Journal of Intelligent Information Systems, 20:1, 11–30, 2003
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

A Data Cube Model for Prediction-Based Web Prefetching

QIANG YANG qyang@cs.ust.hk
Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong

JOSHUA ZHEXUE HUANG jhuang@eti.hku.hk
E-Business Technology Institute, The University of Hong Kong, Hong Kong

MICHAEL NG* mng@maths.hku.hk
Department of Mathematics, The University of Hong Kong, Hong Kong

Abstract. Reducing the web latency is one of the primary concerns of Internet research. Web caching and web prefetching are two effective techniques to latency reduction. A primary method for intelligent prefetching is to rank potential web documents based on prediction models that are trained on the past web server and proxy server log data, and to prefetch the highly ranked objects. For this method to work well, the prediction model must be updated constantly, and different queries must be answered efficiently. In this paper we present a data-cube model to represent Web access sessions for data mining for supporting the prediction model construction. The cube model organizes session data into three dimensions. With the data cube in place, we apply efficient data mining algorithms for clustering and correlation analysis. As a result of the analysis, the web page clusters can then be used to guide the prefetching system. In this paper, we propose an integrated web-caching and web-prefetching model, where the issues of prefetching aggressiveness, replacement policy and increased network traffic are addressed together in an integrated framework. The core of our integrated solution is a prediction model based on statistical correlation between web objects. This model can be frequently updated by querying the data cube of web server logs. This integrated data cube and prediction based prefetching framework represents a first such effort in our knowledge.

Keywords: data cube, data mining, clustering, transition probability matrices, web prefetching

1. Introduction

An important performance issue on the World Wide Web (WWW) is web latency, which can be measured as the difference between the time when a user sends a request and when he/she receives the response. Web latency is particularly important to Internet surfers and e-business web sites. A major method to reduce web latency is web prefetching. Compared with web caching (Cao and Irani, 1997), prefetching goes one step further by anticipating users' future requests and pre-loading the anticipated objects into a cache. When a user eventually requests the anticipated objects, they are available in the cache. In the past, several prefetching approaches have been proposed (Duchamp, 1999; Markatos and Chironaki,

*To whom all correspondence should be addressed.

1998; Palpanas and Mendelzon, 1999). The general idea of prefetching is to request web objects that are highly likely to occur in the near future. Such systems often rely on a prediction model based on statistical correlation between web objects. These models are trained on previous web log data. Thus, prediction model construction is at the core of prefetching algorithms.

The web is a constantly changing environment, where new web log data is available at any instant in time. In such a fast changing environment, it is important to have the ability to support complex queries on large web logs in a multidimensional manner. In this paper, we introduce a cube model for representing sessions to effectively support different mining tasks. The cube model organizes session data into three dimensions. The *Component* dimension represents a session as a set of ordered components $\{c_1, c_2, ., c_p\}$, in which each component c_I indexes the Ith visited page in the session. Each component is associated with a set of attributes describing the page indexed by it. The attributes associated with each component are defined in the *Attribute* dimension of the cube model. Depending on the analysis requirements, different attributes can be defined in the Attribute dimension such as Page ID, Page Category and View Time spent at a page. The *Session* dimension indexes individual sessions. The details of the cube model are given in the next section. In comparison with other representation methods mentioned in Kimball and Merx (2000) and Zaiane et al. (1998), the cube model has the following advantages:

1. It represents sessions in a regular data structure to which many existing data mining algorithms can be applied.
2. It maintains the order of the page sequences.
3. It can easily include more page attributes for different analysis purposes.

Simple operations can be defined to extract necessary data from the model for different mining operations and produce reports for summary statistics and frequency counts of page visits. Therefore, the model can be used as an efficient and flexible base for Web mining.

With the data cube in place, we can then apply efficient data mining algorithms for clustering and correlation analysis. As a result of the analysis, the web page clusters can then be used to guide the prefetching system. In this work, we prefetch those web documents that are "close" to a user-requested document in a cluster model. These pages are managed by an integrated caching and prefetching system. Our system is an integrated data-cube mining and model-based web-prefetching system. It represents a novel integration of data mining research and network research.

This paper is organized as follows. In Section 2 we describe the cube model and some basic operations. In Section 3 we discuss the basis of the clustering algorithms for clustering web log data. Transition probability matrices are introduced to depict the correlation between web objects and used to cluster categorical data sequences arising from web log data. In Section 4 we describe our model based prefetching algorithm. In Section 5 we first show our clustering results of some real Web log files using the two clustering algorithms. Then we present experiment results on latency reduction. Finally, we draw some conclusions and present our future work in Section 6.

2. A cube model to represent user sessions

2.1. Session identification

We consider a Web log as a relation table T that is defined by a set of attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$. Usual attributes include *Host*, *Ident*, *Authuser*, *Time*, *Request*, *Status*, *Bytes*, *Referrer*, *Agent* and *Cookie*. Assume that transactions generated by different users are identified by a subset of attributes $S \subset \mathcal{A}$. Let U be a set of user ids and $F : S \to U$ a function that maps each unique combination of values of S to a user id of U. Let $A_t \notin S$ be the Time attribute. We first perform the following two operations on T:

1. Use F to derive a new user ID attribute A_U in T.
2. Sort T on A_U and A_t.

T is transformed to T' after the two operations. Let $A_k(t_I)$ be the value of attribute A_k in the Ith transaction of T'. We then identify sessions according to the following definition:

Definition 1. A session s is an ordered set of transactions in T' which satisfy $A_U(t_{I+1}) = A_U(t_I)$ and $A_t(t_{I+1}) - A_t(t_I) < \tau$ where $t_{I+1}, t_I \in s$ and τ is a given time threshold (usually 30 minutes).

Host IP address or Domain Name is often used in S to identify users (Shahabi et al., 2000; Spiliopoulou and Faulstich, 1998) but host IP address alone can result in ambiguity in user identification caused by firewalls and proxy servers. More attributes such as Referrer and Agent can be used to resolve this problem (Cooley et al., 1999).

2.2. The cube model

Conceptually, a session defined in Definition 1 is a set of ordered pages viewed in one visit by the same visitor. We define the number of viewed pages in a session as the length of the session. Each page identified by its URL is described by many attributes, including

- Page ID;
- Page_Category: A classification of pages in a Web site based on the context of the page contents;
- Total_Time: The total time spent at a page;
- Time: The time spent at a page in a session;
- Overall_Frequency: The total number of hits at a page;
- Session_Frequency: The number of hits at a page in a session.

The values of these attributes can be computed from particular Web log files. A particular page in a session is characterized by its attribute values while the set of ordered particular pages characterizes a session.

Let $P_{\max}$ be the length of the longest session in a given Web log file. For any session with a length $P < P_{\max}$, we define the pages of the session between $P + 1$ and $P_{\max}$ as missing

pages identified with the missing value "-". As such, we can consider that all sessions in a given Web log file have the same length.

Given the above considerations, we define a cube model for representing sessions as follows:

Definition 2. A cube model is a four tuple $\langle S, C, A, \mathcal{V}\rangle$ where S, C, A are the sets of indices for three dimensions (*Session*, *Component*, *Attribute*) in which

1. S indexes all identified sessions $s_1, s_2, \ldots, s_n$,
2. C consists of $P_{\max}$ ordered indices $c_1, c_2, ., c_{P_{\max}}$ identifying the order of components for all sessions,
3. A indexes a set of attributes, $A_1, A_2, \ldots, A_m$, each describing a property of sessions' components,
4. $\mathcal{V}$ is a bag of values of all attributes $A_1, A_2, \ldots, A_m$.

Figure 1 (left) illustrates the cube model. The order of session components is very important in the cube model while the orders of dimensions S and A are irrelevant. Each index $a_I \in A$ is associated with a pair $\langle$*AttributeName*, *DataType*$\rangle$. In this figure, we assume that sessions are sorted on the value of Length(s_I) where function Length(s_I) returns the real length of session s_I.

Definition 3. Let F be a mapping from $\langle S, C, A\rangle$ to $\mathcal{V}$ that performs the following basic operations on the cube model:

1. $F(s, c, a) = v$ where $s \in S, c \in C, a \in A$ and $v \in \mathcal{V}$,
2. $F(s_k, ., a_i) = V_{s_k,a_i}$ where V_{s_k,a_i} is session s_k represented by attribute a_i,

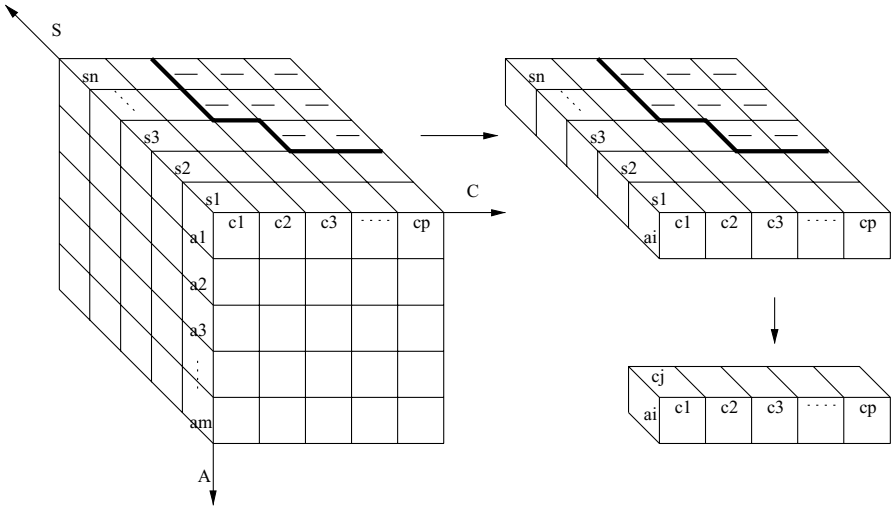

Figure 1. The cube model (left) and some operations (right).

3. $F(.,.,a_i) = V_{a_i}$ where V_{a_i} is a $p \times n$ matrix,
4. $F(.,[c_i, c_{i+z}], a_i)$ returns a $z \times n$ matrix which represents a set of partial sessions.

Definition 4. Let "|" be a cancatenation operator. $F(s_k,.,a_i) \mid F(s_{k+1},.,a_i)$ attaches session s_{k+1} to session s_k.

With these basic operators defined on the cube model, data preparation for different analysis tasks can be greatly simplified. For example, we can use $F(.,.,a_i)$ to take a slice for cluster analysis (figure 1 (right)) and use $F(s_k,.,a_i)$ to obtain a particular session described by a particular attribute for prediction (figure 1 (right)).

Aggregation operations can also be defined on each dimension of the cube model. For example, sessions can be aggregated to clusters of different levels through clustering operations. Page values can be aggregated to categories using a classification scheme.

The Component dimension presents an important characteristic of the cube model. In this dimension, the visit order of the pages in a session is maintained. Because it uses component positions as variables instead of page ids as taken by others (Nasraoui et al., 1999), it provides a regular and flexible matrix representation of page sequences which can be easily analyzed by existing data mining algorithms such as clustering and sequential association analysis.

The Attribute dimension allows the components of sessions to hold more information. For example, we can easily include time spent in each page in cluster analysis. From the these attributes, traditional Web log summary statistics such as the top pages by hits and spending time can be easily obtained.

3. Data mining: Clustering categorical sessions

The data mining technique used in this approach is clustering. The prediction models for prefetching are built from historical web access data. We use $F(.,.,a_i)$ function to extract a session matrix from the cube model and then use clustering algorithms to cluster sessions into clusters from which prefetching prediction models can be built. The clusters in this sense are used to guide the prefetching system. We prefetch those web documents that are "close" to a user-requested document in a cluster model. The cluster model tries to discover the statistical correlation between web objects using web access patterns data-mined from a web log. In the following subsection, transition probability matrices are used to depict the correlation between web objects.

3.1. Transition probability matrices

We consider that a sequence of pages visited in a user session was generated by a "Markov process" of a finite number of states, (see Taha, 1991). The next page (state) to visit depends on the current (state) only. Let n be the number of different pages. The user is said to be in the state $i (i = 1, 2, \ldots, n)$ if his current page is $G_i (i = 1, 2, \ldots, n)$. Let Q_{ij} be the probability of visiting page G_j, when the current page is G_i, i.e., the one-step transition probability. Q_{ij} can be estimated by using the information of the sequence. Suppose the

transition probability matrix Q is known for a given user and Y_m is the probability row vector of the user's state at his mth visit. We have

$$Y_{m+1} = Y_m Q \quad \text{and} \quad Y_{m+1} = Y_0 Q^m.$$

In theory the distribution of the transition frequency of the pages in the sequence (assuming the length of the sequence is much longer than the number of states n) should be consistent with the steady state probability distribution Y. This provides a method for verifying our assumptions. This Markovian approach can be illustrated by the following examples.

Consider the two sequences with three ($n = 3$) possible pages to visit:

$$\Phi_1 = \underbrace{G_1G_2G_3}_{I}\,\underbrace{G_2G_2G_3}_{II}\,\underbrace{G_3G_2G_3}_{III}\,\underbrace{G_1G_2G_3}_{IV}$$
$$\Phi_2 = \underbrace{G_2G_2G_3}_{II}\,\underbrace{G_1G_2G_3}_{I}\,\underbrace{G_1G_2G_3}_{IV}\,\underbrace{G_3G_2G_3}_{III}.$$

The sequence Φ_2 is obtained by interchanging the subsequences I and II and also the subsequences III and IV in the sequence Φ_1. Let $N^{(1)}(\Phi_k)$ ($k = 1, 2$) be the 3×3 one-step transition frequency matrix with the ijth entry $[N^{(1)}(\Phi_k)]_{ij}$ being the number of transitions from page G_i to page G_j in the sequence Φ_k. Therefore we have

$$N^{(1)}(\Phi_1) = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 1 & 4 \\ 1 & 2 & 1 \end{pmatrix} \quad \text{and} \quad N^{(1)}(\Phi_2) = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 1 & 4 \\ 2 & 1 & 1 \end{pmatrix}.$$

The one-step transition probability matrix can be obtained from the transition frequency matrix by dividing the entries of each row by its corresponding row sum. Denote the transition probability matrix of $N^{(1)}(\Phi_k)$ by $Q^{(1)}(\Phi_k)$, we have

$$Q^{(1)}(\Phi_1) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{1}{5} & \frac{4}{5} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}, \quad Q^{(1)}(\Phi_2) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{1}{5} & \frac{4}{5} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}.$$

One possible way to compare (distance) these two sequences is to consider the Frobenius norm of the difference of their transition probability matrices, i.e., $\| Q^{(1)}(\Phi_1) - Q^{(1)}(\Phi_2) \|_F$, where

$$\|B\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}[B_{ij}]^2}.$$

Table 1. Distances between sequences based on transition probability.

$\|Q^{(1)}(\Phi_1) - Q^{(1)}(\Phi_2)\|_F = \sqrt{\frac{1}{8}} = 0.354$
$\|Q^{(1)}(\Phi_1) - Q^{(1)}(\Phi_3)\|_F = \sqrt{\frac{1478}{3600}} = 0.632$
$\|Q^{(1)}(\Phi_2) - Q^{(1)}(\Phi_3)\|_F = \sqrt{\frac{3278}{3600}} = 0.954$

If we have a sequence $\Phi_3 = G_1G_2G_3G_2G_2G_3$ which is the first half of the sequence Φ_1 then we have

$$N^{(1)}(\Phi_3) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad Q^{(1)}(\Phi_3) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 1 & 0 \end{pmatrix}.$$

The Frobenius norm of the difference of two transition probability matrices also works for two sequences of differenet length. The distances under the one-step transition probability matrix approach are given in Table 1. Since the sequence Φ_2 is obtained by interchanging the subsequences in Φ_1, it is clear that the distance is small. We also note that Φ_3 is exactly a part of Φ_1, the distance between Φ_3 and Φ_1 is less than that between Φ_3 and Φ_2.

3.2. Clustering

We note that if we slice from the cube model only the Page variable in the Attribute dimension, we obtain a matrix which contains sessions described in categorical values. We remark that we can also slice more variables such as Page ID and Time and form a session matrix of mixture data types. In this case, we can employ the k-prototypes algorithm (Huang, 1998) that is designed for mixture data types. In this paper we focus on categorical sessions. Below we introduce clustering algorithms for categorical data sequences based on transition probability matrices.

Our clustering algorithm is a variant of the k-means algorithm for clustering categorical data sequences. It has made the following modifications to the k-means algorithm: (i) using the Frobenius norm of the difference of two transition probability matrices as a dissimilarity measure for two categorical sequences, and (ii) replacing the means of clusters with the mean probability matrix, to minimize the following objective function

$$J(W, Z) = \sum_{l=1}^{k} \sum_{i=1}^{n} w_{li} \|Z_l - Q_i\|_F^2 \tag{1}$$

subject to

$$\sum_{l=1}^{k} w_{li} = 1, \quad \text{for } 1 \le i \le n \quad \text{and} \quad w_{li} \in \{0, 1\}, \quad \text{for } 1 \le l \le k,$$

where Q_i is the transition probability matrix of the ith sequence, $k(\leq n)$ is a number of clusters, $W = [w_{li}]$ is a k-by-n real matrix,

$$Z = [Z_1, Z_2, \ldots, Z_k] \in \mathcal{R}^{mk},$$

and m is the number of web objects. Here Z_l contains the mean probability matrix for pages that belong to the lth cluster. In particular, if the sequences Φ_1 and Φ_2 are in the same cluster, Φ_1 and Φ_2 are highly correlated and the mean transition probabilities for the related pages of the sequences in this cluster can be obtained from Z. In Section 4, we will use this mean matrix to calculate the Pre-GDSF prefetching objects.

Minimization of J in (1) with the constraints forms a class of constrained nonlinear optimization problems whose solution is unknown. The usual method towards optimization of J in (1) is to use partial optimization for Z and W. In this method we first fix Z and find necessary conditions on W to minimize F. Then we fix W and minimize F with respect to Z. This process is formalized in the k-means type algorithm.

3.2.1. The k-means type algorithm.

1. Choose an initial point $Z^{(1)} \in \mathcal{R}^{mk}$.
   Determine $W^{(1)}$ such that $J(W, Z^{(1)})$ is minimized. Set $t = 1$.
2. Determine $Z^{(t+1)}$ such that $J(W^{(t)}, Z^{(t+1)})$ is minimized.
   If $J(W^{(t)}, Z^{(t+1)}) = J(W^{(t)}, Z^{(t)})$, then stop; otherwise goto step 3.
3. Determine $W^{(t+1)}$ such that $J(W^{(t+1)}, Z^{(t+1)})$ is minimized.
   If $J(W^{(t+1)}, Z^{(t+1)}) = J(W^{(t)}, Z^{(t+1)})$, then stop;
   otherwise set $t = t + 1$ and goto Step 2.

The matrices Z and W can be calculated according to the following two methods. Let $W^{(t)}$ be fixed, we can determine $Z^{(t+1)}$ by

$$Z_l = \frac{\sum_{i=1}^{n} w_{li} Q_i}{\sum_{i=1}^{n} w_{li}}, \quad 1 \leq l \leq k.$$

Let $Z^{(t+1)}$ be fixed, i.e., $\mathbf{Z}_l$ ($l = 1, 2, \ldots, k$) are given, we can find $W^{(t+1)}$ by:

$$w_{li} = \begin{cases} 1, & \text{if } \|Z_l - Q_i\|_F \leq \|Z_h - Q_i\|_F \quad \forall 1 \leq h \leq k, \\ 0, & \text{otherwise.} \end{cases}$$

for $1 \leq l \leq k, 1 \leq i \leq n$. The $W^{(t)}$ and $Z^{(t)}$ are updated by the above methods in each iteration. It can be shown that the above methods guarantee the convergence of the algorithm.

4. Web caching and prefetching

Now we turn our attention to the application of the cube model and data mining to web prefetching. Below, we first survey the previous work in this area and then describe the ideas behind our prediction based prefetching algorithm Pre-GDSF.

4.1. Performance metrics in caching and prefetching

Two widely used performance metrics are the hit rate and byte hit rate. The hit rate is the ratio of the number of requests that reach the proxy cache and the total number of requests. The byte-hit rate is the ratio of the number of bytes that reach the proxy cache and the total number of bytes requested. In fact, the hit rate and byte-hit rate work in somewhat opposite ways. It is very difficult for one strategy to achieve the best performance for both metrics (Cao and Irani, 1997). Web latency is a more comprehensive metric than both the hit rate and byte hit rate. In our study, we use *fractional latency* to measure the effectiveness of a caching and prefetching system. We define fractional network traffic to measure the increased network load as follows.

Definition 5. Fractional latency is the ratio between the observed latency with a caching or prefetching system and the observed latency without any caching or prefetching system.

Definition 6. Fractional network traffic is the ratio between the number of bytes that are transmitted from web servers to the proxy and the total number of bytes requested.

Obviously, the lower the fractional latency and the fractional network traffic, the better the performance. For example, a fractional latency of 30% achieved by a caching system means the caching system saves 70% of the latency.

4.2. GDSF caching algorithm

In the past, a number of web caching algorithms have been proposed. The Least-Recently-Used (LRU) algorithm replaces the object that was requested least recently. It works well for CPU caches and virtual memory systems. However, in web caching area, it does not perform well compared to some more advanced algorithms, because it considers only the temporal locality of requests (Cao and Irani, 1997; Williams et al., 1996). Least-Frequently-Used (LFU) replaces the object that has been accessed for the least number of times. It tries to keep more popular web objects and evict rarely used ones. A drawback of this policy is that some objects might build high frequency count and never be accessed again.

So far, the most popular caching algorithms are GD-Size and GDSF. The Greedy-Dual-Size (GD-Size) caching algorithm was proposed by Cao and Irani (1997). The algorithm assigns a key value to each object in the cache, so that the object with the lowest key value is replaced. When an object p is requested, GD-Size algorithm computes its key value $K(p)$ as follows:

$$K(p) = L + C(p)/S(p)$$

where $C(p)$ is the cost to bring object p into the cache; $S(p)$ is the object size; L is an inflation factor that starts at 0 and is updated to the key value of the last replaced object. If an object is accessed again, its key value is updated using the new L value. Thus, recently accessed objects have larger key values than those ones that have not been accessed for a

long time. In 1998, Cherkasova enhanced GD-Size algorithm by incorporating a frequency count in the computation of key values (Cherkasova, 1998). The algorithm is called Greedy-Dual-Size-Frequency (GDSF) algorithm. The key value of object p is computed as follows:

$$K(p) = L + F(p) * C(p)/S(p)$$

where $F(p)$ is the access count of object p. When p is initially retrieved to the cache, its frequency $F(p)$ is set to 1. If p is hit in the cache, its frequency is updated to $F(p) = F(p) + 1$. Variations of GD-Size algorithm and GDSF algorithm are distinguished by cost functions. Cao and Irani (1997) suggested that when cost function is defined as $C(p) = 1$, the algorithm achieves best latency reduction. The algorithms with $C(p) = 1$ are denoted as GD-Size(1) and GDSF(1) respectively.

4.3. Predictive prefetching

In the past, several prefetching approaches have been proposed. In 1998, Markatos et al. proposed a top-10 approach for prefetching (Markatos and Chironaki, 1998). Their general idea is to maintain a list of the top 10 popular objects for each web server, so that clients or proxy servers can prefetch these popular objects without significantly increasing network traffic. In 1999, Duchamp proposed a hyperlink based web prefetching approach (Duchamp, 1999). The general idea is to prefetch web objects hyper-linked to the current web page. In this approach, web servers aggregate reference information and then disperse the aggregated reference information piggybacked on GET responses to all clients. This information indicates how often the hyper-linked pages have been previously accessed in relation to the embedding page. Based on the knowledge on which hyper-links are generally popular, clients initiate prefetching of the hyper-links. Palpanas and Mendelzon proposed a compression-based partial-match model to drive prefetching (Palpanas and Mendelzon, 1999). Their study demonstrated the effectiveness of prefetching, especially when prefetching is deployed in browsers with limited cache sizes. However, their experiment only focused on an unrealistically small prefetching buffer size, and did not address the interaction between web caching and web prefetching. It was not clear how prefetching can be used to improve the performance of web caching.

Unlike the previous approaches, we use the web page correlation information obtained from data cube mining (clustering). The application of data mining (clustering) algorithms on the data cubes described in the last section provides important information on how strong two web pages co-occur. This mean transition probability can be read directly off the transition probability matrix for pages that belong to the same cluster. If two pages do not belong to the same cluster then they are not correlated. Otherwise, if a page A is requested by a user, then a page B should be prefetched if the transition probability matrix value of (A, B) cell in the matrix is no less than the given threshold. Therefore, our basic idea is to prefetch web objects that are highly correlated to a currently requested object.

To control the increased network traffic, a cut-off threshold value is defined on the minimum probability value, so that the prefetching algorithm only fetches the objects whose probabilities are above or equal to the cut-off value. The cut-off value can be viewed as

an accuracy control or an aggressiveness control. The higher the cut-off value, the more precise the prediction, and the less aggressive the prefetching.

Given the prediction model, the cached objects and prefetched objects are stored in the same caching buffer. The integrated model can fetch an object in response to a cache miss, or prefetch an object before it is requested based on a prediction. When an object is fetched into the cache, the prediction information based on this object is also stored, so that when the object is hit again, the server needs not consult the original web server to retrieve the prediction information. An integrated system makes the following decisions:

- When to prefetch and which object should be prefetched?
- Which object in the cache should be replaced when there is not enough space in the cache?

To handle these problems, we set a "cap" for the size of prefetched objects that have not been hit, so that the prefetching agent suspends prefetching operation when the space needed by the prefetched objects reaches the limit. In our experiments, we set the "cap" as 30% of the cache size, which we have found through preliminary tests to be good breaking point; future work could test the precise optimal point of such space breakdown. The replacement policy treats the prefetched objects and the on-demand fetched objects equally. When an object is prefetched into the cache, its frequency is set to 1. The frequency count is updated only after the object is actually requested. The replacement policy of Pre-GDSF is the same as that of GDSF, except that it replaces the object with the lowest key value that is not predicted in the prediction queue. Thus, Pre- GDSF saves the objects that will be requested in the near future from being evicted from the cache. The Pre-GDSF algorithm is presented in figure 2.

5. Experimental results

5.1. Trace driven simulation

We use web logs to simulate proxy traces. Our approach is justified by the fact that the web trace and proxy trace demonstrate similar access characteristics, such as Zipf distribution of request popularity (Almeida et al., 1996; Bestavros et al., 1995; Glassman, 1994) and temporal locality (Almeida et al., 1996; Arlitt and Williamson, 1996; Cao and Irani, 1997). Since web objects from a particular web site only occupy a small portion of proxy cache, we set the cache size to a percentage of the total size of the distinct objects from the web server.

Two web logs are used in this study. One is from a U.S. Environmental Protection Agency (EPA)'s web server located at Research Triangle Park, NC. This web log contains all the HTTP requests for a full day. The log was collected from 23:53:25 on Tuesday, August 29 1995 to 23:53:07 on Wednesday, August 30 1995, a total of 24 hours. In total, there were 47,748 requests. The timestamps have 1-second precision. The other is from a NASA Kennedy Space Center's web server in Florida. This web log contains all the HTTP requests for ten days. The log was collected from 00:00:00 July 1, 1995 to 23:59:59

```
Initialize L = 0; Q = ∅
Procedure Model-based Prefetch
begin
   Let ⟨v, tr⟩ be the current request
   Q = Q + {⟨u_i, ts_i, te_i⟩}, where ⟨u_i, ts_i, te_i⟩ is prediction based on v, and
where ts_i is the start time and ts_e the end time of a time window
   if (v in the cache)
      update F(v)
      K(v) = L + F(v) * C(v)/S(v)
   else
      load(v)
      remove all ⟨u_j, ts_j, te_j⟩ from Q, where u_j = v and ts_j < tr < te_j
   end
end
Procedure Prefetch
begin
   while (space taken by prefetched objects < 30% of cache)
      for all ⟨u_j, ts_j, te_j⟩ ∈ Q
      begin
            if (te_j > now) remove ⟨u_j, ts_j, te_j⟩
            load(u_j)
            remove ⟨u_j, ts_j, te_j⟩ from Q
            remove all ⟨u_i, ts_i, te_i⟩ from Q, where u_i = u_j and ts_i < ts_j < te_i
      end
end
Procedure load(p)
begin
   while (no enough space in cache for p)
   begin
      let L = min(K(q_j)), for all q_j in cache and q_j ∉ Q
      evict q such that K(q) = L
   end
   load p into cache
   F(p) = 1
   K(p) = L + F(p) * C(p)/S(p)
end
```

Figure 2. Pre-GDSF algorithm.

July 11, 1995. In this 10-day period, there were 719,514 requests. The timestamps also have 1-second resolution. Since there is no information identifying file modifications, we simply assume the web objects are not changed during the period when the web logs are recorded. Furthermore, we do not consider the impact of uncacheable responses. We reduce the logs by removing the uncacheable requests, such as POST requests or URLs that contain sub-strings such as "?" or "cgi". The summary of the cleaned web logs is listed in figure 3.

5.2. *Results on clustering*

To use the clustering algorithm in Section 3 to cluster a data set, the first task is to specify a k, the number of clusters to create. However, k is generally unknown for real data

Traces	EPA	NASA
Total Requests	47,748	719,514
Cacheable Requests	33,637	639,917
Sessions	2,946	57,525
Trace Size	237,101,186	14,114,855,065
Average Request Size	7,049	22,057
Distinct URLs	3,765	3,573
Size of Distinct Objects	79,648,129	159,039,462

Figure 3. Summary of web logs.

set. We tested for different values of k. After generating the clusters, we need to identify which clusters are likely to present highly correlated web logs. Because the number of clusters is quite large, individual investigation of every cluster became difficult. We selected the average distance of objects to the cluster center, i.e., the mean of the transition probability matrix of the cluster, as a measure for potential effective and useful clusters because the average distance implies the compactness of a cluster which is one of the important factors in cluster validation (Jain and Dubes, 1988). We expect that highly correlated web pages exist in compact clusters which have small average distances. We also looked at the size of clusters, i.e., the number of sessions in a cluster. A reasonable size of clusters can represent the significance of highly correlated web pages.

Here we give an example of the cluster we found in the EPA data set with session lengths between 16 and 20. We plotted in figure 4 all clusters against their average distance of objects to the cluster center (the mean transition frequency matrix of the cluster) and the number of objects in each cluster. The cluster $\otimes$ in figure 4 was analyzed. We find that there is a clear similarity among these sessions in the cluster. Next we study the mean transition probability matrix of this cluster. Figure 5 (left) shows the mean transition probability matrix of this cluster. There are 107 pages involved in this cluster. We found that the mean transition probability matrix is very sparse and there are only 186 nonzero entries. For the mean transition probability matrix of this cluster, we found that a set of web pages

$$\mathcal{C} = \{22, 37, 54, 70, 71, 77, 83, 84, 86, 87, 96, 104, 106\}$$

is closed, i.e., no web page outside this set $\mathcal{C}$ can be reached from any web page in $\mathcal{C}$. If in the transition probability matrix all rows and all columns corresponding to web pages outside this closed set $\mathcal{C}$ are removed, there remains transition probability matrix Q for which the corresponding Markov chain is irreducible (i.e., there exists no closed set other than the set of all web pages or every web page can be reached from every other web pages). The four steps transition probability (the probability from the state i to the state j after four transitions) matrix is shown as in figure 5 (right). We see that the matrix is dense and all the entries are nonzero except for the probabilities related to the closed set $\mathcal{C}$. Using

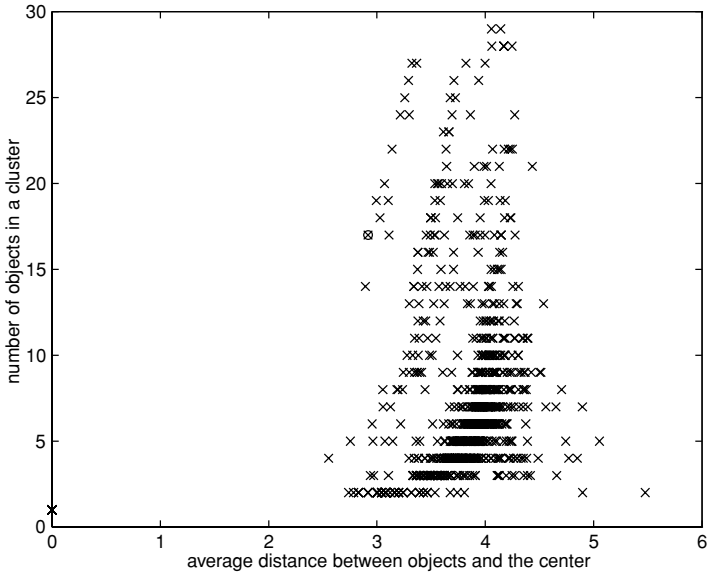

Figure 4. Distribution of numbers of objects and average distances within clusters for the data set EPA16-20.

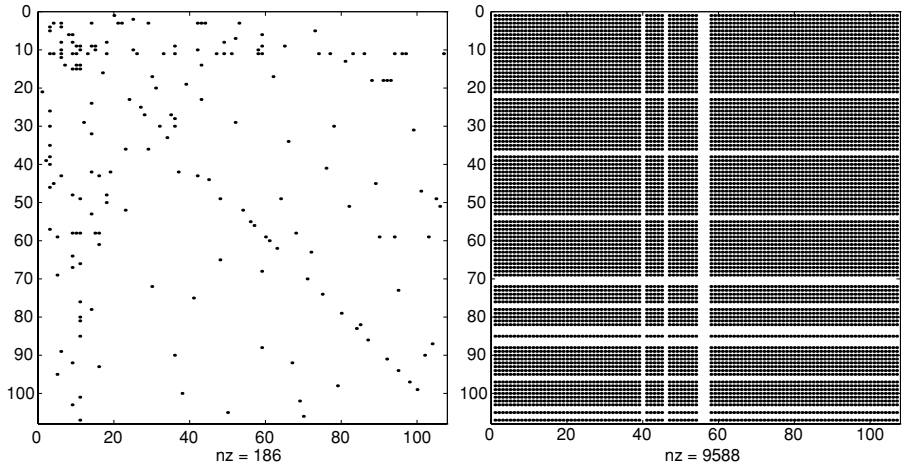

Figure 5. Left: The mean transition probability matrix. Right: The four steps transition probability matrix.

the transition probability matrix Q, a stationary (or invariant) probability distribution of web pages can be computed. We show these probability distribution in figure 6. We found that the probabilities of visiting the other 94 web pages (not the web pages in $\mathcal{C}$) in the cluster are about the same.

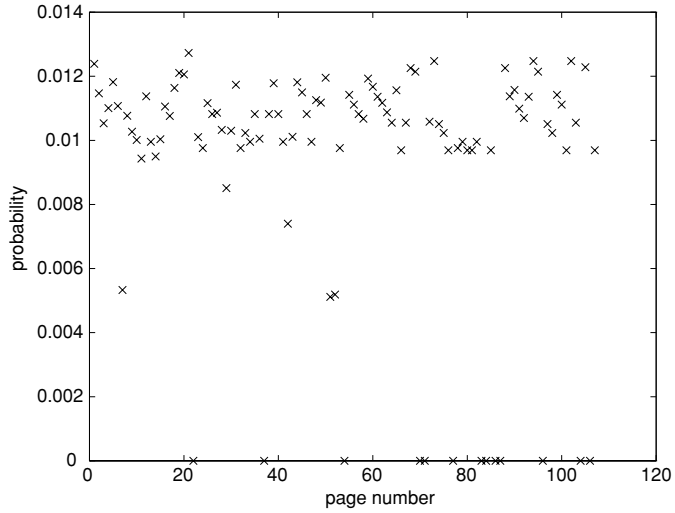

Figure 6. Stationary probability distribution for each web page.

5.3. Latency computation

A proxy server is usually located at the edge of a local area network (LAN) and acts as an intermediary between the clients and the web servers. This model naturally partitions the latency observed by the user into two components: the internal latency, which is caused by the network and the computers within the organization's bound, and the external latency, which is caused by the network and web servers outside the organization's bound. Since most of LANs are orders of magnitude faster than the Internet, the internal latency is usually much smaller than external latency (Wooster and Abrams, 1997). In the experiments, we ignore the internal latency and only consider the external latency.

In Padmanabhan and Mogul (1996). the data-pipe between the proxy and the server is modeled using linear regression. Retrieving an object of size s is assumed to incur a connection time and a transmission time, yielding a total time of

$$b_0 + s * b_1$$

where b_0 is the connection time and b_1 the per-byte transmission rate. In their study, they accumulated proxy logs recording the latency observed by the proxy and the sizes of the requested objects. From the set of n data points, $\{(l_i, s_i)\}$, where l_i is the latency, the parameters of the linear regression can be computed as

$$b_1 = \frac{n\sum_{i=1}^{n} s_i l_i - \sum_{i=1}^{n} s_i \sum l_i}{n\sum_{i=1}^{n} s_i^2 - \left(\sum_{i=1}^{n} s_i\right)^2} \quad b_0 = \frac{\sum_{i=1}^{n} l_i - b_1 \sum_{i=1}^{n} s_i}{n}$$

Their experiment on a host located in University of California, Berkeley suggests the values of the parameters as $b_0 = 1.13$ seconds and $b_1 = 5.36 \times 10^{-5}$ seconds/byte

(Padmanabhan and Mogul, 1996). In our simulation, we adopt these values to compute the latency. However, a similar experiment can be conducted to estimate b_0 and b_1, for any practical applications of our model.

5.4. Results on latency reduction

We use 50% of the sessions in each web log to find the clusters along with the probability matrices and the other 50% of the sessions to evaluate Pre-GDSF algorithm. In particular, we set the cost of each object to 1 and set the prefetching cut-off values to 0.6, 0.5, 0.3 and 0.1 separately. The algorithms are denoted as Pre-0.6, Pre-0.5, Pre-0.3 and Pre-0.1 respectively. To stabilize the status of the cache, we use 1/4 of the training sessions to warm up the cache. We conduct the experiments with different cache sizes, specifically, 0.5%, 1%, 2.5%, 5% and 10% of the total size of the distinct web objects. The results in terms of hit rate, byte hit rate, fractional latency and fractional network traffic are shown in figures 7–10.

It can be seen that Pre-GDSF outperforms GDSF for all cache sizes in terms of hit rate, byte hit rate and latency reduction. For EPA data in figure 7, Pre-0.5 improves hit rate from 56% to 65.5% and byte hit rate from 16.5% to 23%, when the cache size is 0.5%. On NASA data, Pre-0.3 improves hit rate from 67.5% to 75.3% and byte hit rate from 24.1% to 34.7%, when the cache size is 1%. Our results show Pre-GDSF can improve latency with only modest increase in network load. For EPA data, Pre-0.6 reduces latency by 4.8% (from 52.3% to 47.5%), while increasing network traffic by 2.7% when cache size is 0.5%. For NASA data, Pre-0.5 reduces latency by 6.7% (from 54.7% to 48%), while increasing network traffic by 5.8% when cache size is 1%. More importantly, a prefetching system can reduce latency more than a non-prefetching system given the same increase in the available bandwidth, because the connection time is one of the dominating factors of the latency.

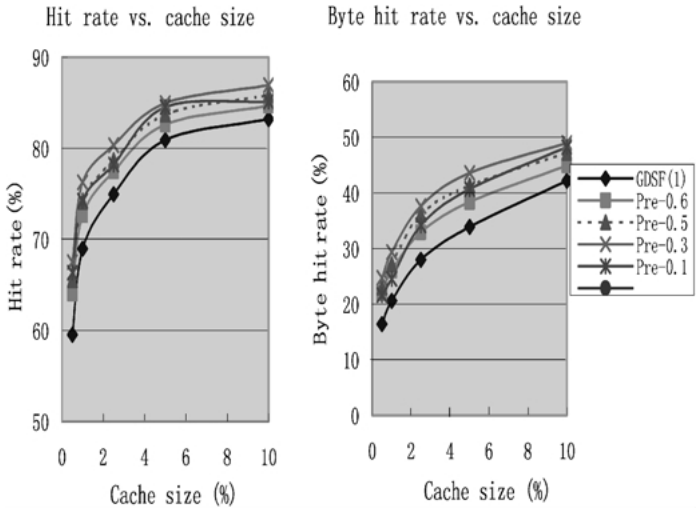

Figure 7. Comparison of hit rate and byte hit rate (EPA).

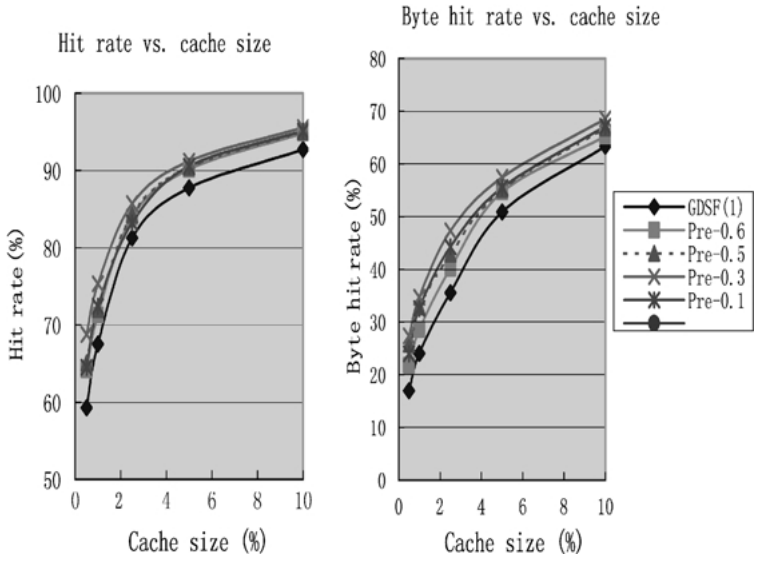

Figure 8. Comparison of hit rate and byte hit rate (NASA).

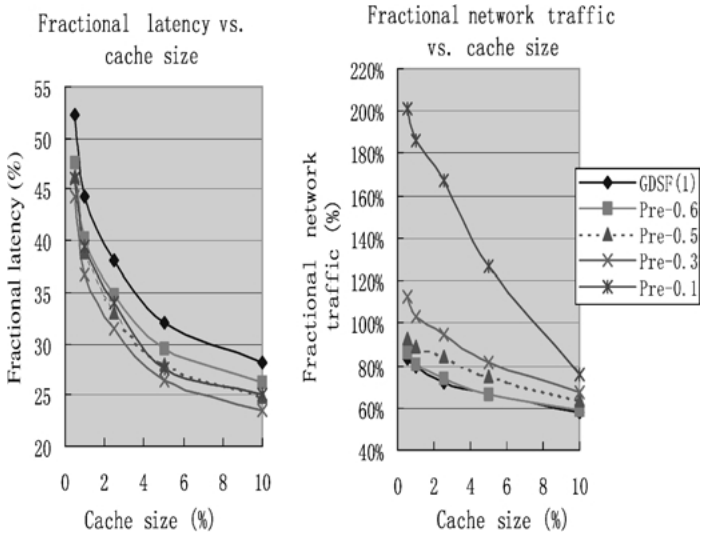

Figure 9. Comparison of fractional latency and fractional network traffic (EPA). The curve of Pre-0.6 is very close to that of GDSF.

Figures 9 and 10 show the tradeoff between latency reduction and increased network load. From these figures, we can see more aggressive prefetching algorithms achieve better latency reduction while increasing more network traffic, except for Pre-0.1. This is because Pre-0.1 is too aggressive. It prefetches too many future objects, thus causing too many

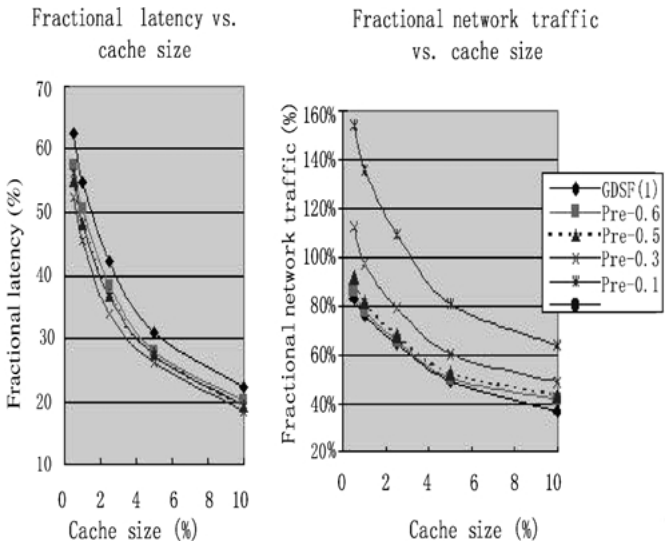

Figure 10. Comparison of fractional latency and fractional network traffic (NASA). The curve of Pre-0.6 is very close to that of GDSF.

wastes and forcing too many frequently accessed objects out of the cache. It verifies that too aggressive prefetching is harmful to the performance.

We also observe that prefetching system is more effective for smaller cache than larger ones. On EPA data, Pre-0.6 reduces latency by 4.8% (from 52.3% to 47.5%), with cache size of 0.5%, while it reduces latency by only 1.8% (from 28.1% to 26.3%), with cache size of 10%. For NASA data, Pre-0.5 reduces latency by 7.4% (from 62.4% to 55%), with cache size of 0.5%, while it reduces latency by only 2.9% (from 22.3% to 19.4%), with cache size of 10%. To explain why this happens, we show the number of prefetches made by Pre-0.5 on NASA data in figure 11.

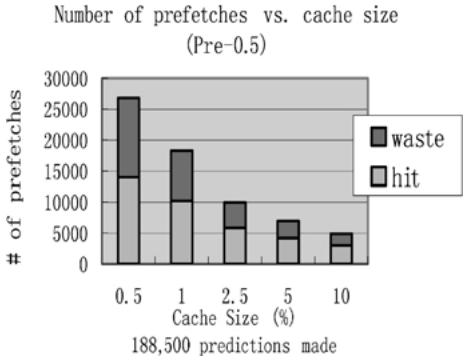

Figure 11. Number of prefetches versus cache size (NASA). *Hit* is the number of prefetched documents that are requested by the user, and *waste* is the number of prefetched documents that are not requested by the user.

In figure 11, Pre-0.5 makes totally about 188,500 predictions. When the cache size is 0.5%, 26,790 of the 188,500 predictions (about 14%) result in prefetching, while the others hit the objects already in the cache. When the cache size is 10%, only 4,876 of the 188,500 predictions (about 2.6%) result in prefetching. This is because larger caches hold more access objects, especially frequently happened ones. Once loaded into the cache, they are more likely to stay in the cache due to the effect of replacement policy. In that case, even though the predictions are correct, they do not result in prefetch. The fact that prefetching benefits small cache is particularly important for proxy servers. Because proxy caches contain web objects from many web sites, the portion of the cache used for each web site is relatively small.

6. Conclusions

We have made two linked contributions in this paper. First, we have presented a cube model to represent Web access sessions. This model is different from other cube approaches (Kimball and Merx, 2000; Zaiane et al., 1998) in that it explicitly identifies the Web access sessions, maintains the order of session's components (or Web pages) and uses multiple attributes to describe the Web pages visited in sessions. The three dimensional cube structure simplifies the representation of sequential session data and allows different data analyses to be easily conducted, such as summary statistical analysis, clustering and sequential association analysis. Second, we have applied our clustering results using the data cube model to the problem of integrated web caching and prefetching. Because the statistical results are stored in the data cubes, by applying the clustering algorithm to transition probability matrices we are able to efficiently obtain the correlation probabilities between web pages. This information is then used to power a prefetching engine. Experimental results show that the resultant system outperforms web systems that are based on caching alone.

In our future work we will conduct further cluster analysis on sessions with more attributes such as time and category, and use such information for prefetching. For example, if two sessions have a similar set of pages, whether the time spent on each would make them different. If we consolidate Web pages into categories with a classification scheme, what kind of cluster patterns would result? How are the cluster patterns related to the topology of the Web site? Can these patterns be used to improve the Web site structure? How can the Web topology be used as constraints in the clustering algorithm? All these interesting questions need further studies to answer. Moreover, in this paper, we only use one-step transition probability matrix to cluster sessions. We plan to conduct a detailed clustering analysis using multi-step transition probability matrices (i.e., the next web page to visit depends on the current and the previous pages) to cluster sessions. We expect higher order Markov model will give us a better prediction model.

Acknowledgment

We thank Michael Zhang for part of the implementation and discussion. We thank Hong Kong University of Science and Technology, the University of Hong Kong and Simon Fraser University for their support.

References

Almeida, V., Bestavros, A., Crovella, M., and Oliveira, A. (1996). Characterizing Reference Locality in the WWW. In *Proceedings of the International Conference in Parallel and Distributed Information Systems*, Miami Beach, FL, pp. 92–103.

Arlitt, M. and Williamson, C. (1996). Web Server Workload Characterization: The Search for Invariants. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*.

Bestavros, A., Cunha, C., and Crovella, M. (1995). Characteristics of WWW Client-Based Traces. Technical Report, Boston University.

Cao, P. and Irani, S. (1997). Cost-Aware WWW Proxy Caching Algorithms. In *USENIX Symposium on Internet Technologies and Systems*, Monterey, CA.

Cherkasova, L. (1998). Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy. In HP Technical Report, Palo Alto.

Cooley, R., Mobasher, B., and Srivastava, J. (1999). Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems*, 1(1), 1–27.

Duchamp, D. (1999). Prefetching Hyperlinks. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, Boulder, CO.

Glassman, S. (1994). A Caching Relay for the World Wide Web. In *The first International World Wide Web Conferencing*, Geneva, Switzerland.

Huang, Z. (1998). Extensions to the *k*-means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Mining and Knowledge Discovery*, 2(3), 283–304.

Jain, A.K. and Dubes, R.C. (1988). *Algorithms for Clustering Data*. Prentice Hall.

Kimball, R. and Merx, R. (2000). *The Data Webhouse Toolkit—Building Web-Enabled Data Warehouse*. Wiley Computer Publishing.

Markatos, E. and Chironaki, C. (1998). A Top 10 Approach for Prefetching the Web. In *Proceedings of INET'98 Conference*, Geneva, Switzerland.

Nasraoui, O., Frigui, H., Joshi, A., and Krishnapuram, R. (1999). Mining Web Access Logs Using Relational Competitive Fuzzy Clustering. In *Proceedings of the Eight International Fuzzy Systems Association Congress.*

Padmanabhan, V. and Mogul, J. (1996). Using Predictive Prefetching to Improve World Wide Web Latency. *Computer Communication Review*, 26(3), 22–36.

Palpanas, T. and Mendelzon, A. (1999). Web Prefetching Using Partial Match Prediction. Web Caching Workshop, San Diego, CA.

Shahabi, C., Faisal, A., Kashani, F.B., and Faruque, J. (2000). INSITE: A Tool for Real-Time Knowledge Discovery from Users Web Navigation. In *Proceedings of VLDB2000*, Cairo, Egypt.

Spiliopoulou, M. and Faulstich, L.C. (1998). WUM: A Web Utilization Miner. In *EDBT Workshop WebDB98*, Valencia, Spain, Springer.

Taha, T. (1991). *Operations Research*, 3rd edn., Collier Macmillan, N.Y., USA.

Williams, S., Abrams, M., Standridge, C., Abdulla, G., and Fox, E. (1996). Removal Policies in Network Caches for World Wide Web Documents. In *Proceedings of ACM SIGCOMM*, Stanford, CA, pp. 293–305.

Wooster, R. and Abrams, M. (1997). Proxy Caching that Estimates Page Load Delays. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara, CA, pp. 325–334.

Zaiane, O.R., Xin, M., and Han, J. (1998). Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs. In *Proceedings of Advances in Digital Libraries Conference (ADL'98)*, Santa Barbara, CA, pp. 19–29.