TR 01-004

Web Proxy Server with Intelligent Prefetcher for Dynamic Pages Using Association Rules

Ajay Bhushan Pandey, Jaideep Srivastava, and Shashi Shekhar

January 26, 2001

A Web Proxy Server with an Intelligent Prefetcher for Dynamic Pages Using Association Rules

Ajay Bhushan Pandey
pandey@cs.umn.edu
Computer Science Department
University of Minnesota
Minneapolis MN 55455

Jaideep Srivastava
srivasta@cs.umn.edu
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

Shashi Shekhar
shekhar@cs.umn.edu
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

Abstract

The growth of the World Wide Web has emphasized the need for improved user latency. Primarily, two techniques, i.e., caching and prefetching are being used for improving user latency. Several studies have been conducted on cache replacement policies for improving the cache hit ratio. On the prefetching side, studies have been conducted on prefetching models based on decision tree, Markov chain, and path analysis. Increasing use of dynamic pages, frequent changes in site structure, and user access patterns on the Internet have limited the efficacy of caching techniques and emphasized the need for prefetching with a predictive model which is easier to build and update and which has improved predictive performance. In this paper, we studied the existing caching and prefetching techniques and explored as to how the knowledge of user access patterns discovered through web mining and information on site structure can be used to predict future requests by user. In a web site, certain sets of pages exhibit strong correlations with each other, which manifest in user access patterns. Such sets of correlations can be discovered in the form of association rules by web mining of server logs. We accordingly propose a prefetching model based on association rules. We also present a predictive model based on site structure where a certain number of children of the current page are prefetched. In addition, we also present a design and prototype implementation of a proxy server which uses these models for prefetching as well as results of experiments to evaluate the performance of these two models. In our experiments we found that the association rules prefetching model has better predictive value than the site structure model and gives a good cache hit ratio without much additional traffic load.

1. Introduction:

The exponential growth of the World Wide Web has caused a dramatic increase in Internet traffic and serious performance degradation in terms of user latency and bandwidth on the Internet. It has also made it critical to look for ways to accommodate increasing numbers of users while preventing excessive delays, congestion, and widespread blackouts. This has led to a great deal of effort devoted to studying web caching and prefetching techniques through the use of proxies. Most web caching

schemes maintain global caches, which straddle across users. Because pages use static html and hence are the same for all users, global caching works quite well.
However there is an increasing trend of generating dynamic pages in response to http requests for the following reasons:

1. For personalizing customer experience, the trend is to send personalized pages based on user profile and preferences so that information on a page is relevant to the user. Personalized pages also save multiple round trips before the user finally gets the information relevant to him.

2. Since data content on a page needs to be updated frequently, making changes in static pages is impractical. It is much cheaper and faster to maintain a database, which is updated continuously, and to generate pages from the database on the fly. Pages displaying stock quotes, weather information, shopping/auction sites (where products, costs, and players change every moment) are generated dynamically. A page actually acts as a placeholder for the data, which is updated frequently.

3. Pages containing user specific account information/transaction are generated dynamically. These pages are relevant only to the specific user at a particular time. These pages cannot be cached.

4. All response pages on a secure connection are generated dynamically as per the password and other security features such as encryption keys. These pages expire immediately by setting the Expire field and/or by the Pragma directive of ‘no-cache’ in the HTTP header of the server response, to prevent them from being misused in a Replay attack.

As the Internet grows and becomes a primary means of communication in business with all its variants (B-2-B, B-2-C, C-2-C and C-2-B), the majority of pages will tend to be dynamic pages. In such a situation, traditional web caching will be unworkable. As a result, research efforts are on to look for methods to improve user latencies for sites with dynamic pages. Moreover since the generation of a dynamic page requires processing on the server side after receiving the HTTP request, it further degrades user perceived latency. Ideally, while a user is going through the page, if the server can predict future requests and pre-generate them, it can improve user latency. We can consider another model where a proxy server is sitting between the server and the end user. If the proxy server is able to predict future page requests and prefetch and keep them in its local cache, latency can be improved. However, there is a limitation to this approach- prefetching can reduce user latency but only at the cost of increased bandwidth since many of the prefetched pages may not be requested by the user and thus cause an increase in traffic. Therefore, attempts are being made to design an intelligent prefetcher based on an improved prediction model which will improve user latency with a minimal increase in bandwidth.

1.1 Our Contribution

In this study we carried out the following tasks:

1. Surveyed existing research on web prediction models and prefetching techniques

2. Developed two intelligent prefetching schemes which take into account user access patterns (derived from association rules generated from server logs) and site structure.

3. Designed a proxy server with the proposed intelligent prefetching schemes and implemented a prototype version of it for a single client.

4. Extracted knowledge in the form of association rules from a web log of a commercial site server and trained the proxy server with this knowledge.

5. Evaluated the performance of the proxy server using the above two intelligent prefetching schemes and analyzed and compared the results, i.e. cache hit ratio and bandwidth.

1.3 Outline and Scope

In Section 2 we survey the current research on web caching and prefetching techniques and discuss the limitations of the current approaches and define our problem statement. Section 3 presents two intelligent prefetching schemes, one based on association rules and the other based on site structure. In Section 4 we describe the architecture and our implementation of a proxy server which uses the two prefetching schemes. Section 5 describes the experimental design and the results of the experiments conducted for evaluating the proposed intelligent prefetching schemes. Finally, in Section 6 we conclude by summarizing the main results and noting future research opportunities in this area.

Scope: Due to limitations of time, in this study we did not examine the effect of cache size on the performance of the proposed model. We also did not evaluate the predictive performance of the model based on sequence rules.

2. Survey of Existing Research on Prefetching Techniques

The issue of caching and prefetching has mostly been studied with regard to static HTML pages and finding an appropriate prediction model to determine prefetching and caching/replacement strategies. The efficacy of prefetching and caching strategies depends upon how accurate the prediction model is. The prediction models can be categorized into two groups: one based on local knowledge of the access pattern and preference of the client, and the other based on the server's knowledge of access patterns and popularity of its documents. In the first group of models, the proxy sits near the client and makes a prediction of future URLs based on the access pattern of the client, prefetches pages, and performs cache management according to the prefetching and caching strategies derived from the model. In the second group of models, the Proxy acts as an agent of the server which has full access to server data, i.e., popularity of the documents and access pattern of all clients, and performs prefetching and caching operations. In this study we concentrated on the models based on server knowledge and its use for determining prefetching strategies.

The Top-10 approach, proposed by Markatos and Chironaki [1], is based on the cooperation of clients and servers to make successful prefetch operations. This work deals mostly with the prefetching of static HTML pages. The server periodically calculates a list of most popular documents and makes them available to the clients, which they can then prefetch. In order to make sure that the list is sent only to the clients who can potentially use them, Top-10 classifies clients on the basis of the number of requests they make. Prefetching is activated only after a client has made a sufficient number of requests to the server. The study proposes adjusting the number of documents in TOP-10's list and access threshold to optimize latency and bandwidth. In a trace driven simulation taken from server traces of web servers of Parallab, the Super Computing Center associated with the University of Bergen, Norway ; Computer Science Department of University of Rochester, NY, USA, NASA's Kennedy Space Research Center; and FORTH (Foundation for Research and Technology Hellas), it was found that the TOP-10 approach manages to prefetch up to 60 % of requests with a 20 % corresponding increase in traffic.

A Study at Microsoft and Harvard University [2] suggested using path profiles to predict HTTP requests. This study specifically targeted prefetching dynamic pages. Its model is based on matching the path of the current user with that in the server database derived from the past access history. A path is a sequence of URLs accessed by a single user, ordered by the time of access. A path profile is a set of pairs, each of which contains a path, and the frequency of the path, i.e., the number of times the path occurs over the period of the profile. The study suggests generating a tree from the path profile consisting of paths whose frequency exceeds a certain threshold. They suggested a threshold to limit the size of the tree, and drop the paths which do not occur frequently. For predicting future path P, a user's current session in the form of a path S is passed and is matched with the tree. The most recent URL accessed is the most important predictor because the user is likely to select his next URL from the page of this URL. Therefore, matching is worked backwards from most recent to least recent, applying S to the path tree for predicting the next URL. This backward stepping is repeated to generate all paths, which are suffixes of the path generated from S working backwards from most recent to least recent. The longest matching path is used for prefetching. This algorithm was tested on server traces of www.planetall.com, the Domino based server of Lotus Corporation, iisa.mocrosoft.com (containing sample ASP pages), and www.microsoft.com. The authors claimed a predictability rate of over 40 % in each case.

A study [3] at the Multimedia Technology Research Lab in Korea sought to enhance the hit-rate performance of cache servers by prefetching ‘ Brand new Document.’ This study deals only with static HTML pages. In this approach, the server pushes to the client, along with the response to the current HTTP request, all new documents in anticipation that these documents will not be in the client's local cache and the client is likely to make requests for these new documents. The authors in their experiment observed that when only caching is used the hit ratio is very low in small cache size zones, but increases to 40 % if caching is combined with prefetching. However, when cache size is increased, marginal gain of prefetching over simple caching diminishes; this is expected because for a very large cache, most of the documents would probably be found in the cache, and no additional benefit would be gained by prefetching. The authors also found that in the lower cache size zone, where the additional benefit in cache-hit ratio is about 40%, the traffic increase is only about 25 %. The authors also suggested combining Brand New prefetching with the TOP-10 approach and sending the client's list of cache with its HTTP request through 'Piggybacking' to improve the cache hit performance.

Padmanabhan and Mogul [4] described a server-knowledge based predictive model based on the algorithm by Griffons and Appleton [12] with certain modifications. The prediction algorithm constructs a dependency graph that depicts the pattern of access to different files stored on the server. The graph has a node for every file. There is an arc from node A to node B if B was accessed within W accesses after A, where W is the look ahead window size. Every arc is assigned a weight which is the ratio of the number of accesses to B within a window after A to the number of accesses to A itself. It may be noted that this weight is not actually the probability that B will be requested immediately after A. The dependency graph is dynamically updated as the server receives new requests. When A is accessed, the server will send a hint to the client to prefetch B if the arc from A to B has large weight since there is a good chance of B being accessed within the W window. The authors conducted simulations based on this predictive model and measured two performance metrics – average access time per file and fractional increase in network traffic. They found that average access time reduced by 50 % but at the cost of an 80 % increase in network traffic. They also found that beyond a point, aggressive prefetching follows the law of diminishing returns and has no significant impact on access time.

Azer [5] suggested a speculative model for predicting future requests. His model is derived from the statistical information that the server maintains for each document it serves. The statistical information is used to generate a probability matrix P, where $p[i..j]$ is the conditional probability that a document d_j will be requested within a limited window time T_w ,given that d_i has already been requested. The server log was also used to generate closure of P ,i.e., $p^* = p^N$. These matrixes were later used for speculating which document a user will request next and this information was used for prefetching. When a request for a document d_i is received, the server responds by sending to the client the documents d_j that satisfy an inequality based on functions P and P^*. This inequality determines the particular policy employed. For example, it could send hints for prefetching document d_j along with a requested document d_i if $P^*[i, j] \geq t_p$ for some threshold property $0 < t_p \leq 1$. This model was tested on the server log of the Computer Science Department of Boston University. The results showed that using only 5 % extra bandwidth resulted in a 30 % reduction in server load, 23 % reduction in service time and 18 % reduction in client miss rate. Using 10 % extra bandwidth results in a reduction of 35 %, 27 % and 23 % in these metrics, respectively. The results also showed that beyond some point, excessive speculation does not seem to pay off. An aggressive speculation that causes 50 % increase in traffic yields a 45 % reduction in server load, 40 % reduction in service time, and 35 % reduction in client miss rate. Increasing traffic by another 50 % improves performance of server load, service time, and client miss rate by only 7 %, 6% , and 2% respectively.

Bonchi, Giannotti, Manco, Nanni, Pedreschi, Renso, Ruggieri[8] studied a web predictive model based on decision trees for adaptive web caching. They mined web logs to develop a predictive model in the form of a decision tree which , given a request for a web object A, will predict the time of the next request for the web object A. A decision tree is a data structure consisting of decision nodes and leaves. The root node represents the set of all training cases. A decision node specifies a test over one of the attributes, which is called an attribute selected over the node. For each possible outcome of a test over an attribute selected for a node, a child node will be present. The cases abstracted at the node are partitioned among the child nodes according to the possible outcomes of the test. The decision tree is used to classify a case, i.e., to assign a class value to a case depending on the values of the attributes of the case. A path from the root to a leaf of the decision tree can be followed based on the attribute values of the inner nodes . The class specified at the leaf is the class predicted by the decision tree. In this model, the training tuples were derived from http requests in the web logs. Each training tuple was observed on attributes such as *Ndir* , *FirstDir* , *Hour* , *FileExtension* , *EntitySize* , *LastAccess* , *LastKAccess* .

Where

Ndir	= Depth in the server file System of the requested response
FirstDir	= Root directory of the path in the server file system containing the requested resource
FileExtension ,	= File extension of the requested resource
EntitySize	= File size of the requested resource
LastAccess	= How long in the past in terms of number of requests the URL was requested
LastKAccess	= Number of requests to the same URL in last K requests

In the model, *NextAccess* (number of requests after which the current URL will be requested) was chosen as the class variable. C4.5 algorithm was used to construct a decision tree based on the training sets and the above class variable. A decision tree thus constructed was used to predict the value of *NextAccess* for a given web object. The authors proposed the ORCL strategy, an extension of SLRU (Size adjusted LRU). According to the ORCL strategy, weight of an entity is $\Delta' T * size$, where $\Delta' T$ is the number of requests that will be received until the next access to the object. The entities with higher values of $\Delta' T * size$ are removed first from the Cache. The decision tree built in the model is used to predict the value of the class variable *NextAccess* that is essentially $\Delta' T$.The authors also defined another strategy S2 which uses features of both LRU and ORCL. It assign a weight to each entity E_i
as under

$$W_{s2}(E_i) = i + \alpha(\Delta\ T) * \beta * (E_i.size)$$

for some inversely proportional functions $\alpha(\ )$ and $\beta(\ )$.
The performance of strategies S2, LRU, and ORCL was compared in terms of hit rate for two workloads NatPort[8] and Berkley[8]. S2 performance in terms of hit rate was 25 % higher than LRU. Also, prediction accuracy was 70 –80 % for Berkley and 84% to 87 % for NatPort. The authors observed that with a Pentium III 650 MHz processor and cache size of 409.2 M bytes, the discretization process takes one hour and tree construction takes 10 minutes.

Sarukkai[9] proposed a probabilistic model for web link prediction and path analysis using Markov chains. The Markov Chain Model consists of a matrix of state transition probabilities, and initial state probability vector λ

$$\hat{S}(t) = \hat{S}(t-1)A$$

where
$\hat{S}(t)$ and $\hat{S}(t-1)$ are probability vectors for all states at time t and $t-1$ respectively
A = Matrix representing transition probabilities from one state to another.
If there are n states , then the matrix A will be of order $n \times n$

The Matrix A can be estimated as follows:

$$A(s,s') = \frac{C(s,s')}{\sum_{s''} C(s,s'')}$$

$$\lambda(s) = \frac{C(s)}{\sum_{s'} C(s')}$$

where
$C(s,s')$ is the count of the number of times s' follows s in the training data.
All client requests are buffered into a client buffer and flushed once a minimum threshold is exceeded. Given a start URL, the tour generator outputs a sequence of states which corresponds to the tour generated by the model. The tour predicts a sequence of URLs which will be visited next by the client. The training and simulation were conducted on

the data collected at Research Triangle Park, North Carolina. It was observed that 50 % of the requests could be predicted with the Markov chain prediction model.

Zaine, Xin, and Han[10] presented a design of a web log miner, a knowledge discovery tool for mining server log files for discovering web access patterns. In the first stage, data from the web logs is filtered to remove irrelevant information and a relational database is created containing the meaningful data. In the second stage, a data cube is constructed using available dimensions. On-line analytical processing is used in the third stage to drill down, roll-up, slice, and dice in the web log data cube. Finally, in the fourth stage, data mining techniques are put to use with the data cube to predict, classify, and discover interesting correlations.

3. Limitations of Current Solutions and Problem Statement

The TOP-10 approach [1] mostly deals with static pages and requires co-operation of the clients. One may argue that this approach can be extended to dynamic pages where a server will always send TOP-10 dynamically generated documents to its clients with request response. This approach causes a massive increase in traffic. Another problem with this approach is that it presumes that all users have the same preferences and therefore sends the same TOP-10 list to everyone who has accessed more than a threshold number of pages. It also does not keep track of a user's history of accesses during the current session.

The Brand New Approach [3] is also not workable for sites with large numbers of dynamic pages since all dynamic pages will fall in the category of ' Brand New Documents' and all of them will have to be sent to every user. This will increase network traffic by several orders of magnitude and will thus be impractical. The authors have themselves noted that this approach would work only in the case of static html pages.

Though Azer[5]'s work deals with the prefetching of dynamic pages, it has certain limitations. His prediction model is based on pairwise probability, i.e. the conditional probability of a user requesting a page d_j given that he has asked for page d_i. The underlying assumption behind this model is that the next request is only dependent upon the previous request. The validity of this assumption is questionable because when a user is visiting a site, his next request is not only a function of only his current request but of all his past requests as well.

The Path Profile approach by Schechter et al.[2] has been specially proposed for dynamic pages. The authors have found prediction rate at above 40 % in their simulations. The limitation with this method is its on-line computational complexity, which may limit the prefetching performance. What is important here is that the algorithm should not only make good predictions, but also do so quickly so that the pages are prefetched before the next request comes. The authors have not given an analysis of the size of the profile tree and the computational complexity. The concern is that a site having a large number of dynamic pages may result in a huge profile tree. This tree needs to be traversed at the receipt of every request at the server/proxy server end. If the prediction of the next request is computation intensive, then this will affect the performance of the proxy. Moreover, it is not clear why the authors have chosen the longest path that is a suffix of the current path. It is possible that a smaller path which too is a suffix of the current path may have a higher probability of occurrence. Thirdly, this approach assumes that a web access pattern is strictly sequential. It treats $(a \to b \to c)$, $(c \to b \to a), (b \to a \to c), (a \to b \to k \to c)$ as different paths; this may not be correct, because if (a,b,c) and (d,e) are strongly connected objects, then there is good likelihood that (d,e) will be requested not only in the case of an $(a \to b \to c)$ path but in other paths as well, i.e. in case of $(c \to b \to a), (b \to a \to c), (a \to b \to k \to c)$.

The principles used in Padmanabhan and Mogul's work[4] can be extended to make predictions for prefetching dynamic pages, but the work has limitations similar to the model proposed by Azer[5]. The dependency graph in their work is basically a collection of pairwise dependencies, i.e., if A has been fetched then what are the chances of fetching B. This method does not take care of set dependencies, i.e., if the elements of set (a,b,c) have been fetched then what are the chances that the elements of an another set (d,e) will be fetched. In this sense the prediction model is inadequate. Moreover, the model involves on-line creation and updating of dependency graphs which places additional load on the server and may affect its performance.

Bonchi et al.'s [8] suggestion of use of decision trees in predicting web access patterns and its use in cache management can be extended to prefetching. Similarly Sarrukai's [9] 's Markov chain predictive model can be used in predicting web access patterns. Both decision tree and Markov chain approaches are theoretically capable of generating almost complete predictive models provided they are trained well with a very large training data and node attributes are properly selected. But large training data will result in a huge decision tree. The efficacy of the decision tree model also depends upon how it has been trained i.e., how the nodes attributes have been identified. Similarly in the case of the Markov Chain model, a web site with a large number of pages , say n , will generate Markov transition Matrix of the order of $n \times n$. which will make these models very difficult to update. User access patterns are dynamic as they change over time. Therefore any predictive model must be easily updateable. Any attempt to reduce computational complexities to make these two models more easily updateable will require pruning of data, which may cause loss of vital data and take away the completeness feature of these two models.

3. Intelligent Prefetching

Our goal is to address the limitations in the above models and propose a predictive model which is computationally less complex, updateable, simple to implement, flexible, and yet provides good latency-bandwidth performance. We discuss in the following paragraphs two possible models : one based on association rules and the other based on site structure.

3.1 Association Rules Based Intelligent Prefetching

The proposed approach is primarily based on knowledge discovery of server access patterns through data mining. The assumption is that within a site, there are certain sets of URLs which exhibit strong correlations/associations with each other. The limitation with some of the models described earlier is that they rely on pairwise dependencies rather than on set dependencies. In fact, pairwise dependency to some extent is a very narrow and special case of set dependency, and, therefore, any predictive model based on pairwise dependency alone will be less accurate and have limited predictive value. A web site can be compared to a super store which has several departments offering several products. A user goes through various departments and pick out the products. There could be a strong association among product sets even though none might exist among its individual products. Similarly web browsing of web sites and particularly e-commerce sites is expected to exhibit similar behavior, i.e. a client who has visited pages A,B,C is also likely to visit pages D,E. We can discover set dependencies among the pages of a site using data mining in the form of association /sequence rules.

*Association Rules***:** These rules correlate the presence of a set of items with another range of values for another set of variables. An association rule is of the form

$$LHS \Rightarrow RHS$$

$$X \Rightarrow Y$$

where $X = \{x_1, x_2, x_3 \cdots x_n\}$ and $Y = \{y_{1,} y_2, y_3, \ldots\ldots y_m\}$ are sets of items with x_i and y_j being distinct items for all i and j. This association rule states that if a client visits X, then he is likely to visit Y. The support for the rule LHS => RHS is the percentage of client sessions that hold all the pages in the union, i.e. the set LHS U RHS. A low support will imply that there is no overwhelming evidence that the pages in LHS U RHS occur together in a majority of the sessions. For computing the confidence for the Association Rule LHS => RHS , we take all the user sessions which contain the LHS and the confidence of the rule is the percentage of such user sessions that also contains the RHS. Since usually such transaction databases contain extremely large amounts of data, current association rule discovery techniques try to prune the search space according to the *support* for the items under consideration[11].

Sequence Rules: Sequence rules give a set associations in the temporal domain, i.e., X=> Y means that if all pages in X have been fetched, then there is a likelihood that the pages in set Y will also be fetched within a certain time window. In other words, X precedes Y.

The next issue is to make a choice between association rules and sequence rules in the proposed prediction model. We prefer association rules for our model because sequence rules, being temporal in nature, tend to be more restrictive than association rules. A minor deviation by a user will cause a mismatch in the sequence rule and deprive the prefetching of its benefits. Users may not always follow a strict sequential pattern while browsing a site. More often than not, a user will visit some other links and come back to his usual trail. Moreover, for an e-shopping site, it is not always possible to say that the user would purchase X before Y. If X and Y are strongly related items, then we can say that X and Y will both be purchased. However, the sequence in which items in X and Y will be purchased will be indeterminate.

A predictive model based on association is computationally less complex to build. Since it does not seek to build a complete model, it needs less training data than required by decision tree and Markov chain models. It is also not as sensitive to the selection of attributes as the decision tree model is. Several fast and efficient algorithms have been developed to mine association rules[13,14,15]. This feature makes association the predictive model more suitable for use in web prediction because it can be frequently updated to take care of the changing nature of web access patterns. It is true that the predictive model based on association is less complete than those based on decision trees and Markov models, yet it is preferable because of the lower computational workload involved in building the model and its update ability. In fact, because the association rule model is easier to build and can be updated very frequently, it could be complement the Markov and decision tree models. Instead of frequently building decision tree and Markov models afresh to deal with the problem of changing access patterns, the knowledge gained through the association rules model can be used to train and generate the decision tree and Markov models.

We therefore propose a prediction model based on association rules and an intelligent prefetching scheme, i.e. the Association Rules Prefetching((ARP) scheme.

3.2 Site Structure Prefetching Scheme

We also propose a Site Structure Prefetching(SSP) scheme which uses site structure information and usage statistics to predict future page requests and compare it with prefetching with association rules. In the Site Structure Prefetching scheme, a certain number of children of the current page ordered by the usage count are prefetched. If a user is on a page, there is a good likelihood that the user will click one of the links on the current page, i.e., request one of the children of the current page. If he is going to request a child of the current page, then there is a good chance that he will request for a popular one.

4. Intelligent Prefetching Based Proxy Server

Based on the prediction models discussed earlier, we have designed a proxy server whose architecture and implementation are described next in this section.

4.1 Proxy Server Architecture :

A block diagram of the architecture of the proxy server is shown in Figure 1

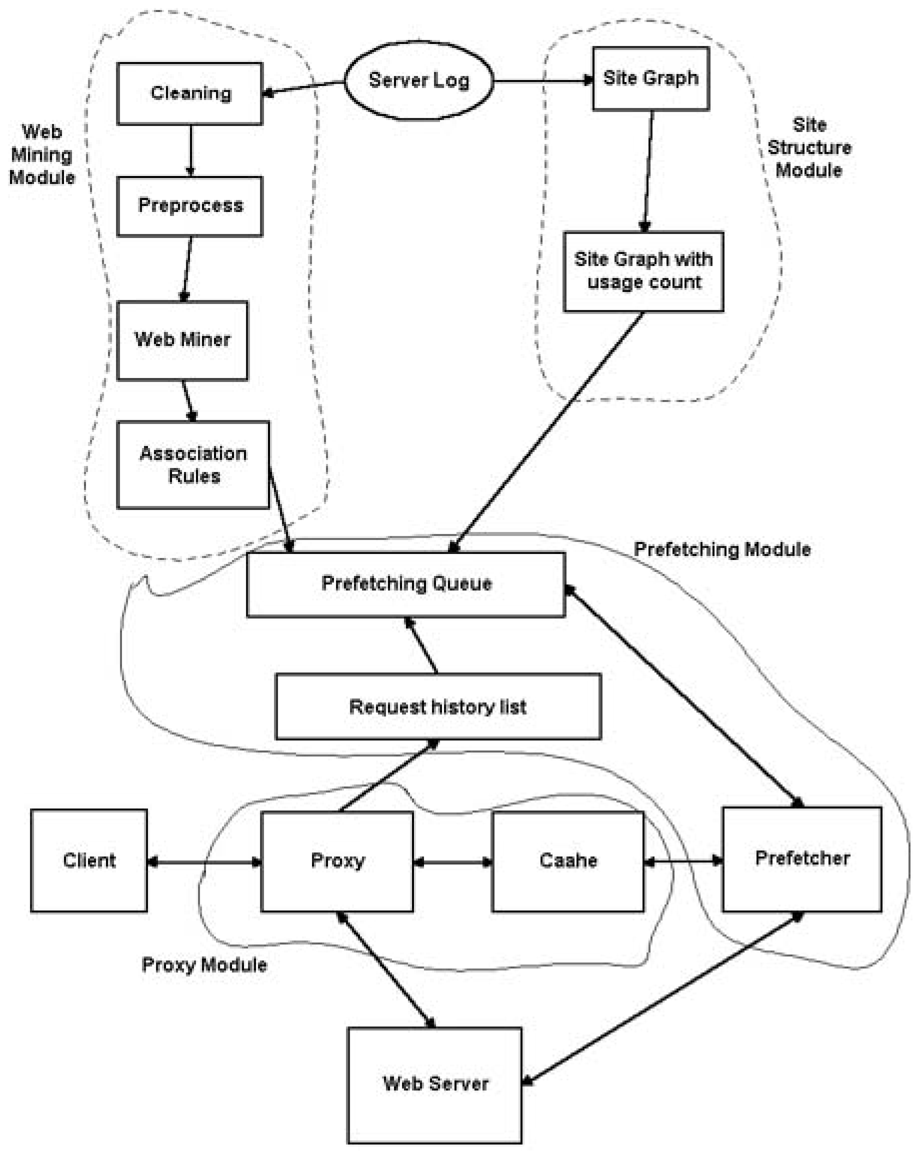


Figure 1: Proxy Server Architecture

We assume that the proxy server has full cooperation of the server and access to server data. The various modules and the sequence of operations are described below

1. In the web-mining module, server logs are filtered and preprocessed to get information in a format required by a web miner for generating association rules. Association rules are generated using the web miner for a range of support and

confidence. This module is run offline at specified intervals to generate up-to - date association rules.

2. The Site structure module contains the site graph. It also maintains usage counts for each node, i.e., page of the web site. The usage count of the nodes is updated offline at specified intervals. It also has the information whether a page is cacheable. According to the http protocol a browser or a proxy server would not cache a dynamic page because a dynamic page comes with a Pragma directive ‘ No Cache ‘ in the header. Since in this case the proxy server has full cooperation of the server, it has the prior knowledge whether a page is going to remain valid during the life of a user session and therefore can be cached by the proxy server regardless of the Pragma directive about caching.
3. A cache is created for each user session and then deleted at the end of the user session. Information about the usage of each page in the cache is kept in a table.
4. The prefetcher module contains a request history list, a prefetching queue, and a prefetching thread. Depending upon the prefetching scheme, a prefetching queue is constructed which maintains a list of the pages which need to be prefetched. The prefetcher thread takes out a URL from the prefetching queue and checks whether the page is in the cache. If the page is not found, it gets the page from the main server and puts it in the cache. The prefetcher thread, the prefetching queue, and the user history list are destroyed at the end of the session.
5. The proxy application module listens to the incoming client requests. After receiving a request, it checks whether the corresponding page is in the cache and, if the page is found, it reads the page from the cache and sends it to the client. If the page is not found in the cache, the proxy server makes a request to the main server, gets the page from the server, and sends it to the client. It also adds the page in the cache if the page is cacheable. It adds the URL of the request in the user history list and updates the prefetching queue on the basis of the algorithm described in Section 4.2.

4.2 Prefetching algorithm

A prefetching queue is maintained in which URLs are added according to the prefetching scheme. There is a request history list which contains all the requests made by the user during the session. In the Site Structure Prefetching scheme, for each URL in the request history list, a list of specified number of children ordered by the usage count is obtained from the site graph and is added in the prefetching queue. The algorithm in the form of a pseudo code for adding to the prefetching queue for the Site Structure Prefetching scheme is given in Figure 2.

```
add_to_site_struct_pref_queue()
{
    url = request_history_list.dequeue();
    if(!url.is_children_prefetched())
    {
        childlist = sitegraph.get_children(url);
        sorted_child_list  = sort_on_usage_count(child_list);
        for( int i = 0;i<n; i++)
        {
                prefetching_que.enqueue(sorted_childlist.elementAt(i);
        }
    }
    url.set_children_prefetch_flag(true);
}
```

Figure 2 : Algorithm for adding elements into site structure prefetching queue

For the Association Rules Prefetching scheme, all association rules whose heads are present in the request history list are identified. The URLs in the body of each identified rule are added to the prefetching queue. The algorithm for adding elements to the prefetching queue for the Association Rule Prefetching scheme is described in Figure 3.

```
add_to_assoc_rule_pref_queue()
{

        rule_list = assoc_rule.get_rule_head_list(request_history_list);
        for (int i = 0; i<rule_list.size(); i++)
        {
                rule  = rule_list.elementAt(i);
                if (!rule.isPrefetched())
                {
                        rule_body_list =
                                assoc_rule.get_rule_body_list(rule);
                        prefetching_queue.enqueue(rule_body_list);
                        rule.set_prefetch_flag(true);
                }

        }

}
```

Figure 3: Algorithm for adding elements into association rule prefetching queue

The procedure for prefetching which is the same for both the schemes is given in Figure 4

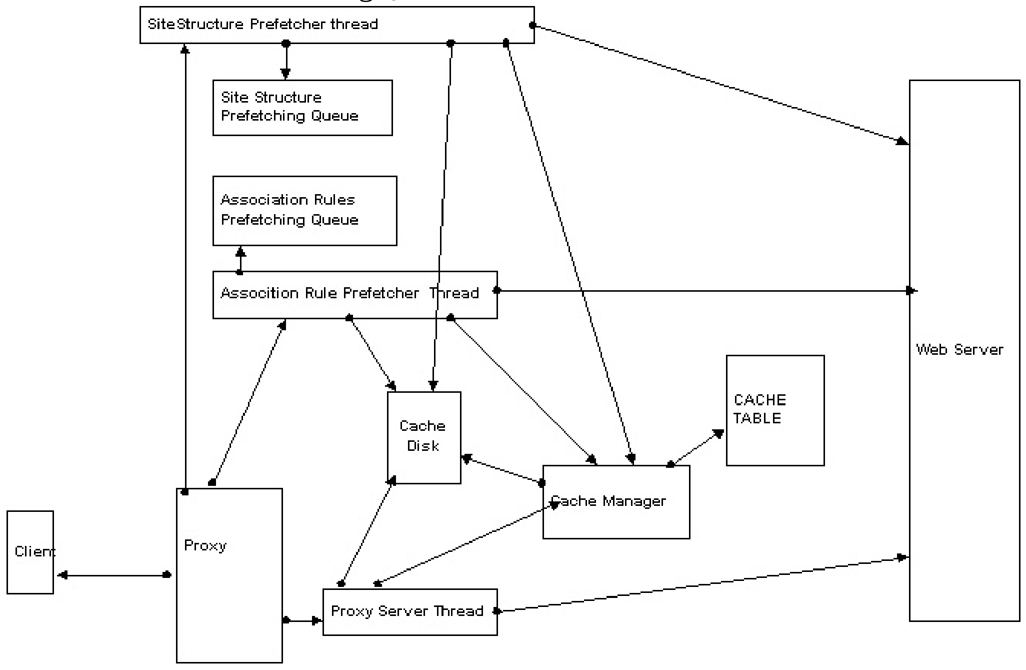
```
prefetch()
{
        while( true)
        {

                if (prefetching_queue.isEmpty())
                {
                        sleep(100 milliseconds);
                }
                else
                {
                        url = prefetching_queue.dequeue();
                        if (!cache.isPresent(url))
                        {
                                fetch_from_server(url);
                                write_into_cache(url);
                        }
                }
        }
}
```

Figure 4: Prefetching procedure

4.3 Implementation

1. Since the primary aim of the implementation was to test the efficacy and feasibility of the proposed models and the architecture of the proxy server, the prototype server was implemented in Java for a single client only. Figure 5 shows the block diagram of the various modules of the Proxy Server.

2. At the start, the main Proxy application instantiates and starts the AssociationRulePrefetcher, SiteStructure Prefetcher, and CacheCleaner threads. It also instantiates CacheManager, AssocRulePrefetchingQueue, and SiteStructurePrefetchingQueue.



SiteStructure Prefetcher thread
Site Structure Prefetching Queue
Association Rules Prefetching Queue
Assocition Rule Prefetcher Thread
Web Server
CACHE TABLE
Cache Disk
Cache Manager
Client
Proxy
Proxy Server Thread

Fig 5: Proxy Server Architecture

3. The proxy creates a server socket by instantiating a Java ServerSocket class on the specified port. The Java server socket class object waits for requests to come over the network and accepts them. For each incoming request, a socket stream , a Java Socket class object, is created. This socket stream has complete information about the client and its request. For each http request, it instantiates and starts a ProxyServer thread by passing the socket stream to its constructor.

4. There is also a session object which keeps track of user sessions through session IDs. In the implementation of the prototype, we assumed that client request would contain a session ID which would uniquely identify a user session. However, in practice, cookies, time stamps, and user agent of the http request can be used to generate session IDs[6].

5. ProxyServer reads and parses the socket stream and extracts out the session ID and the URL of the requested page. It contacts CacheManager to see whether the requested URL is in the Cache.

6. The Cache Manger maintains cache information in an ORACLE table, CACHE_TABLE, whose schema is as follows

URL	**VARCHAR2(200)**
FILENAME	**VARCHAR2(30)**
CNT	**NUMBER(10)**
TIMESTAMP	**NUMBER(20)**
SZ	**NUMBER(10)**

FILENAME specifies the name of the file in which the page corresponding to the URL is stored on the hard disk. CNT denotes the number of times the url has been requested by the client in a given session. TIMESTAMP and SZ denote the time the file was last accessed and the size of the file respectively. CNT, TIMESTAMP and SZ facilitate implementation of the appropriate cache replacement policy. Even though the Cache life is limited to a single session, there is a possibility that a large number of pages may be requested in a single session causing overflow of the cache. Therefore, an appropriate cache replacement policy will be required. Since in this implementation, our primary aim was to test our prediction model with regard to prefetching , we have followed the traditional LRU cache replacement policy.

8. The CacheManager checks in CACHE_TABLE for the URL and ,if found, it returns the FILENAME corresponding to the URL to the requesting thread. If the URL is not found in the table, the CacheManager generates a unique filename corresponding to the URL and returns it to the requesting thread.

9. Whenever a prefetcher thread or the ProxyServer thread fetches a URL from the web server, it informs the Cache Manager, which creates an entry for the corresponding URL in the CACHE_TABLE.

10. Since multiple threads share the CacheManager, for synchronization the CacheManager methods are declared to be Java synchronized methods.

11. If the ProxyServer finds a URL in the Cache, it reads the corresponding file from the disk and writes on to the client socket. It also request the CacheManager to update the CNT and the TIMESTAMP entry in the CACHE_TABLE for the corresponding URL. It then adds the URL to the SiteStructurePrefetching Queue. It also adds the URL in the session list, a member object of the Session object.

12. The Association rules are stored in a text file. Each rule is read from the text file and an object of RuleObj class is instantiated for each rule. The definition of the RuleObj class is :

```
 Class RuleObj
{
        public Vector ruleHead ;
        public Vector ruleBody ;
        public boolean prefetch = false;
 }
```

All RuleObj objects are stored in the vector object Rule.

13. The ProxyServer thread matches the session list with the heads of the association rules and adds the bodies of the matching rules to the AssociationRulesPrefetchingQueue.

14. The AssociationRulesPrefetchertThread constantly polls the Prefetching queue, takes out a url from the queue, and queries the CacheManager about the presence of a URL in the Cache. If the URL is present, the CacheManager returns the corresponding FILENAME; otherwise, it generates a unique filename and returns it to the prefetcher thread. The prefetcher thread then fetches the url from the web server and stores the page on the disk under the filename returned by the CacheManager.

15. The site structure and usage count of each page are maintained in the oracle tables SITE and USAGE_COUNT, respectively. Their schemas are as follows:

SITE TABLE

PARENT	**NUMBER**
CHILD	**NUMBER**

USAGE_COUNT TABLE:

PAGE	**NUMBER**
USE_COUNT	**NUMBER**

The USAGE_COUNT table is updated offline at user- specified intervals.

15. The SiteStructure prefetcher thread constantly polls the SiteStructurePrefetching queue, takes out a URL from the queue, queries the SITE and USAGE_COUNT table to get 'x' number of the children of the URL, ordered by USE_COUNT. The prefetcher thread checks with the CacheManager about the existence of these children in the cache. If they are not in the cache, the prefetcher thread fetches them from the web server, writes on the disk and informs the CacheManager to update the CACHE_TABLE.

16. At the end of the session both the prefetching queues and session list are deleted. All rows of the CACHE_TABLE are also deleted.

17. Logs for the ProxyServer and Prefetcher threads are maintained. They contain the details such as URL fetched, time required for fetching, size of the file, and whether there was a cache hit or not.

5. Experimental Evaluation

5.1 Experiment Design

To evaluate the performance of the proposed models we conducted experiments on the proxy server using the log of the Fingerhut web server (www.fingerhut.com) which was available in sessionized form in an Oracle table by the WebSIFT project[16]. We obtained the log for a day which had 19430 sessions and contained 243,029 requests. We obtained Fingerhut site structure information which too was available in an Oracle table in the WebSIFT project. The Fingerhut site had in all 8,147 pages.

We extracted session IDs and the list of URLs requested in each session from the server log and assigned a unique number to each URL. Half of the one –day's server log was used for mining association rules and training the proxy server, and the other half was used for generating requests to the proxy server during the experiments. We used IBM Data Miner to mine association rules for different levels of support and confidence. While generating rules, we limited maximum rule length to 5.

The proxy server was run on a Sun Sparc 10.The client for sending requests was run on a different machine. Since the main objective of the experiment was to evaluate prefetching models, we did not send actual http requests. Instead, the client sent only a URL number as a request and the proxy server obtained a page http://www.cs.umn.edu from the server of the Computer Science Department if the URL number was not in the cache.

During each experiment, the client sent 3000 requests (approximately 500 user sessions) to the proxy server. We measured the Cache Hit Ratio, which is a measure of latency improvement. Cache Hit Ratio(CHR) is defined as the ratio of number of pages found in the cache and the total number of pages requested. For bandwidth requirement we measured bandwidth ratio, which is the total number of files fetched from the server (i.e., by the Prefetcher thread and the Proxy Server thread) divided by the total number of requests. Ideally while calculating bandwidth ratio, file size should be used. But in these experiments, since for every request the same file is fetched, it does not make a difference whether we used size or number of files.

For measuring the performance of the association rules prefetching model, we varied the minimum support and confidence of the association rules and measured the above two metrics. For the site structure prefetching model, we varied the number of prefetched children and measured the metrics.

5.2 Experiment Results and analysis

5.2.1 Association Rule Prefetching

In this experiment, we obtained association rules from the Fingerhut server log for different levels of minimum support and confidence. Figure 6 shows the relationship between the number of rules and the support for various levels of confidence. As can be seen, the number of rules grows almost exponentially with a decrease in support level. We generated rules for up to minimum support level 10% because, at this level the number of rules had gone up to 7,969 and any further reduction in support level would have made the number of rules prohibitively high. A large rule set adds to the Proxy Server load and affects its performance.

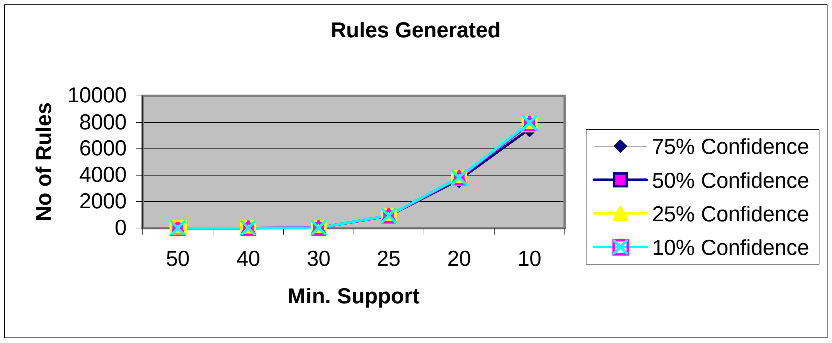


Fig 6: Rule vs. Min Support

The figure also shows that for a specific support level, there is not much difference in the number of rules for confidence in the 75% - 10% range. This implies that for this particular site, rules occur with very high confidence level for most support levels.

Since varying confidence for a given support did not cause any substantial change in rules, we used rules generated with 25 % confidence and different support levels for prefetching in the experiments. The results in the form of graphs, i.e. cache hit ratio vs. min support and bandwidth ratio vs. min support are in shown Figures 7 and 8.

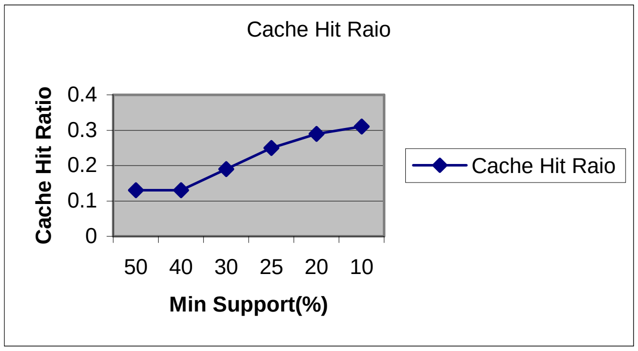


Fig:7 Cache Hit Ratio for ARPS Scheme

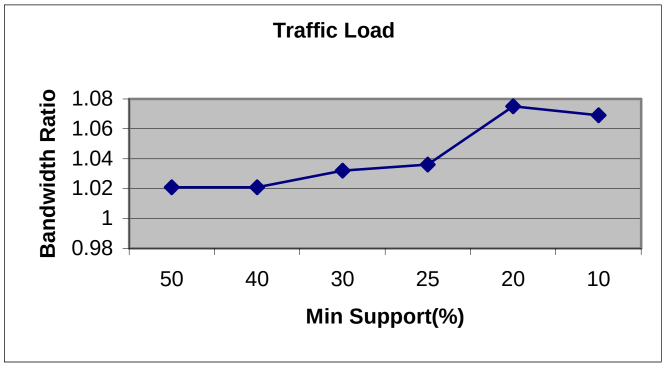


Fig 8: Bandwidth performance for ARPS

We observe in Figure 7 and Figure 8 that Cache Hit Ratio (CHR) increases from 0.13 to 0.33 when minimum support is decreased from 50% to 10%. We also observe that in the lower range of support, i.e., 10-25%, CHR increases at a lesser rate. This implies that aggressive prefetching beyond a certain point starts giving diminishing returns. So far as bandwidth is concerned, though the number of files prefetched increases with decrease in support , the Bandwidth ratio increases only marginally. The reason is that although low support generates a larger number of rules which causes more files to be prefetched , it also increases the CHR, and thus the net effect on the bandwidth is limited.

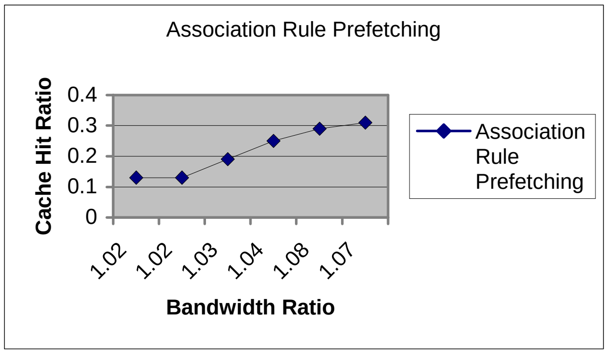


Fig 9: Cache Hit Ratio Vs Bandwidth for ARPS

For clarity, we show the Cache Hit Ratio(CHR) vs. Bandwidth Ratio in Figure 9. We observe that as we increase the Bandwidth Ratio (i.e., aggressive prefetching with larger rules set), we get a higher CHR. We also see that if we continue to increase bandwidth (i.e. more aggressive prefetching), the result is diminishing returns. However, we also observe that we get a 33% CHR at the cost of only an 8% increase in traffic, which shows the effectiveness of the Association Rules Prefetching scheme. These figures would vary for different sites because it depends on how strongly various pages are related to each other and how good the association rules are that we find for those sites.

5.2.2 Site Structure Prefetching:

The results for the Site Structure Prefetching scheme are shown in Figure 10 and Figure 11.

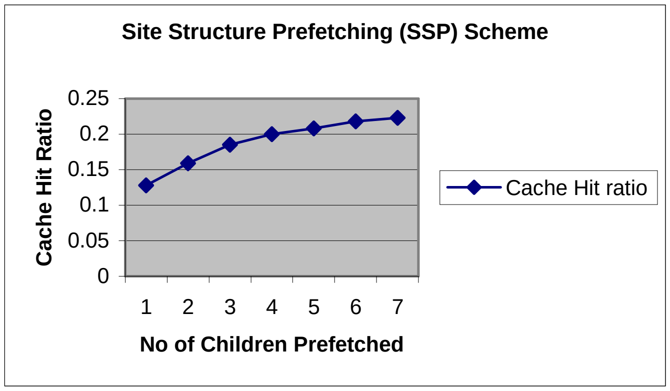


Figure 10: Cache Hit Ratio in SSP Scheme

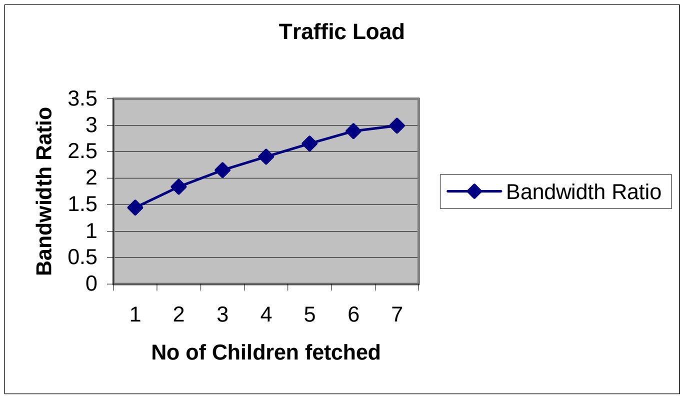


Figure 11: Bandwidth performance in SSP Scheme

As can be seen, CHR increases from 0.13 to 0.23 when the number of prefetched children is increased from one to seven. The bandwidth ratio increases from 1.5 to 3.0 for this range of children. We also observe that though CHR increases with an increase in the number of prefetched children , its rate of increase declines for higher numbers of children showing the law of diminishing returns. Like the Association Rules Prefetching scheme, here also aggressive prefetching does not yield proportionate benefit after a certain point.

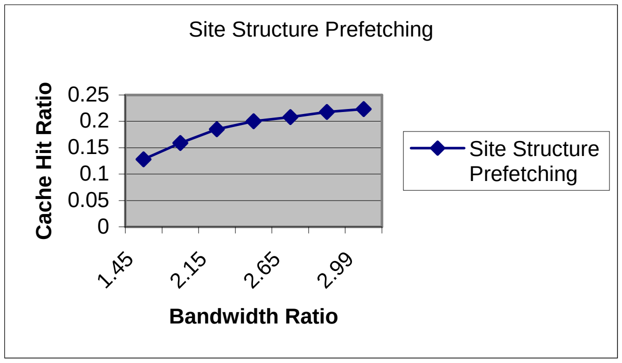


Figure 12: Cache Hit Ratio Vs Bandwidth Ratio for SSP Scheme

We also plotted Cache Hit Ratio(CHR) vs. Bandwidth Ratio and the graph is shown in Figure 12. We see that for a CHR of 0.13 we need a Bandwidth Ratio of 1.45 which means a 45 % increase in traffic load. A CHR of 0.23 requires a Bandwidth Ratio of 2.99, which means a 200% increase in traffic load.

5.2.3 Comparison between the performance of Association Rule Prefetching Scheme and Site Structure Prefetching Scheme

On comparison of Figure 9 and Figure 12, we observe that the Association Rule Prefetching scheme is more efficient than the Site Structure Prefetching scheme. The Association Rule Prefetching scheme gives a higher Cache Hit Ratio with a smaller increase in the bandwidth as compared to the Site Structure Prefetching scheme. The Association Rule Prefetching scheme gives a 33% CHR with only an 8% increase in traffic, whereas the Site Structure Prefetching scheme gives a CHR of 0.23 at the cost of almost a 200 % increase in traffic.

6. Conclusions

We have studied how the knowledge discovered through web mining can be used to improve user perceived latency on the World Wide Web. We presented a prefetching model based on association rules. We have also presented a design and prototype implementation of a proxy server which uses this model to prefetch dynamic pages. Our experiments show that the association rule model has better predictive value than the site structure model. With the association rule model, one can get a cache hit ratio of 33 % with only an 8% increase in traffic load. Even though the association rule model may not be as complete as models based on Markov chain, decision tree or path analysis, building an association rules model is computationally less costly than building other models. Therefore it is computationally cheaper for web usage where there are frequent changes in web site structures and user access patterns.

From our experience in this study, we believe that there are many directions that future could take. One would be to obtain a predictive model based on sequence rules and compare its performance with that of the model based on association rules. Another direction would be to evolve a bench mark for measuring performance of proxy servers based on the Association Rule model, Markov model, and decision tree model, and evaluate their performance on common parameters. Another area of interest would be to build Markov and Decision tree models from the knowledge discovered through association rules and study their performance.

7. Acknowledgements: We acknowledge the assistance of Mukund Deshpande, a PhD student in the Department of Computer Science, University of Minnesota, for his useful suggestions and making available necessary data from the WebSIFT project for the experiments in this study.

8. References

[1] Evangelos P. Markatos and Catherine E. Chironaki. "A Top 10 Approach for Prefetching the Web," In *proceedings of INET'98:Internet Global Summit,* July 1998

[2]Stuart Schechter, Murali Krishnan, and Michael D. Smith. Harvard University and Microsoft. Using Path Profiles to predict http requests. WWW7,1998 http://decweb.ethz.ch/WWW7/1917/com1917.htm

[3] Suyoung Yoon, Eunsook Jin, Jungmin Seo, Ju-Won Song, Multimedia Technology Research Lab, Korea Telecom, http://www.isoc.org/inet99/proceedings/posters/106

[4] Venkata N. Padmanabhan and Jeffrey C. Mogul. "Using Predictive Prefetching to Improve World Wide Web Latency," Computer Communication Review, 26(3):22-36, 1996. http://www.acm.org/sigcomm/ccr/archive/1996/jul96/ccr-9607-mogul-padmanabhan.ps.

[5] Azer Bestavros. “Using Speculation to reduce server Load and service Time on the WWW” , In proceedings of CIKM’95:The Fourth ACM International

[6]Robert Cooley,Mukund Deshpande, Pang-Ning Tan, Jaideep Srivastava. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data[2000], in *SIGKDD Explorations, Vol. 1, Issue 2,* 2000

[7] Thomas M Kroeger, Darell D E Long, Jeffrey Mogul, Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, In Proceedings of the Symposium on Internet Technologies and Systems, Monterey, California,1997

[8] Francesco Bonchi,Fosca Giannotti,Giuseppe Manco,Mirco Nanni,Dino Pedreschi,Chiara Renso, Salvatore Ruggieri, Adaptive Web Caching Using Decision Trees, Technical Report PISA KDD Laboratory

[9] Sarukkai, Ramesh R., Link Prediction and Path analysis Using Markov Chains
9th World Wide Wide Conference, May, 2000. http://www9.org/w9cdrom/68/68.html

[10]Zaiane Osmar,R , Xin, Man, Han, Jiawei, Discovering Web access patterns and trends by applying OLAP and Data Mining Technology on web logs, Proc. Advances in Digital Libraries Conf.(ADL'98),pp:19-29,Santa Barbara,CA,1998.

[11] R.Cooley,B.Mobasher, and J.Srivastava, Web Mining: Information and pattern discovery on the world wide web. In Proceedings of the 9th IEEE International Conference on Tools with artificial Intelligence (ICTAI'97), November 1997.

[12] James Griffion, Randy Appleton "Reducing File System Latency using predictive Approach", Proceedings of 1994 USENIX Technical Conference, Cambridge MA, June 1994

[13] Jiawei Han, Jian Pei, "Mining Frequent Patterns by Pattern-Growth: Methodology and Implications, *ACM SIGKDD Explorations (Special Issue on Scaleable Data Mining Algorithms), December 2000.*

[14] R Agarwal and R Srikant. Fast Algorithms for mining association rules. In proc.1994 Intl. Conf VLDB 1994

[15] J Han, J.Pei and Y Yin, Mining frequent patterns without candidate generation. In proc 2000 ACMSIGMOD Intl Conf. Management of Data, May 2000

[16]WebSift project,Department of Computer Science and Engineering, University of Minnesota,2000