.C. Mogul. Exploring the bounds of web latency reduction from cacheing and prefetching. *Proceedings of the Symposium on Interworking Systems and Technologies*, pages 13-22. USNIX, December 1997.

[11] V.N. Padmanabhan and J.C. Mogul. Using predictive prefetching to improve world wide web latency. *Computer Communications Review*, vol 26, pp 22-36, July 1996.