Proceeding of the First International
Workshop on Scalable Web Service, SWS 2000,
with the 29th International Conference on Parallel Processing, ICPP 2000,
Toronto, Canada, pp-95-102.

Active Streaming in Transport Delay Minimization

Javed I. Khan
Internetworking and Media Communications Research Laboratories
Department of Math & Computer Science
Kent State University, 233 MSB, Kent, OH 44242
javed@kent.edu

Abstract

In this paper we present a technique for reducing response delay for web systems, which is based on a proactive cache scheme. It combines predictive pre-fetching and streaming to overlap the read-time with loading time. This graph-based model analyzes the hyperlink structure to form the prediction. It also utilizes data streaming to further minimize the pre-load, without compromising the responsiveness. The analysis demonstrates that such new hyper-graph based pre-fetching can reduce the lag-time of a cache system by a factor ranging from 2-10. In this paper, we focus on pre-fetching and provide the technique, the optimization algorithms and the simulation results.

Keywords: Multimedia, proactive internet caching, pre-fetching, streaming, delay optimization.

1.Introduction

Perhaps, one of the key difference between a classical distributed system and the emerging network-based applications is that in the later the performance is not only a function of the application algorithm but is also heavily dependent on detailed network characteristics such as bandwidth and congestion.

In today's internet environment, the network related delay range from few millisecond to minutes. Consequently, the impact of scrupulous improvement in application algorithm may go completely unnoticed by the end user, while typically they wait order of magnitude longer time experiencing transfer delay. The success and failure of many internet-based systems and applications depend on the user perception of the net delay.

The delay experienced by the user is contributed by many factors. In the critical path are network forward trip delay dominated by network latency, the server response time, server side application processing time, return trip delay dominated by down-stream bandwidth, and finally the parsing/rendering time at the client. The most commonly used approach for reduction of response delay is to increase the network bandwidth. However, quite often it is also the most expensive solution. The higher the bandwidth typically the lower is it's overall utilization.

A more effective means for reduction of response delay approach is network caching. Caching allows relatively static and repeatedly used documents to be stored in close proximity of the Browser. Cache can be placed right at the Browser, and/or few hops away to be shared by a group of clients. Cache reduces the overall path latency and removes the delay due to congestion in links upstream to the critical path.

In this paper, we investigate yet another approach for reduction of response delay- link-ordered pre-fetching. Typically, the rendering and reading time of a document is also quite large and is in human perceptual time scale. Potentially the transfer delay of subsequent documents or document components can be overlapped with the rendering and reading time of previous documents.

Interestingly, caching as well as pre-fetching both has been extensively applied in hardware systems to offset high latency of memory access in high performance systems. Over last few years, driven by the growing need of reducing network latency, caching techniques have been proposed for the Internet. However, it seems hardware cache operates in relatively more predictable environment, where the design variables such as page size, and access time differences among various storage stages are limited to few classes only. In comparison, the variability of file size, and network response lag faced in the internet are quite broad, also they are difficult to predict. The only advantage in the internet cache is that here the write back from primary to secondary storage is simpler as generally there is typically one writer.

However, studies indicate that principal of locality is less conspicuous in web access. A number of recent studies, with various innovative caching schemes reports caching efficacy in the rate of 30-60% [1, 11, 9, 5], where a modern hardware cache can achieve hit rate over 90%. Perhaps, one of the key reasons is that a large number of resources are new references. In the web, there is no short-term iterative construct in the causal chain above. In addition, due to the size limitation, resources those are fetched into the cache cannot be kept indefinitely, till it is requested again. Consequently, the hit ratio is significantly poorer than processor caches.

In this paper, we present a dynamically deployable proactive cache system that continually analyzes the link dependency and relative access frequency of documents around the access neighborhood of the clients. According to Markov potential based analysis it pre-loads selective high likelihood documents in the background, while the client consumes the current document in the foreground. Unlike the few other techniques proposed so far, our system analyzes the hyperlink structure for prediction. We call it *hypergraphic prefetching*.

Another novel feature of this pre-fetching technique is that it uses fragment *streaming* to minimize the pre-load, without increasing the response-lag. Each resource is dynamically divided into two parts the *lead* and the *stream* segments. The available bandwidth correspondingly is also separated into two sub-channels; *feed channel* for loading the streaming segment of the current resource, the *fetch channel* to proactively load the lead segments of future resources.

The size of the lead segment is computed optimally so that a minimum but just sufficient amount of data is pre-fetched and buffered. The rest is fetched by streaming, if and when the media is traversed. With the lead segment pre-fetched in the cache, streaming matches the consumption rate. Thus, it delivers content without any additional response delay.

2.Related Works

The internet caching has been studied for quite some time [1,11,9,5]. However, the study of pre-fetching is quite recent. In one of the pioneering studies, Kroeger et. al. demonstrated that with ample knowledge of future reference a combined caching and pre-fetching can reduce access latency as much as 60% [9]. The year after, Jacobson and Ca [8] proposed a method based on partial context matching for low bandwidth clients and proxies. Last year, Palpanas and Mendelzon [10] proposed an approach associated with servers. Both of these methods used variants of partial matching of context (past sequence of accessed references) for prediction of future reference. These methods proposed prediction based on reference history aggregated among the cached references.

Compared to the preceding techniques, we demonstrate that the prediction model can be made much more effective by bringing in the hyperlink reference structure among the neighboring web resources. Indeed, the freedom of context switching is strongly bounded by the underlying link pattern of the hyperspace. As we will demonstrate, with such hyperlink structure based prediction model, the access lag can be reduced by significant factor. A comparable example will be the 'Google' web search engine [3] that demonstrated significant increase in search efficacy by directly weighing in the hyperlink reference patterns among documents.

In a related work, Crovella and Bradford [4] studied yet another advantage of pre-fetching- the reduction of burstiness of network traffic. Bangla et. al. proposed pre-fetching of only modified cached content to reduce cache communication in their proposal of 'optimistic-delta' [2]. [7] has proposed XML extensions for document fragmentation, which can potentially be applied for streaming resources of wide classes. Also, in this work, we are building analytical models of the system to understand its characteristics. Prior works in the area are mostly empirically validated.

Proceeding of the First International
Workshop on Scalable Web Service, SWS 2000,
with the 29th International Conference on Parallel Processing, ICPP 2000,
Toronto, Canada, pp-95-102.

3. Hypergraphic Pre-fetching

3.1. Access Model

First, we describe the hyper-graph notation. Fig-1 shows the model. Each node in the hyper-graph represents a resource. Nodes are connected as per the embedded hyperlinks. The reader moves through a set of nodes in this hyper-space called anchor nodes. When a client (reader) program is active, the idea is to track client's movement or anchor sequence in this hyper-space, monitor its neighboring regions, and pre-fetch a subset of these nodes with high likely-hood of traversal into the proactive cache. The pre-fetching is done in the background while the current resource is being consumed by the client to overlap the loading with the reading time.

$H(V_H,E_H)$ denotes the entire hyper-space. However, a subgraph $h(V_h,E_h)$ is the visible sub-graph of H about which the cache has link and node information. However, for tractability, entire $h(V_h,E_h)$ is further pruned. Before each optimization phase, a pruned sub-graph called *roaming- sphere*s graph $G(V_G,E_G)$ is used for actual optimization. Although the node and link information of $h(V_h,E_h)$ or $G(V_G,E_G)$ is assumed available, but there content, however may not be resident in the proactive cache at the beginning. The algorithm uses $G(V_G,E_G)$ to determine the pre-loading schedule of the nodes within it. A subset of the nodes in $G(V_G,E_G)$ is eventually preloaded and added in the cache. C (VC, EC) represents the part of hyperspace, which is finally resident in the cache. Each node in $h(V_h,E_h)$ has read-time and size attributes. Each link in it has a transition frequency f(i,j) associated with it. Links are bi-directional and transition frequencies are asymmetric.

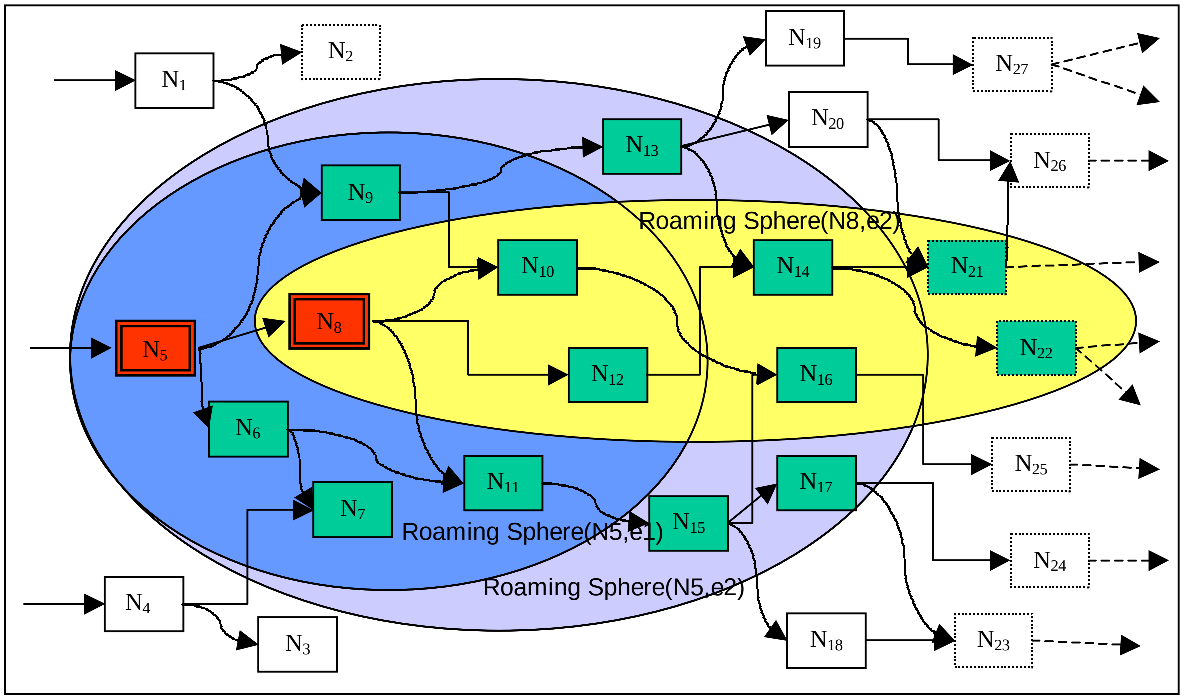

Fig-1: The Markov Predicton Network used for cache prefetching. The *roaming- sphere*s (N_5,e_2) and (N_5,e_2) show two analysis graph for two pruning thresholds both anchored at node N_5. The priority algorithm ranks nodes 5-12 (or 5-17 based on the selected threshold). The pre-fetch mechanism loads the nodes including N_8 accordingly, while the reader is reading N_5. By the time reader completes reading N_8 is preloaded and server. When reader moves to N_8, a new *roaming-sphere* is formed to build modified ranking.

3.2. Transport Model

The resource transport model has been designed after the generalized streaming delivery of content. Each resource has two parts the *lead segment* and the *stream segment*. The available bandwidth is correspondingly separated into two sub-channels; *feed channel* for loading the streaming segment of the current anchor, the *fetch channel* to proactively load the lead segments of resources from the focus zone. Fig-2 shows the event sequence for the transport model. We assume that D_{total} is the size of a resource, $D_{preload}$ is the bytes in lead segment and D_{stream} is the bytes to be streamed. The ratio of bandwidth allocated to sub-channels is α. Both α and the corresponding actual amount of D_{stream} can be dynamically adjusted based on optimization objectives. Streaming reduces the amount of date that is pre-fetched.

Fig-2 shows the analysis model where we group the nodes in the roaming-spare graph into three sets. The first is the anchor node itself which is traversed at stage i-1 A^{i-1}. The second is the set N_H which includes the nodes those have been assigned higher priority than the anchor node that actually will be traversed at stage i, and finally the set N_L, which includes the nodes those have been assigned lower priority than the anchor node at stage i-1. While, the client is rendering anchor A^{i-1}, the nodes of the stage i, are loaded in order of certain priority $P^i(n)$. If the set N^i_H is large, then the effectiveness of pre-loading disappears. If, the client completes reading the previous anchor node, while either N_H or N_L is being loaded, the system moves to the next stage.

3.3. Analysis

The model above provides the opportunity to answer a host of design questions. First and the most important one is what is the best pre-fetching sequence of the nodes that will minimize the expected *read-time lag* for a given network bandwidth? In this paper, we assume that the cache system is constrained by the bandwidth. Consequently, we deal with the above issue. Below we present the results:

Theorem-1: (Branch Decision) *In any node if, $T_1, T_2, T_3, \ldots T_n$ are loading times, and if f_1,, f, f_3,... f_n, are relative frequencies of the link traversal, the average delay is minimum if the links are assigned priority P_i and the node with the highest P_i is loaded first, where, the priority is computed as:*

$$p_i = \frac{f_i}{T_i} \qquad \text{...(1a)}$$

As a corollary, the above result can be extended further for a tree where the node priority can be determined for node N_i from the link traversal frequencies of the path. If 1,2,3,4..i are the links in the path from root (current anchor) to the node N_i. The priority function should be computed as:

$$p_i = \frac{f_1 \cdot f_2 \cdot f_3 \cdots f_i}{T_i} \qquad \text{...(1b)}$$

For general graph $G(V_G,E_G)$ where the node priority can be determined by computing order-n Markov potential Q_i for a node N_i, and where the priority function should be computed as

$$p_i = \frac{Q_i}{T_i} \qquad \text{...(1c)}$$

Corollary-1: *In a chain a zero delay streaming is possible for unit n in this sequence, if the cumulative presentation time $T_{p,i}$ for units before n, is larger than the cumulative loading time $T_{L,i}$ of units up to n. The entire sequence of N elements is zero delay if the following is true for all n up to N:*

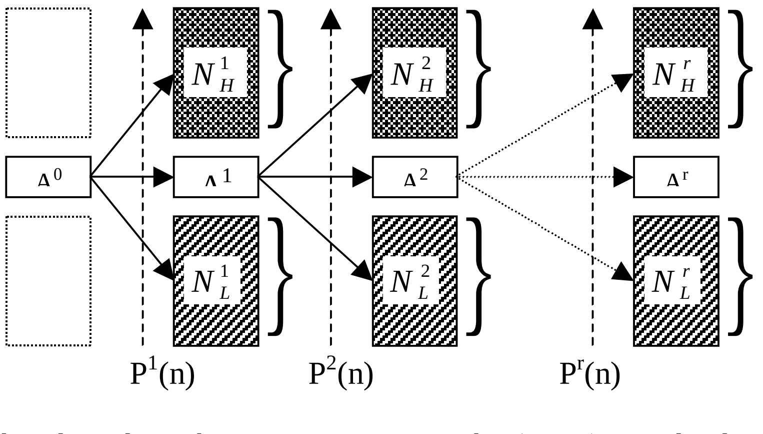

Fig-2. During each cycle, nodes in the *roaming- sphere*s graph $G(V_G,E_G)$ are ordered according to their Markov potential $P^1(n)$. The anchor nodes in each stage are shown as A. Each time, a set of nodes receives higher priority, and another set receives lower. The performance also depends on the relative size of these sets.

$$\sum_{k=1}^{k=n-1} T_{P,k} > \sum_{k=2}^{k=n} T_{L,k} \qquad ...(2)$$

Further, following expressions can be used to determine the size of the *lead* segments. Let us assume that the margin time=0, and α is the ratio of bandwidth allocated to the feed channel to that to fetch channel. We use the constraint that the reading time must be equal to the streaming time. Consequently, given a rendering rate R_{render} the amount of data that has to be pre-fetched for a resource is given by (3):

$$D_{lead} = D_{total}\left(1 - \frac{B_{feed}}{R_{render}}\right) = D_{total}\left(1 - \frac{\alpha \cdot B_{channel}}{R_{render}}\right) \qquad ...(3)$$

Corresponding pre-load time is given by:

$$T_{preload} = \frac{D_{preload}}{B_{preload}} = D_{total}\left(\frac{1}{B_{preload}} - \frac{\alpha}{R_{render}}\right) \qquad ...(4)$$

Only the D_{lead} amount of data should be pre-loaded for min delay browsing. It takes $T_{preload}$ time overlapped with previous reading time.

3.4. Schema

Now we briefly describe the action of the pre-fetch strategy designed on the above results. Each time the cache detects a new *user-agent* (embedded in HTTP header) it initializes a new pre-fetch session for tracking its roaming sphere. The pre-fetch algorithm first computes $G(V_G,E_G)$ by using a tree view of the graph. In this scheme the conditional frequency of the links in $h(V_h,E_h)$ is computed according to equation-(1b). Links reachable using multiple path is given the sum of the two scores. The prunned roaming sphere sub-graph $G(V_G,E_G)$ is then determined by considering links with loading score above a small cut-off threshold E. We call E the reach of the current roaming sphere.

The algorithm then computes the Markov potential Q_i for each node in the $G(V_G,E_G)$ network based on local relative transition frequency. Then it sets the pre-fetch priority of the nodes according to equation-(1c) with respect to the current anchor. The pre-fetch mechanism retrieves documents following this loading order.

However, it does not try to pre-fetch the entire document. Rather based on the bandwidth allocated for streaming, it determines the size of the lead segment for each document using equation-(3). It only pre-fetches the lead segments of the nodes in G, while it reads the remaining part of the current anchor via stream sub-channel.

The loading order remains valid until the reading time of current anchor node N. Upon completion of reading node N, a new node is traversed. The algorithm maximizes the likely-hood that that the new node is already in the cache. At this new anchor point, the focus graph is reevaluated. The subsequent evaluation is incremental. The pre-fetched nodes reveal more nodes from the hyperspace. In addition, the new anchor point changes the preference weights. Consequently, in this new graph, nodes downstream are upgraded by 1/f(i,j), while nodes brunching out from upstream are downgraded by factor 1/f(j,i).

3.5. System Description

An associated and interesting system design problem in the proposed pre-fetching scheme is the estimation of the link traversal statistics. The problem has two components: the collection and the propagation of access statistics. A number of strategies can be potentially used. We are currently investigating a few. Below we briefly outline two approaches.

In the transparent client scheme, a server side plug-in can collect the access statistics for its documents spanning the hyper-space local to it. Even without any explicit notification mechanism from the clients, it can detect the source of the traversals by analyzing the reference log. Perfect frequency statistics, can also be collected by embedding the referral source identification in the request message of the transport protocol. From the source URLs, the server plug-in can track the reverse links. This can be implemented within current HTTP 1.1 by using the 'referer' request-header field [6]. We suspect the system does not have to be meticulous. We estimate keeping statistics only for the top links for the 'hot' documents will be enough to create visible performance improvement.

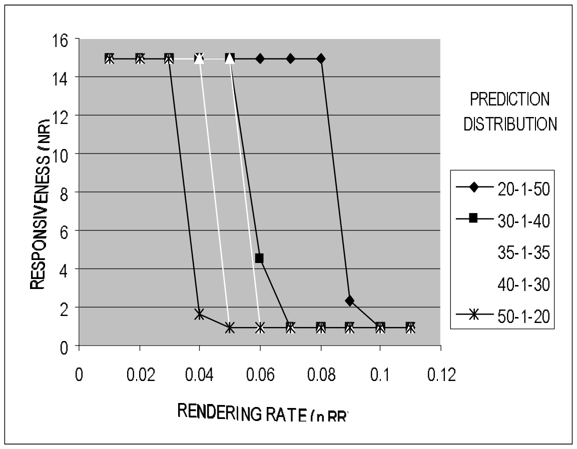

Fig-3(a) Responsiveness for slow rendering media.

Two strategies are possible for the propagation of access statistics. The statistics can be embedded right into the markup documents as a special link attribute. This can be done in XML. An extended HTTP directive can make pre-fetching cache systems aware of the embedded attributes, conveniently tripping the pre-fetch loading mechanism. In an alternate scheme the statistics can be appended directly as a vector attribute in the response header, without modifying the content. (it too can be implemented using reserve pool without HTTP protocol extension).

The collection mechanism can be extended to a larger hyper-space by letting a group of co-operating server cache systems to periodically exchange the local link statistics and build up statistics for larger hyper-spaces spanning a community of servers. The communication can be potentially minimized efficiently. Since the link sinks already have the URLs of the link sources, sink server plug-ins can initiate occasional communication, and deliver the statistics directly to the sources corresponding to heavy access links. We are currently investigating how this expanded model of co-operating hyper-space statistics collection and propagation model can be deployed and managed with Active Net.

4.Results

We have performed a series of simulations of

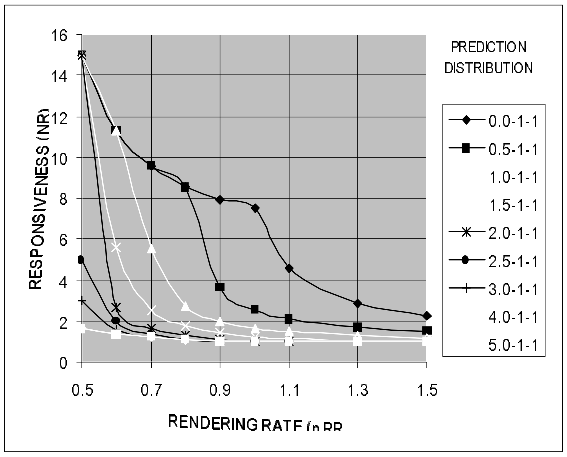

Fig-3(b) Responsiveness for line rate matched media.

the proposed system to evaluate the performance under a number of scenarios. To measure responsiveness we observed *normalized responsiveness* (NR) (y-axis). It is the response lag experienced by the proposed *hypergraphic pre-fetch* scheme normalized with that experienced by the scheme without pre-fetching. It indicates the factor by which pre-fetching scheme reduced the response lag.

The reduction in response lag is dependent on the ratio of *network bandwidth* to the *rendering rate* (such as text reading speed, play rate for audio or video). We call this ratio *normalized rendering rate* (nRR). Fig-3(a) plots the performance of the system when the ratio varies from .01-.1. Fig-3(b) shows the plot for content type with faster rendering rate factor close to 1. The former approximately represents the uncompressed text reading scenario on about 10Kbps network, typical of low speed wireless network. The later represents compressed multimedia presentation on WAN, which are typically rate matched for that speed.

As evident in both the cases, the response lag improved in the range between 10-2. In both cases, we measured the improvement of responsiveness. As expected, the gain from pre-fetching is more dramatic for slow rendering content.

The performance of the system is also affected by the accuracy of the prediction model. As shown in Fig-2, we assumed various amount of prediction error. Distribution vector X-Y-Z respectively represents the bytes fetched before anchor (for nodes ranked with higher priority than the actual anchor), the byes in actual anchor (eventually the bytes which are read), and the bytes fetched after the anchor (for nodes ranked with lower priority than the anchor node).

Fig-3(a) plots the curves when in total 71 units (X+Y+Z=71) have been fetched, but with various High and Low priority bytes distributions. As shown the responsiveness improved in the range of 1-15. Clearly, with lower size of pre-anchor bytes resulted in sustained performance improvement even with high rendering rate. In Fig-3(b), we kept Y=1 and Z=1 which varied X from 0-5, indicating perfect prediction (no pre anchor byte) to almost five times pre anchor bytes. As evident, the system responded faster almost by the factor 10-2 times. The above plots demonstrate the performance improvement without streaming.

Fig-4 plots the potential reduction of the lag that can be achieved by combining streaming. It plots the lag-time (normalized by the lag-time of cache without pre-fetching). It shows the factor (y-axis) by which the lag time improves with respect to percentage of bandwidth (x-axis) assigned to the fetch channel (LBF).

It also plots 4 curves showing the performance for the ratio of rendering rate to a total bandwidth varying in the high range from 1.0 to 2.0. We also used high odd in the distribution with vector 0-1-1. Without streaming (full bandwidth allocated to fetch LBF=1) the responsiveness improves by a factor of 1.66-7.5. As evident, with the incorporation of active streaming, the read-time lags now decrease by another factor of 2-10. Also interestingly, we achieved the above performance by tracking a relatively small focus graph consisting of 10 nodes associated with each user.

5.Conclusions & Current Work

The effectiveness of pre-fetching is particularly significant for the internet reference compared to hardware instruction. The rate of repetition of reference is quite low in web cache, compared to hardware cache. It seems that a key reason might be that web-reference pattern does not have any looping construct. Such looping construct is quite common in hardware instruction reference. Basic caching alone is only effective for repeat reference and cannot help in reducing the access lag for new references. The pre-fetch prediction model on the other hand, reduces access lag in the face of new document references. Thus, pre-fetch model appears to be even more attractive technique for the internet domain.

In this research we have presented a pre-fetch scheme, which when integrated with streaming, apparently promises an optimum reduction in wasted pre-fetch, as well as ensures

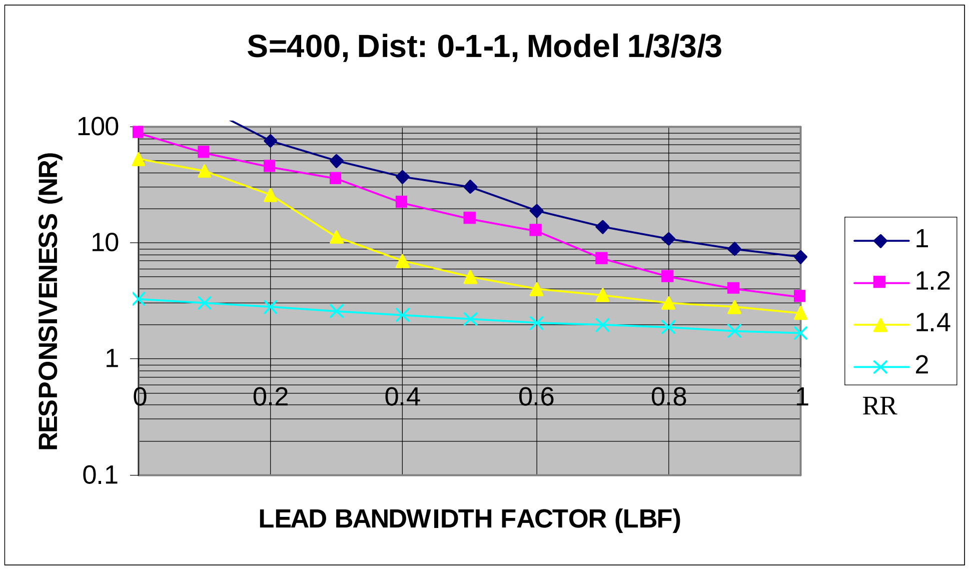

Fig-4 Effect of Active Streaming

responsiveness of the web system. We have also attempted to obtain an analytical model of the scheme. We have demonstrated how to obtain the minimum response lag through the Branch Decision Theorem based on the estimate of the node access frequencies.

Fundamentally, the efficacy of the model, however, will depend how well the statistics collected on the past access frequency can reflect the access frequency for which the theorem optimizes. Although, such stationary nature of underlying stochastic process has been safely assumed in wide range of other engineering systems, an interesting next step will be to carry out empirical analysis designed to measure the site-wise efficacy of such prediction in web. Interestingly, the frequency function can be generalized into other priority functions (such as priority of clients). It will still optimize the cost on that basis.

We have also outlined techniques by which the scheme can be implemented. By all likelihood, current transport protocols (such as HTTP, XML) will continue accepting innovative additions (a number of high pay off optimizations are already on the horizon), and therefore, their current constraints should not be a major concern in exploring new optimizations methods in such early stage of the technology. Nevertheless, the proposed prefetching can be implemented without requiring any major modification of the transport protocol.

The pilot investigations which predate ours, mostly presented empirical case study (some on large data set found in a particular web-site), however without any attempt to formalize the problem. Consequently, it remains unknown if the readings can be duplicated on a different scenario, or which factors play how much role in the cost performance equation. More formal and analytical understanding of a problem of such importance is surprisingly lacking. This research is part of our broader initiative to fill this gap.

It seems the performance of an effective web system will eventually be built with various custom systems appliances. Prefetcher and server plug-ins for access statistics collection and propagation and are examples of such custom appliances. Clearly, the deployment and management of such complex systems will be a formidable task. As a part of our ongoing research, we are currently investigating an active net deployable stream pre-fetcher and plug-in modules, which can be dynamically launched and managed as active proxies in network.

The work is currently being funded by DARPA Research Grant F30602-99-1-0515 under it's Active Network initiative.

6.References:

[1] Marc Abrams, Charles R. Standridge, G. Abdulla, A. Edward, Fox and Stephen Williams, Removal policies in network caches for World-Wide Web documents, ACM SIGCOMM '96. Stanford, CA, 1996, pp 293-305.

[2] G. Banga, F. Douglis, and M. Rabinovich. Optimistic Deltas for WWW Latency Reduction. In Proc. 1997 USENIX Technical Conf., pp. 289-303. CA, January, 1997

[3] S. Chakrabarti, B. Dom, R. Kumar, et. al, Hyperserching the Web, Scientific American, June 1999. [Last retrieved from: http:// www.sciam.com/1999/0699raghavan.html]

[4] M. Crovella, P. Barford, The Network Effects of prefetching, Proc. Of IEEE INFOCOM, San Francisco, USA, 1998.

[5] F. Douglis, A. Feldman, B. Krisnamurty and J. Mogul, Rate of Change and Other Matrices: A Live Study of the World Wide Web, Proc. Of USENIX Symposium on Internet Technology and Systems, Barkeley, December 1997, pp-147-158.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk & T. Berners-Lee, Hypertext Transfer Protocol HTTP/1.1, RFC 2068, January 1997.

[7] Grosso, Paul, Daniel Veillard, "XML Fragment Interchange", W3C Working Draft 1999 June 30, [Retrieved from: http://www.w3.org/1999/06/WD-xml-fragment-19990630.html]

[8] Q. Jacobson, Pei Cao, Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies, 3rd International WWW Caching Workshop, Manchester, England, June 15-17 1998.

[9] T. Kroeger, D. D. E. Long & J. Mogul, Exploring the Bounds of Web Latency Reduction from Caching and prefetching, Proc. Of USENIX Symposium on Internet Technology and Systems, Barkeley, December 1997, pp-319-328.

[10] P. Themisoklis & A. Mendelzon, Web Prefetching Using Partial Match Prediction, The 4th International. Web Caching Workshop, San Diego, 1999.

[11] Duane Wessels and K Claffy, ICP and the Squid Web Cache, IEEE Journal on Selected Areas in Communication, April 1998, Vol 16, #3, pages 345-357.