Effective Prediction of Web-user Accesses: A Data Mining Approach

Alexandros Nanopoulos Dimitris Katsaros Yannis Manolopoulos*

Data Engineering Lab, Department of Informatics
Aristotle University, Thessaloniki 54006, Greece
{alex,dimitris,manolopo}@delab.csd.auth.gr

Abstract

The problem of predicting web-user accesses has recently attracted significant attention. Several algorithms have been proposed, which find important applications, like user profiling, recommender systems, web prefetching, design of adaptive web sites, etc. In all these applications the core issue is the developement of an effective prediction algorithm. In this paper, we focus on web-prefetching, because of its importance in reducing user perceived latency present in every Web-based application. The proposed method can be easily extended to the other aforementioned applications.

Prefetching refers to the mechanism of deducing forthcoming page accesses of a client, based on access log information. We examine a method that is based on a new type of association patterns, which differently from existing approaches, considers all the specific characteristics of the Web-user navigation. Experimental results indicate its superiority over existing methods.

Index Terms — Prediction, Web Log Mining, Prefetching, Association Rules, Data Mining.

1 Introduction

The problem of modeling and predicting a user's accesses on a web-site has attracted a lot of research interest. It has been used [18] to improve the web performance through caching [5, 12, 36] and prefetching [31, 20, 32, 25, 34, 35], recommend related pages [17, 33], improve search engines [9] and personalize the browsing in a web site [34].

The core issue in prediction is the development of an effective algorithm that deduces the future user requests. The most successful approach towards this goal has been the exploitation of the user's access history to derive predictions. A thoroughly studied field in this area is Web prefetching. It shares all the characteristics that identify the Web prediction applications like the previously mentioned. Nevertheless, web prefetching presents some particularities involving the underlying communication medium, i.e., the Internet. However, developments in predictive Web prefetching algorithms can be easily adapted to the related topics, e.g., user profiling, recommender systems, design of adaptive web sites, etc.

The objective of prefetching is the reduction of the user perceived latency. Since the Web's popularity resulted in heavy traffic in the Internet, the net effect of this growth was a significant increase in the user perceived latency. Potential sources of latency are the web servers' heavy load, network congestion, low bandwidth, bandwidth underutilization and propagation delay. The obvious solution, that is, to increase the bandwidth, does not seem a viable solution, since the Web's infrastructure (Internet) cannot be easily changed, without significant economic cost. Moreover, propagation delay cannot be reduced beyond a certain point, since it depends on the physical distance between the communicating end points. The caching of web documents at various points in the network (client, proxy, server [5, 12, 36]) has been developed to reduce the latency. Nevertheless, the benefits reaped due to caching can be limited [24], when Web resources tend to change very frequently, resources cannot be cached (dynamically generated web documents), they contain cookies and when request streams do not exhibit high temporal locality.

On the other hand, prefetching refers to the process of deducing client's future requests for Web objects and getting that objects into the cache, in the background, before an explicit request is made for them. Prefetching capitalizes on the spatial local-

*Contact author. email: manolopo@csd.auth.gr, tel: +3031 996363, fax: +3031998419

ity present in request streams [1], that is, correlated references for different documents, and exploits the client's idle time, i.e., the time between successive requests. The main advantages of employing prefetching is that it prevents bandwidth underutilization and hides part of the latency. Nevertheless, an over-aggressive scheme may cause excessive network traffic. Additionally, without a carefully designed prefetching scheme, several transferred documents may not be used by the client at all, thus wasting bandwidth. Nevertheless, an effective prefetching scheme, combined with a transport rate control mechanism, can shape the network traffic, reducing significantly its burstiness and thus improve network performance [11].

In general, there exist two prefetching approaches. Either the client will inform the system about its future requirements [30] or, in a more automated manner and transparently to the client, the system will make predictions based on the sequence of the client's past references [14, 31]. The first approach is characterized as *informed* and the latter as *predictive* prefetching. In the design of a prefetching scheme for the Web, its specialties must be taken into account. Two characteristics seem to heavily affect such a design: a) the client server paradigm of computing the Web implements, b) its hypertextual nature. Therefore, informed prefetching seems inapplicable in the Web, since a user does not know in advance its future requirements, due to the "navigation" from page to page by following the hypertext links.

1.1 Mechanism of Predictive Prefetching

Web servers are in better position in making predictions about future references, since they log a significant[1] part of requests by all Internet clients for the resources they own. The prediction engine can be implemented by exchange of messages between the server and clients, having the server piggybacking information about the predicted resources onto regular response messages, avoiding establishment of any new TCP connections [13]. Such a mechanism has been implemented in [13, 19] and seems the most appropriate, since it requires relatively few enhancements to the current request-response protocol and no changes to HTTP 1.1 protocol.

In what follows in this article, we assume that there is a system implementing a server-based predictive prefetcher, which piggybacks its predictions as hints to its clients. Figure 1 illustrates how such an enhanced Web server could cooperate with a prefetch engine to disseminate hints every time a client requests a document of the server.

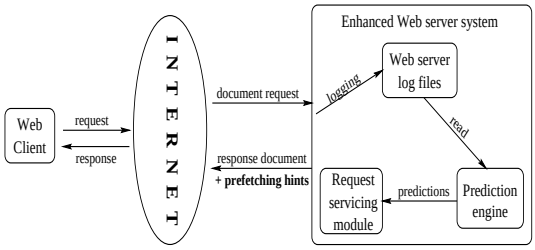

Figure 1: Proposed architecture of a prediction-enabled Web server.

1.2 Paper contribution

In this paper we focus on predictive prefetching. First, we identify three factors: a) the order of dependencies between page accesses, b) the noise present in user access sequences due to random accesses of a user (i.e., access that are not part of a pattern), c) the ordering[2] of accesses within access sequences, that characterize the performance of predictive web prefetching algorithms. We develop a new algorithm that takes into account the above factors. An extensive experimental comparison of all algorithms, for the first time (the performance of existing algorithms have been examined only independently), indicates that the proposed algorithm outperforms existing ones, by combining their advantages without presenting their deficiencies.

The rest of the paper is organized as follows. Section 2 reviews related work and outlines the motivation of this work and Section 3 describes the proposed algorithm. Section 4 provides the experimental results and finally, Section 5 contains the conclusions.

2 Related Work

Research on predictive web prefetching has involved the important issue of log file processing and the determination of user transactions (sessions) from it[3]. Several approaches have been proposed towards this direction [15, 16]. Since it is a necessary step for every web prefetching method, more or less similar approaches on transaction formation from log files have

[1] They only miss the requests satisfied by browser or proxy caches.

[2] The term *ordering* refers to the arrangment of accesses in a sequence, whereas *order* to the dependencies among them.

[3] Notice that this issue is not required for prefetching in the context of file systems.

been proposed in [31, 32, 25]. However, the most important factor for any web prefetching scheme is the prediction algorithm, which is used to determine the actual documents to be prefetched.

The prediction scheme described in [31] uses a prefetching algorithm proposed for prefetching in the context of file systems [21]. It constructs a data structure, called *Dependency Graph* (DG), which maintains the pattern of access to different documents stored at the server. The left part of Figure 2 illustrates an example of a dependency graph. For a complete description of the scheme see [21, 31]. The work described in [7] uses essentially the dependency graph, but makes predictions by computing the transitive closure of this graph. This method was tested and did not showed significantly better results compared to the simple dependency graph.

The scheme described in [32, 20] also uses a prefetching algorithm from the context of file systems [14]. It is based on the notion of an m-order *Prediction-by-Partial-Match* (PPM) predictor. An m-order PPM predictor maintains Markov predictors of order j, for all $1 \leq j \leq m$. This scheme is also called *All-m^{th}-Order Markov model* [18]. The right part of Figure 2 illustrates a 2nd order PPM predictor, where paths emanate from the tree root with maximum length equal to $m+1$ (=3). For a complete description of the scheme see [14, 23, 32, 20].

Recently, several algorithms have been proposed for mining patterns from web logs [16, 8, 10, 15, 29, 27]. Although these patterns can be characterized as descriptive, since they indicate regularities discovered from user access information, algorithms for web log mining and for predictive web prefetching share the common objective of determining statistically significant user access sequences, i.e., *access patterns*. The web prefetching strategy proposed in [25] develops a specialized association rule mining algorithm to discover the prefetched documents. It discovers dependencies between pairs of documents (association rules with one item in the head and one item in the body).

Other related work includes [26], which describes a prefetching algorithm that is also based on association rule mining. However, the subject of this paper is Web-server caching, and more particularly the prefetching of documents from the Web server's disk to its main memory. This approach differs from web prefetching, which concerns the prefetching of documents from the server into the client's cache. Improvements on the efficiency of PPM is examined in [18] (which uses the name All-mth-Order Markov model for PPM). Three pruning criteria are proposed: a) support-pruning, b)confidence-pruning, c) error-pruning. The subject of [18] is mainly the efficiency (experimental results indicated only marginal improvements in the effectiveness of PPM). Nevertheless, support-pruning is also examined in [25, 26] and in this paper as well. The other two criteria are used in a post-processing step, on the set of discovered rules, and can be applied to any prefetching scheme, thus they are orthogonal issues to the subject examined in this paper. Finally, two variations of the PPM prefetcher are described in [34, 35]. The first one is a subset of the PPM whereas in the second one the selection of prefetching rules to activate is determined by "weights" assigned on them.

2.1 Motivation

Most of the existing web prefetching schemes differ from the corresponding ones proposed in the context of file systems only because they use techniques for the identification of user transactions. For the core issue in prefetching, i.e., prediction of requests, existing algorithms from the context of file-systems have been utilized. Consequently, existing web prefetching algorithms do not recognize the specialized characteristics of the web. More precisely, two important factors are:

- The order of dependencies among the documents of the patterns.
- The interleaving of documents belonging to patterns with random visits within user transactions.

These factors arise from both the contents of the documents and the site of the structure (the links among documents), and are described as follows.

The choice of forthcoming pages can depend, in general, on a number of previously visited pages. This is also described in [18]. The *DG*, the 1-order *PPM* and the scheme in [25] consider only first order dependencies. Thus, if several visits have to be considered and there exist patterns corresponding to higher order dependencies, these algorithms do not take them into account in making their predictions. On the other hand, the higher-order *PPM* algorithms use a constant maximum value for the considered orders. However, no method for the determination of the maximum order is provided in [32, 20]. A choice of a small maximum may have a similar disadvantage as in the former case, whereas a choice of a large maximum may lead to unnecessary computational cost, due to maintenance of a large number of rules.

A web user may follow, within a session, links to pages that belong to one of several patterns. However, during the same session, the user may also navigate to other pages that do not belong to this pattern

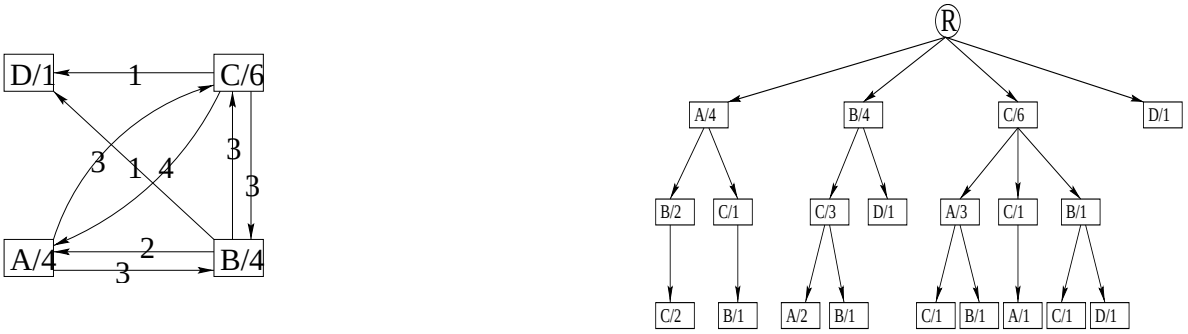

Figure 2: *Left:* Dependency graph (lookahead window 2) for two request streams ABCACBD and CCABCBCA. *Right:* PPM predictor of 2nd order for two request streams ABCACBD and CCABCBCA.

(or that may not belong to any pattern at all). Hence, a user transaction can both contain documents belonging to patterns and others that do not, and these documents are interleaved in the transaction. However, *PPM* (of first or higher order) and [25] prefetchers consider only subsequences of consecutive documents inside transactions. On the other hand, the *DG* algorithm does not require the documents, which comprise patterns, to be consecutive in the transactions. However, since the order of the *DG* algorithm is one, only subsequences with two pages (not necessarily consecutive in the transactions) are considered[4].

Consequently, none of the existing algorithms considers all the previously stated factors. The objective of this paper is the development of a new algorithm that considers all the factors and subsequently the evaluation of their significance.

3 Proposed Method

3.1 Type of prefetching rules

Associations consider rules of several orders [3] (not of one only). The maximum order is derived from the data and it does not have to be specified as an arbitrary constant value [3]. For the support counting, a transaction T, supports sequences that do not necessarily contain consecutive documents in T. However, although the ordering of document accesses inside a transaction is important for the purpose of prefetching, it is ignored by the association rules mining algorithms [3]. The approach in [16] takes into account the ordering within access sequences, however, similar to PPM algorithm [32], it considers only subsequences with consecutive accesses within transactions.

The required approach involves a new definition of the candidate generation procedure and the *containment* criterion (for the support counting procedure). At the k-th phase, the candidates are derived from the self-join $L_{k-1} \bowtie L_{k-1}$ [3]. However, to take the ordering of documents into account, the joining is done as follows. Let two access sequences be $S_1 = \langle p_1, \ldots, p_{k-1} \rangle$ and $S_2 = \langle q_1, \ldots, q_{k-1} \rangle$, both in L_{k-1}. If $p_1 = q_1, \ldots, p_{k-2} = q_{k-2}$, then they are combined to form two candidate sequences, which are: $c_1 = \langle p_1, \ldots, p_{k-2}, p_{k-1}, q_{k-1} \rangle$ and $c_2 = \langle p_1, \ldots, p_{k-2}, q_{k-1}, p_{k-1} \rangle$ (i.e., c_1 and c_2 are not considered as identical, as in [3]). For instance, sequences $\langle A, B, C \rangle$ and $\langle A, B, D \rangle$ are combined to produce $\langle A, B, C, D \rangle$ and $\langle A, B, D, C \rangle$. The same holds for the second phase ($k = 2$). For instance, from $\langle A \rangle$ and $\langle B \rangle$, $\langle A, B \rangle$ and $\langle B, A \rangle$ are produced. The *containment* criterion is defined as follows:

Definition 1 *If $T = \langle p_1, \ldots, p_n \rangle$ is a transaction, an access sequence $S = \langle p'_1, \ldots, p'_m \rangle$ is contained by T iff:*

- *there exist integers $1 \leq i_1 < \ldots < i_m \leq n$ such that $p'_k = p_{i_k}$, for all k, where $1 \leq k \leq m$.* □

A sequence, S, of documents contained in a transaction, T, with respect to Definition 1 is called a *subsequence* of T and the containment is denoted as $S \preceq T$.

Related to the above scheme are the ones presented in [27, 28]. By considering *user-sessions* as *customer-sequences*, the problem can also be transformed to the setting of [4] for mining sequential patterns (each request corresponds to a customer transaction). Nevertheless, the special phases of [4] for the processing of customer-transactions (i.e., elements of customer-sequences) are not applicable, since the latter are of

[4]It has to be noticed that the use of thresholds for statistical significance, e.g., support values [3], does not address the intervening of random accesses within the patterns. The patterns cannot be retrieved with larger values of support, since they will still remain broken to pieces. The problem is addressed with appropriate candidate generation procedure (c.f., Section 3).

length one when considering user-sessions. Therefore, the direct application of the approach in [4] is not efficient. The scheme proposed in [22] uses a similar approach, called mining of path fragments. The main objective of path fragments is to consider subsequences with non-consecutive accesses within transactions (for handling noise), i.e., the issue that is addressed by Definition 1. The method in [22] is based on discovering patterns containing regular expressions with the $*$ wild-card between accesses of a sequence.

Although the use of wild-cards presents differences in a semantic level (it may distinguishes the sequences that explicitly do not contain consecutive accesses), for the purpose of web-prefetching, the use of Definition 1 assures the addressing of noise within transactions without the need for wild-cards. Path fragments [22] where only examined against the approach in [16] (subsequences with consecutive accesses). However, [4, 27] had addressed the handling of subsequences with non-consecutive accesses[5]. Moreover, the use of wild-cards requires the modification of the frequency counting procedure. The candidate-trie [3] should store, additionally to ordinary candidates, the ones containing wild-cards. Consequently, a significant space and time overhead (since the wild-cards may appear in a number of combinations that grows rapidly with the size of candidates). Also during the probing of a trie by a transaction, at each trie-node with the wild-card $*$, every child of the node has to be followed. This increases the time complexity of support counting. However, [22] does not present any method for the support counting phase to address the above issues, and no experimental results are provided to examine its performance.

In the sequel, the web prefetching scheme that is based on the type of rules determined by Definition 1 is denoted as WM_o (o stands for ordered web mining).

3.2 Algorithm

The candidate generation procedure of WM_o, due to the preservation of ordering, impacts the number of candidates. For instance, for two "large" documents $\langle A\rangle$ and $\langle B\rangle$, both candidates $\langle A, B\rangle$ and $\langle B, A\rangle$ will be generated in the second phase (differently, Apriori [3] would produce only one candidate, i.e., $\{A, B\}$). The same holds for candidates with larger length, for each of which the number of different permutations is large (nevertheless, ordering has to be preserved to provide correct prefetching).

[5]In [22] the approaches in [4] and [16] are treated as identical, despite the difference that [4] handles subsequences with non-consecutive accesses.

In order to reduce the number of candidates, pruning can be applied according to the site of the structure [27]. This type of pruning is based on the assumption that navigation is performed following the hypertext links of the site, which form a directed graph. Therefore, an access sequence, and thus a candidate, has to correspond to a path in this graph. The candidate generation procedure and the apriori-pruning criterion [3] have to modified appropriately, and are depicted bellow.

Algorithm genCandidates(L_k, G)
// L_k the set of large k-paths and G the graph
begin
foreach $L = \langle \ell_1, \ldots, \ell_k\rangle$, $L \in L_k$ {
  $N^+(\ell_k) = \{v | \exists$ arc $\ell_k \rightarrow v \in G\}$
  foreach $v \in N^+(\ell_k)$ {
    //apply modified apriori-pruning
    if $v \notin L$ **and** $L' = \langle \ell_2, \ldots, \ell_k, v\rangle \in L_k$ {
      $C = \langle \ell_1, \ldots, \ell_k, v\rangle$
      if $(\forall S \preceq C, S \neq L' \Rightarrow S \in L_k)$
        insert C in the candidate-trie
    }
  }
}
end

For the example depicted in Figure 3a, candidate $\langle B, E, C\rangle$ corresponds to a path in the graph. On the other hand, candidate $\langle B, C, E\rangle$ does not, thus it can be pruned. The reason of pruning the later candidate is that no user transaction will contain $\langle B, C, E\rangle$, since there are no links to produce such an access sequence that will contain (according to Definition 1) this candidate and increase its support. For instance, assuming the example database of Figure 3b, candidate $\langle B, E, C\rangle$ will have support equal to two (contained in first and fourth transaction), whereas candidate $\langle B, C, E\rangle$ is not supported by any transaction. The same applies for the example presented earlier in this section. Among candidates $\langle A, B\rangle$ and $\langle B, A\rangle$, the former can be pruned. Evidently, $\langle B, A\rangle$ is contained in three transactions, whereas $\langle A, B\rangle$ in none. Containment is tested with respect to Definition 1. For example, candidate $\langle B, A\rangle$ (which corresponds to a path) is contained in the third transaction, i.e., $\langle B, D, A\rangle$, although documents B and A are not consecutive in the transaction.

As it is illustrated, instead of generating every possible candidate and then pruning with respect to the site structure, a much more efficient approach is followed. More particularly, candidate generation, candidate storage and representation, and support counting over the transactions database take into account the structure of the site during all these steps

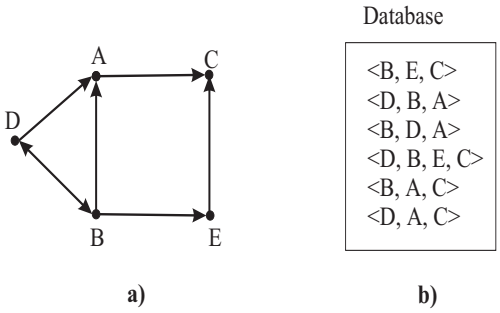

Figure 3: *a:* An example of site structure. *b:* A database of access sequences.

and not in a post-processing one, thus improving the efficiency. As it is shown, candidate generation is performed by extending candidates according to their outgoing edges in the graph, thus with respect to the site structure. Consequently, the ordering is preserved and, moreover, only paths in the graph are considered. Additionally, the apriori-pruning criterion of [3] is modified, since, for a given candidate, only its subsequences have to be tested and not any arbitrary subset of documents, as it is designated for basket data [3]. Candidates are stored in a trie structure. Each transaction that is read from the database is decomposed into the paths it contains and each of them is examined against the trie, thus updating the support of the candidates. Due to space restrictions, more details can be found in [27, 28].

The overall execution time required for the support counting procedure is significantly affected by the number of candidates [3], hence its efficiency is improved by this pruning criterion. Although several heuristics have been proposed for the reduction of the number of candidates for the Apriori algorithm, they involve basket data. The pruning with respect to the site structure is required for the particular problem, due to the ordering preservation and the large increase in the number of candidates. The effectiveness of pruning is verified by experimental results in Section 4.

4 Performance Results

This section presents the experimental results on the performance of predictive web prefetching algorithms. We focus on *DG, PPM, LBOT*[6]*, WM* (denoting the plain association rules mining algorithm [3]) and WM_o algorithms. Both synthetic and real data were used. The performance measures used are the following (their description can be found also in [32]): *Usefulness* (also called Recall or Coverage): the fraction of requests provided by the prefetcher. *Accuracy* (also called Precision): the fraction of the prefetched requests offered to the client, that were actually used. *Network traffic*: the number of documents that the clients get when prefetching is used divided by the one when prefetching is not used. First, we briefly describe the synthetic data generator. Then, we present the results, and finally we provide a discussion.

4.1 Generation of Synthetic Workloads

In order to evaluate the performance of the algorithms over a large range of data characteristics, we generated synthetic workloads. Each workload is a set of transactions. Our data generator implements a model for the documents and the linkage of the Web site, as well as a model for user transactions.

We choose so that all site documents have links to other documents, that is, they correspond to HTML documents. The fanout of each node, that is, the number of its outgoing links to other nodes of the same site, is a random variable uniformly distributed in the interval [1..*NFanout*], where *NFanout* is a parameter for the model. The target nodes of these links are uniformly selected from the site nodes. If some nodes have no incoming links after the termination of the procedure, then they are linked to the node with the greatest fanout. With respect to document sizes, following the model proposed in [6], we set the maximum size equal to 133KB and assign sizes drawn from a lognormal distribution[7] with mean value equal to 9.357KB and variance equal to 1.318KB.

In simulating user transactions, we generated a pool of P paths ("pattern paths", in the sequel). Each path is a sequence of linked in the site and pairwise distinct Web server documents, and will be used as "seeds" for generating the transactions. Each of these paths is comprised of 4 nodes (documents), simulating the minimum length of a transaction. The paths are created in groups. Each group comprises a tree. The paths are actually the full length paths found in these trees. The fanout of the internal tree nodes is controlled by the parameter bf. Varying this parameter we are able to control the 'interweaving' of the paths. The nodes of these trees are selected using either the 80-20 fractal law or from the nodes

[6]In the experiments the algorithm proposed in [25] will be referenced as *LBOT*

[7]Without loss of generality, we assume that HTML files are small files. Thus, according to [6] their sizes follow a lognormal distribution.

that were used in the trees created so far. The percentage of these nodes is controlled by the parameter *order*, which determines the percentage of node dependencies that are non-first order dependencies. For example, 60% order means that 60% of the dependencies are non-first order dependencies. Thus, varying this parameter, we can control the order of the dependencies between the nodes in the path. The use of the fractal law results in some nodes to be selected more frequently than others. This fact reflects the different popularity of the site documents, creating the so-called "hot" documents.

In order to create the transactions, we first associate a weight with each path in the pool. This weight corresponds to the probability that this path will be picked as the "seed" for a transaction. This weight is picked from an exponential distribution with unit mean, and is then normalized so that the sum of the weights for all the paths equals 1. A transaction is created as follows. First, we pick a path, say $\langle A, B, C, x \rangle$, tossing a P-sided weighted coin, where the weight for a side is the probability of picking the associated path. Then, starting from node A we try to find a path leading to node B or with probability $corProb$ to node C, whose length is determined by a random variable, following a lognormal distribution, whose mean and variance are parameters of the model. This procedure is repeated for every node of the initial path except from those that, with probability $corProb$, were excluded from the path. The mean and variance of the lognormal distribution determine the "noise" inserted in each transaction. Low values for mean and variance leave the transaction practically unchanged with respect to its pattern path, whereas larger values increase its length with respect to the pattern path. Table 1 summarizes the parameters of the generator.

N	Number of site nodes
$NFanout$	Max num of nodes' links
T	Number of transactions
P	Number of pattern paths
bf	Branching factor of the trees
$order$	Order of the dependencies
$noiseMean$	Mean value of the noise
$noiseVar$	Variance of the noise
$corProb$	Prob. excluding a path pattern

Table 1: The parameters for the generator.

4.2 Comparison of all algorithms

In order to carry out the experiments we generated a number of workloads. Each workload consisted of T=100,000 transactions. From these, 35,000 transactions were used to train the algorithms and the rest to evaluate their performance. The number of documents of the site for all workloads was fixed to N=1000 and the maximum fanout to $NFanout$=100, so as to simulate a dense site. The branching factor was set to bf=4 to simulate relatively low correlation between the paths. The number of paths of the pool for all workloads was fixed to P=1000. With several experiments, not shown in this report, it was found that varying the values of the parameters P and N does not affect the relative performance of the considered algorithms. For all the experiments presented here, the *order* of the *PPM* algorithm was set equal to 5, so as to capture both low and higher order dependencies. For the experiments, the *lookahead window* of the DG algorithm was set equal to the length of the processed transaction, in order to be fair with respect to the other three algorithms. Also, in order to decouple the performance of the algorithms from the interference of the cache, we flushed it after the completion of each transaction. Each measurement in the figures that follow is the average of 5 different runs. At this point, we must note that the used performance measures are not independent. For example, there is a strong dependence between usefulness and network traffic. An increase in the former cannot be achieved without an analogous increase in the latter. This characteristic is very important for the interpretation of the figures to be presented in the sequel. In order to make safe judgments about the relative performance of the algorithms, we must always examine the two measures, while keeping fixed the value of the third metric, usually the network traffic, for all the considered algorithms.

The first set of experiments assessed the impact of noise on the performance of the prediction schemes. For this set of experiments, we selected a confidence value such that each scheme incurred about 50% overhead in network traffic (i.e., all methods were normalized to the same network traffic, which is the measure of resource utilization – the same applies also to the forthcoming experiments). The confidence value for DG and $LBOT$ was set equal to 0.10, for PPM equal to 0.25 and for the other two methods equal to 0.275 (for WM_o and WM the support threshold was set to 0.1%, which used as the default value). The results of this set of experiments are reported in the left part of Figure 4. From these figures, it can be seen that WM_o clearly outperforms all algorithms in terms of

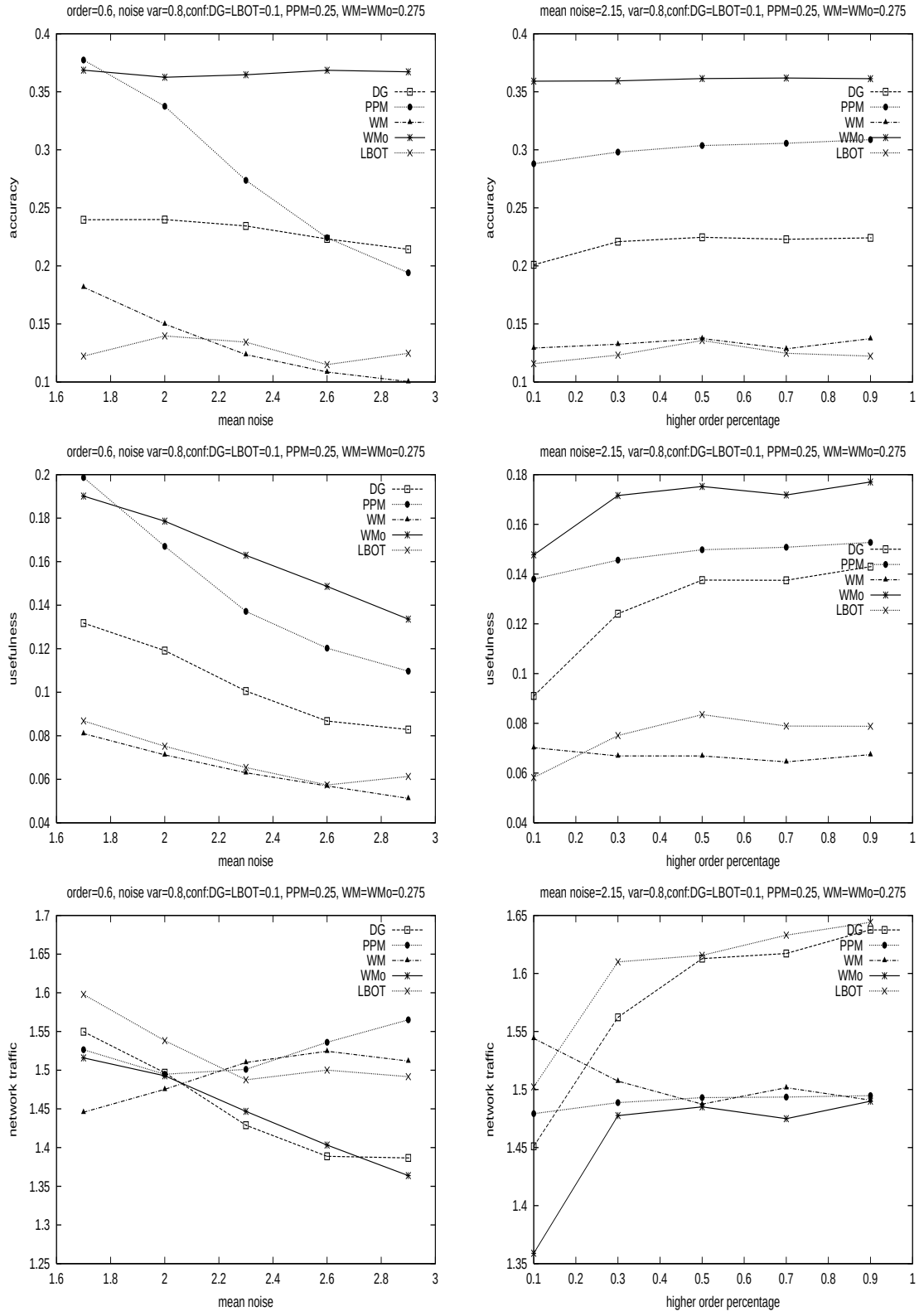

Figure 4: Performance as a function of noise (left) and *order* (right).

accuracy and usefulness for the same network traffic. It is better than *PPM* as much as 40% and than *DG* as much as 100%, in terms of accuracy. *PPM* is better than WM_o for small values of noise, but it achieves this in the expense of larger network traffic. It can also be seen that the accuracy of WM_o is not affected by the increasing noise at all, implying that WM_o makes correct predictions even at the presence of high noise. The usefulness for all algorithms drops with noise. In particular, the usefulness of *PPM* drops more steeply for noise mean values of up to 2.3. Beyond this value, usefulness for *PPM* seems to be stabilized, but this is achieved in expense of higher network traffic.

The second set of experiments evaluated the impact of the varying *order* on the performance of the methods. For this set of experiments, the confidence value for each method was the same as in the last set, whereas the mean value and variance of noise was set to 2.15 and 0.8, respectively. The results of this set of experiments are reported in the right part of Figure 4. The general result is that only *DG* and *LBOT* are affected from the varying *order*, since in order to keep its usefulness and accuracy in the same values, they increase their network traffic. The rest of the algorithms seem insensitive to the varying *order* with WM_o performing the best among them, in terms of both accuracy and usefulness.

Next, we evaluated the benefits of the prefetching algorithms for an LRU cache and compared it with the performance of the same cache with no prefetching at all. For this experiment the range of cache size was selected to be in the range of a few hundred KBytes, to simulate the fact that not all, but only a small part of the Web client cache is "dedicated" to the Web server documents. The confidence was set so that all algorithms incur network traffic 150%. The results of this experiment are reported in Figure 5. From this figure, it is clear that prefetching is beneficial, helping a cache to improve its hit ratio as much as 50%. The figure shows that the hit ratio increases steadily. This is due to the fact that, when the cache size becomes large enough to hold all the site documents, then any future reference will be satisfied by the cache and its hit ratio will approach 100%. The same figure shows that interference due to cache does not "blur" the relative performance of the prefetching algorithms. Therefore, WM_o outperforms all other algorithms. The performance gap would be wider in environments with higher noise and higher order of dependencies between the accesses. For small cache sizes, WM_o, *PPM* and *DG* have similar performance because for these sizes, the cache is not large enough to hold all prefetched documents, and thus many of them are replaced before they can be used. On the other hand, for very large cache sizes, the performance of all algorithms converges, since almost all the site documents are cached, as explained before.

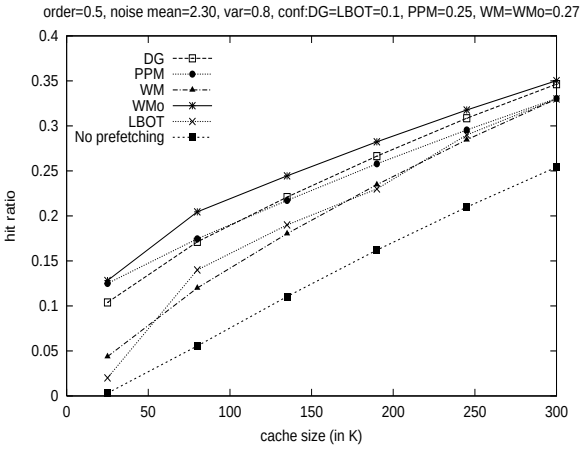

Figure 5: Cache hits as a function of the cache size.

4.3 Real Datasets

We conclude the evaluation of the examined prefetching algorithms by presenting some experiments that were conducted using real web server traces. In the following, due to space limitations, we present the results obtained from one trace, namely the *ClarkNet* (available at http://ita.ee.lbl.gov/html/traces.html). We used the first week of requests and we cleansed the log (e.g., by removing CGI scripts, staled requests, etc.). The user session time was set to 6 hours and 75% of the resulted transactions were used for training. The average transaction length was 7, with the majority of the transactions having length smaller than 5, so we set the order of the *PPM* prefetcher to 5 and the lookahead window of the *DG* prefetcher to 5. For WM_o we turned off the structure-based pruning criterion, since the site structure for this dataset is not provided (however, for a real case application, the site structure is easily obtainable).

Table 2 presents the results from this experiment. The measurements where made so that the network traffic incurred was the same for all algorithms. As it is illustrated, WM_o achieves better performance than all the other algorithms, in all cases, in terms of accuracy and usefulness. This also verifies the performance results obtained from synthetic data. It has to be noticed that publicly available log files, like the *ClarkNet* (this log is from year 1995), are not characterized by high connectivity and many alternative navigation possibilities, which impact the num-

	DG (w=5)		LBOT		5-order PPM		WM		WM_o	
T	**A**	**U**	**A**	**U**	**A**	**U**	**A**	**U**	**A**	**U**
1.9	0.13	0.12	0.08	0.075	0.19	0.17	0.07	0.06	0.20	0.18
1.8	0.16	0.13	0.09	0.072	0.20	0.17	0.05	0.04	0.22	0.17
1.7	0.19	0.13	0.10	0.070	0.22	0.14	0.06	0.04	0.24	0.17
1.6	0.16	0.11	0.05	0.035	0.23	0.13	0.07	0.04	0.27	0.17
1.5	0.11	0.06	0.06	0.032	0.25	0.12	0.08	0.04	0.29	0.15

Table 2: Comparison of prefetchers with real data

ber and the characteristics of the navigation patterns. Thus, synthetic data can better simulate the latter case, for which the performance gain obtained due to prefetching is more significant, because this kind of applications are high demanding (e.g., e-commerce sites).

4.4 Efficiency of WM_o

Finally, we examined the effectiveness of the proposed pruning criterion. We use a synthetic dataset with the same characteristics as the ones used in the experiments of Section 4.2. This experiment compares the proposed WM_o algorithm with a version that does not use pruning with respect to the site structure, and is denoted as $WM_{o/wp}$ (WM_o without pruning). Moreover, we examined the WM algorithm (Apriori algorithm for basket data), in order to provide a comparatively evaluation of the result. Figure 6 illustrates the number of candidates for these methods with respect to the support threshold (given as a percentage). As it is depicted, WM_o significantly outperforms both $WM_{o/wp}$ and WM for lower support thresholds, where the number of candidates is larger. For larger support thresholds, WM_o still outperforms $WM_{o/wp}$ and presents a comparative number of candidates with those produced by WM. Nevertheless, as shown by the previous experiments, the performance of WM for the overall prefetching purpose is significantly lower than that of WM_o.

The number of candidates significantly impacts the performance of this type of algorithms. This is in accordance with related work on association rule mining [3]. Therefore, the efficiency of WM_o is improved by the proposed pruning. Moreover, pruning is efficiently applied by considering the site structure in every step of the rule discovery algorithm (see Section 3.2). Detailed experimental results on the execution time of this procedure can been found in [28].

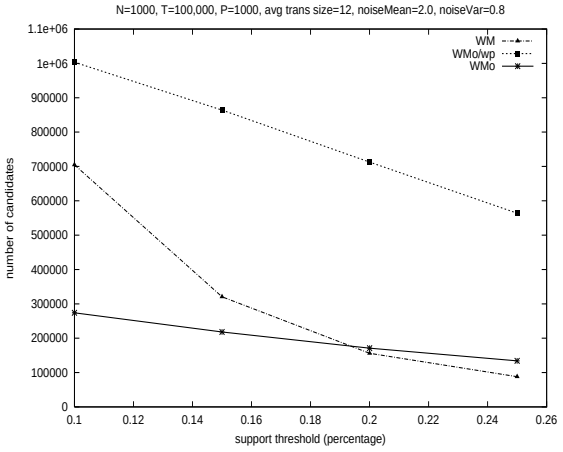

Figure 6: Number of candidates w.r.t. support threshold

5 Conclusions

We considered the problem of predictive Web prefetching, that is, of deriving users' future requests for Web documents based on their previous requests. Predictive prefetching suits the Web's hypertextual nature and reduces significantly the perceived latency. Web prefetching has common characteristics with other Web applications that involve prediction of user accesses, like user profiling, recommender systems and design of adaptive web sites. Hence, the proposed method can be easily extended to these kind of applications.

We presented the important factors that affect the performance of Web prefetching algorithms. The first factor is the order of dependencies between Web document accesses. The second is the interleaving of requests belonging to patterns with random ones within user transactions and the third one is the ordering of requests. None of the existing approaches has considered the aforementioned factors altogether.

We proposed a new algorithm called WM_o, whose characteristics include all the above factors. It compares favorably with previously proposed algorithms,

like *PPM*, *DG* and existing approaches from the application of Web log mining to Web prefetching. Further, the algorithm addresses efficiently the increased number of candidates.

Using a synthetic data generator, the performance results showed that for a variety of order and noise distributions, WM_o clearly outperforms the existing algorithms. In fact, for many workloads WM_o achieved large accuracy in prediction with quite low overhead in network traffic. Additionally, WM_o proved to be very efficient in terms of reducing the number of candidates. These results were validated with experiments using a real Web trace.

In summary, WM_o is a effective and efficient predictive Web prefetching algorithm. Future work includes:

- Investigation of the interaction between caching and prefetching. The goal of such an effort would be the development of adaptive Web caching policies that dynamically evaluate the benefits of prefetching and incorporate the prefetching decisions with the replacement mechanism.

- The extension of the present work towards the direction of using another data mining method, namely clustering, as proposed in [37], to deal with Web users access paths.

References

[1] V. Almeida, A. Bestavros, M. Crovella and A. de Oliveira: "Characterizing Reference Locality in the WWW", *Proceedings Conference on Parallel and Distributed Information Systems (PDIS'96)*, pp. 92–103, Dec. 1996.

[2] P. Atzeni, G. Mecca and P. Merialdo: "To Weave the Web", *Proceedings 23rd Conference on Very Large Data Bases (VLDB'97)*, pp. 206–215, Aug. 1997.

[3] R. Agrawal and R. Srikant: "Fast Algorithms for Mining Association Rules", *Proceedings 20th Conference on Very Large Data Bases (VLDB'94)*, pp. 487–499, Sep. 1994.

[4] R. Agrawal and R. Srikant: "Mining Sequential Patterns", *Proceedings IEEE Conference on Data Engineering (ICDE'95)*, pp. 3–14, 1995.

[5] C. Aggarwal, J. Wolf, P.S. Yu: "Caching on the World Wide Web", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, pp. 95–107, Feb. 1999.

[6] P. Barford and M. Crovella: "Generating Representative Web Workloads for Network and Server Performance Evaluation", *Proceedings ACM Conference on Measurement and Modeling of Computer Systems, (SIGMETRICS'98)*, pp. 151–160, Jun. 1998.

[7] A. Bestavros: "Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time", *Proceedings IEEE Conference on Data Engineering (ICDE'96)*, pp. 180–189, Feb. 1996.

[8] J. Borges and M. Levene: "Mining Association Rules in Hypertext Databases", *Proceedings Conference on Knowledge Discovery and Data Mining (KDD'98)*, pp. 149–153, Aug. 1998.

[9] S. Brin and L. Page: "The Anatomy of Large-scale Hypertextual Web Search Engine", *Proceedings Int. World Wide Web Conference (WWW'98)*, pp. 107–117, 1998.

[10] B. Berendt and M. Spiliopoulou: "Analysis of Navigation Behavior in Web Sites Integrating Multiple Information Systems", *The VLDB Journal*, Vol. 9, No. 1, pp. 56–75, May 2000.

[11] M. Crovella and P. Barford: "The Network Effects of Prefetching", *Proceedings of the IEEE Conf. on Computer and Communications, (INFOCOM'98)*, pp. 1232–1240, Mar. 1998.

[12] P. Cao and S. Irani: "Cost-Aware WWW Proxy Caching Algorithms", *Proceedings 1997 USENIX Symposium on Internet Technology and Systems (USITS'97)*, pp. 193–206, Jan. 1997.

[13] E. Cohen, B. Krishnamurthy and J. Rexford: "Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters", *Proceedings ACM SIGCOMM Conference*, pp. 241–253, Aug. 1998.

[14] K.M. Curewitz, P. Krishnan and J.S. Vitter: "Practical Prefetching via Data Compression", *Proceedings ACM SIGMOD Conference*, pp. 257–266, Jun. 1993.

[15] R. Cooley, B. Mobasher and J. Srivastava: "Data Preparation for Mining World Wide Web Browsing Patterns", *Knowledge and Information Systems*, Vol. 1, No. 1, pp. 5–32, Feb. 1999.

[16] M.S. Chen, J.S. Park and P.S. Yu: "Efficient Data Mining for Path Traversal Patterns", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 2, pp. 209–221, Apr. 1998.

[17] J. Dean and M. Henzinger: "Finding Related Pages in World Wide Web", *Proceedings Int. World Wide Web (WWW'99)*, pp. 1467–1479, 1999.

[18] M. Deshpande and G. Karypis: "Selective Markov Models for Predicting Web-Page Accesses", *Proceedings SIAM Int. Conference on Data Mining (SDM'2001)*, Apr. 2001.

[19] D. Duchamp: "Prefetching Hyperlinks", *Proceedings USENIX Symposium on Internet Technologies and Systems (USITS'99)*, Oct. 1999.

[20] L. Fan, P. Cao, W. Lin and Q. Jacobson: "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance", *Proceedings ACM Conference on Measurement and Modeling of Computer Systems, (SIGMETRICS'99)*, pp. 178–187, Jun. 1999.

[21] J. Griffioen and R. Appleton: "Reducing File System Latency Using a Predictive Approach", *Proceedings 1994 USENIX Conference*, pp. 197–207, Jan. 1995.

[22] W. Gaul and L. Schmidt-Thieme: "Mining Web Navigation Path Fragments", *Proceedings Workshop on Web Usage Analysis and User Profiling (WEBKDD'00)*, 2000.

[23] T. Kroeger and D.D.E. Long: "Predicting Future File-System Actions from Prior Events", *Proceedings USENIX Annual Technical Conference*, pp. 319–328, Jan. 1996.

[24] T. Kroeger, D.E. Long and J.Mogul: "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching", *Proceedings USENIX Symposium on Internet Technologies and Systems (USITS'97)*, pp. 13–22, Jan. 1997.

[25] B. Lan, S. Bressan, B. C. Ooi and Y. Tay: "Making Web Servers Pushier", *Proceedings Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, Aug. 1999.

[26] B. Lan, S. Bressan, B. C. Ooi and K. Tan: "Rule-Assisted Prefetching in Web-Server Caching", *Proceedings ACM Int. Conference on Information and Knowledge Management (CIKM'00)*, pp. 504–511, Nov. 2000.

[27] A. Nanopoulos and Y. Manolopoulos: "Finding Generalized Path Patterns for Web Log Data Mining", *Proceedings East-European Conference on Advances in Databases and Information Systems (ADBIS'00)*, pp. 215–228, Sep. 2000.

[28] A. Nanopoulos and Y. Manolopoulos: "Mining patterns from graph traversals", *Data and Knowledge Engineering (DKE)*, vol. 37, no. 3, pp. 243–266, Jun. 2001.

[29] J. Pei, J. Han, B. Mortazavi-Asl and H. Zhu: "Mining Access Patterns Efficiently from Web Logs", *Proceedings Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00)*, Apr. 2000.

[30] H. Patterson, G. Gibson, E. Ginting, D. Stodolsky and J. Zelenka: "Informed Prefetching and Caching", *Proceedings ACM Symposium on Operating Systems Principles (SOSP'95)*, pp. 79–95, Dec. 1995.

[31] V. Padmanabhan and J. Mogul: "Using Predictive Prefetching to Improve World Wide Web Latency", *ACM SIGCOMM Computer Communications Review*, Vol. 26, No. 3, Jul. 1996.

[32] T. Palpanas and A. Mendelzon: "Web Prefetching Using Partial Match Prediction", *Proceedings 4th Web Caching Workshop*, Mar. 1999.

[33] P. Pirolli, H. Pitkow and R. Rao: "Silk from a Sow's Ear: Extracting Usable Structures from the Web", *Proceedings ACM Conference on Human Factors and Computing Systems (CHI '96)*, pp. 118–125, Apr. 1996.

[34] J. Pitkow and P. Pirolli: "Mining Longest Repeating Subsequences to Predict World Wide Web Surfing", *Proceedings USENIX Symposium on Internet Technologies and Systems (USITS'99)*, Oct. 1999.

[35] R. Sarukkai: "Link Prediction and Path Analysis Using Markov Chains", *Computer Networks* Vol. 33, No. 1–6, pp. 377–386, Jun. 2000.

[36] J. Shim, P. Scheuermann and R. Vingralek: "Proxy Cache Algorithms: Design, Implementation, and Performance", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 549–562, Aug. 1999.

[37] T.W. Yan, M. Jacobsen, H. Garcia-Molina and U. Dayal: "From User Access Patterns to Dynamic Hypertext Linking", *Computer Networks*, Vol. 28, No. 7–11, pp. 1007–1014, May 1996.