Exploiting Data Mining Techniques for Broadcasting Data in Mobile Computing Environments

Yücel Saygin, *Member*, *IEEE*, and Özgür Ulusoy, *Member*, *IEEE*

Abstract—Mobile computers can be equipped with wireless communication devices that enable users to access data services from any location. In wireless communication, the server-to-client (downlink) communication bandwidth is much higher than the client-to-server (uplink) communication bandwidth. This asymmetry makes the dissemination of data to client machines a desirable approach. However, dissemination of data by broadcasting may induce high access latency in case the number of broadcast data items is large. In this paper, we propose two methods aiming to reduce client access latency of broadcast data. Our methods are based on analyzing the broadcast history (i.e., the chronological sequence of items that have been requested by clients) using data mining techniques. With the first method, the data items in the broadcast disk are organized in such a way that the items requested subsequently are placed close to each other. The second method focuses on improving the cache hit ratio to be able to decrease the access latency. It enables clients to prefetch the data from the broadcast disk based on the rules extracted from previous data request patterns. The proposed methods are implemented on a Web log to estimate their effectiveness. It is shown through performance experiments that the proposed rule-based methods are effective in improving the system performance in terms of the average latency as well as the cache hit ratio of mobile clients.

Index Terms—Broadcast disks, broadcast histories, mobile databases, data mining, prefetching, broadcast organization.

1 Introduction

Recent advances in computer hardware technology have made possible the production of small computers, like notebooks and palmtops, which can be carried around by users. These portable computers can also be equipped with wireless communication devices that enable users to access global data services from any location. A considerable amount of research has recently been conducted in *mobile database systems* areas with the aim of enabling mobile (portable) computers to efficiently access a large number of shared databases on stationary/mobile data servers [1].

The bandwidth limitation of the wireless communication medium induces high communication cost for mobile clients. However, it might be possible to reduce this cost as servers can use a channel shared by mobile clients to broadcast data. The cost of broadcasting data of common interest is independent of the number of clients receiving it. This fact made the dissemination of data by broadcasting through wireless communication medium a cost efficient approach. However, unlike on-demand data service, broadcast environments introduce high access latency for clients. This is due to the fact that broadcast forms a unidirectional stream of data in the air, like a tape, and clients need to wait for the particular data of their interest to appear in the broadcast. The aim of our work is to decrease this latency through intelligent organization of the broadcast data and enabling the clients to perform predictive prefetching. Intelligent broadcast organization is done at the server side and the main idea is to place the data items frequently requested together close to each other. This broadcast organization method aims to decrease the access latency for subsequently requested data items. At the client side, predictive prefetching aims to improve the client cache hit ratio by predicting the data items that might be requested in the future. Predictive prefetching lowers the access latency by increasing the cache hit ratio.

We claim that the sequence of data items requested over time contains precious information about the temporal and spatial patterns of requests and this information should be exploited in broadcasting data. The broadcast requests issued over time in a mobile database environment can be stored in a *broadcast history*, where a lot of useful information about the broadcast request patterns and their relative issuing times is hidden. The broadcast history is the chronological sequence of data items that have been requested by clients. Our methods are based on analyzing the broadcast history using data mining techniques. These techniques are used for extracting useful information in broadcast histories. Data mining research deals with finding associations and sequences in individual data items by analyzing a large collection of data. We are particularly interested in the extraction of sequences, as well as clustering the data items. The problem of finding sequential patterns in a set of items has been studied before; however, to the best of our knowledge, no research results have been published so far on automated use of the resulting sequential patterns in broadcast environments. In this

- *Y. Saygin is with the Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey. E-mail: ysaygin@sabanciuniv.edu.*
- *Ö. Ulusoy is with the Department of Computer Engineering, Bilkent University, Ankara, Turkey. E-mail: oulusoy@cs.bilkent.edu.tr.*

Manuscript received 19 May 1999; revised 11 Jan. 2001; accepted 30 May 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 109855.

paper, we discuss how we can exploit sequential rules for organizing data broadcast for efficient data access by mobile clients. Our methods are based on clustering the data items that are frequently requested together. Hypergraph partitioning methods based on sequential patterns are used for clustering data items. The sequential rules obtained from sequential patterns are used for predictive prefetching at the client side. The rules are made available to mobile clients through broadcasting and they establish a rule base for predictive prefetching. Some experiments are performed to assess the effectiveness of the proposed methods. The broadcast history is simulated through a Web log and it is used for both extracting the sequential patterns and evaluating the performance impact of the resulting rules for predictive push, cache replacement, and prefetching.

The outline of the paper is as follows: Section 2 provides the background information and discusses the related work. The motivation behind our work is described in more detail in Section 3 through simple and concrete examples. Section 4 describes the process of dividing continuous client requests into transactions to be used for mining sequential patterns. Data organization in the broadcast disk through hypergraph partitioning methods is described in Section 5. Utilization of sequential rules in prefetching, cache replacement, and scheduling of broadcast data in mobile systems is discussed in Section 6. Section 7 describes the experimental set up and performance results for the proposed methods. Finally, Section 8 concludes the paper and outlines future research directions.

2 Background and Related Work

A discussion of the state-of-the-art data broadcasting research is provided in Section 2.1. The previous work performed on mining associations and sequential patterns is summarized in Section 2.2.

2.1 Data Broadcast

The continuous broadcast of data items from the server to a number of clients can be considered as simulating a rotating storage medium or a broadcast disk, as proposed in [2]. There are two main approaches for data dissemination through broadcast [3]:

1. *Push-based approach* is where data is broadcast according to predefined user profiles. This approach does not consider the current client requests.
2. *Pull-based approach* is where data is broadcast according to user requests. This approach is the same as the client-server paradigm.

Each approach has its own benefits and drawbacks. In the push-based approach, the server sends the data regardless of the current user requests, where users may end up receiving unrequired data. In the pull-based approach, on the other hand, the server load is very high since the server has to listen to client requests. To overcome these drawbacks and make use of the benefits of both approaches, a hybrid approach was developed by Acharya et al. that combines the two approaches and is called *interleaved push and pull* [3]. In this approach, there exist both a broadcast channel and a backchannel for the user requests. A hybrid data delivery model that combines push and pull[1] was also proposed by Stathatos et al. [4].

In our work, we assume a hybrid broadcast scheme where the user requests sent by the back channel are logged in the broadcast history. User requests are processed as in the hybrid scheme of [3], but the organization of the data to be broadcast is determined by sequential patterns obtained by mining the broadcast history.

In broadcast disks, the set of items that are broadcast is called the *broadcast set*. The sending sequence of the items in the broadcast set is called the *broadcast schedule*. Construction of the broadcast schedule is crucial for the performance of the broadcast disk. The broadcast schedule affects the waiting times of mobile clients for the item of their interest to arrive. This problem is similar to the problem of scheduling disk requests since we are dealing with a "disk on air" in a sense [5]. The difference is that the disk on air is unidirectional and single-dimensional since we cannot go back and forth and do random access on it. Therefore, it is more appropriate to call this new type of storage medium "one-way tape on air."

The problem of scheduling the broadcast requests is to determine the sequence of items in the broadcast schedule. We need to determine what should be put in the schedule and in what sequence by taking into account the previous request patterns of the clients (i.e., the broadcast history). Organizing the data on air is similar in a sense to organizing the items in a store since we can view the data on air as a commodity waiting to be bought by the clients. In a store, it is very logical to put the items often requested together close to each other. Similarly, the broadcast items that are requested together frequently should be broadcast close to each other in time.

When we are dealing with broadcasting in mobile environments, caching and prefetching of broadcast items also turn out to be very important from the performance viewpoint of the mobile system. The impact of caching in terms of communication cost in mobile environments was studied by Sistla and Wolfson [6]. It was shown that caching is an important factor for minimizing the communication cost, which is of great importance in mobile computing environments due to bandwidth limitations. Caching in mobile computing environments has different characteristics than caching in a traditional client-server environment. This is due to the fact that, in mobile computing environments, data items that are not cached are not equidistant to the client since the broadcast disk is single dimensional. Some caching strategies were proposed by Acharya et al. considering this aspect of broadcast disks [7]. These strategies take into account the access probabilities of the cached items together with the frequency of broadcast. A prefetching technique for broadcast disks was proposed by Acharya et al. [8]. This technique uses a heuristic that calculates a value for each data page by multiplying the probability of access for that page by the time that will elapse before that page appears next on the broadcast disk. The decision about whether a data page on broadcast is going to be replaced by one of the data pages in the mobile client cache is based on the values calculated.

1. The authors call them broadcast and unicast, respectively.

2.2 Data Mining

Advances in data storage and processing techniques have made it possible to store and process huge amounts of data. With POS (Point of Sale) machines, companies are able to store the items sold in a per-transaction basis. This new, valuable, and growing mass of data can be viewed as a gold mine since it contains valuable information that can be exploited to increase the profits of the companies. Data mining research deals with finding relationships among data items and grouping the related items together. The two basic relationships that are of particular concern to us are:

- *Association*, where the only knowledge we have is that the data items are frequently occurring together and, when one occurs, it is highly probable that the other will also occur.
- *Sequence*, where the data items are associated and, in addition to that, we know the order of occurrence as well.

Our main interest is finding the sequences among the data items that occur frequently. As the concept of sequences is based on associations, we first briefly introduce the issue of finding associations. The formal definition of the problem of finding association rules among items is provided by Agrawal and Srikant [9] as follows: Let $I = i_1, i_2, \ldots, i_m$ be a set of literals, called items, and D be a set of transactions such that $\forall T \in D, T \subseteq I$. A transaction T *contains* a set of items X if $X \subseteq T$. An *association rule* is denoted by an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. A rule $X \Rightarrow Y$ is said to hold in the transaction set D with *confidence* c if $c\%$ of the transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$.

The problem of mining association rules is to find all association rules that have support and confidence greater than some thresholds specified by the user [9]. The thresholds for confidence and support are called *minconf* and *minsup*, respectively. Various algorithms have been proposed so far for finding association rules. Some of these algorithms include AIS [10], SETM [11], Apriori [9], and AprioriTid [9].

The sequence problem aims to find the sequence relationships among data items [12]. Clustering research deals with grouping together similar data [13]. Clustering assumes that the relationship among data items is previously known and this relationship is used to group the data items that are strongly related to each other.

In our work, we deal with finding sequential patterns and their use in data broadcast and prefetching. Discovery of event patterns from event sequences was presented in [14] and [15]. Mannila et al. proposed efficient algorithms for finding event patterns by analyzing event sequences [15]. The application area used in their work is telecommunication alarm management. Bettini et al. presented algorithms based on Mannila et al.'s approach for discovering event patters. They tackled a more complex problem of finding event patterns where events have multiple granularities [14].

One of the challenges of mining client access histories is that such histories are continuous while mining algorithms assume transactional data. This causes a mismatch between the data required by current algorithms and the broadcast history we are considering. Therefore, we need to convert continuous client requests into transactional form, where client requests in a transaction correspond to a *session*. A major application area that data mining researchers are interested in is the Web, and the problem of converting continuous Web access logs to transactional form also appears in that domain. Data mining techniques are utilized for the organization and efficient querying of the Web data. Finding association rules in the context of Web was studied before [16], [17]. However, to the best of our knowledge, the automated use of association rules in scheduling broadcast requests, prefetching, cache replacement, or in any other similar application has not yet been researched. The data mining research conducted in the context of the Web is closer to our research of analyzing histories in terms of the nature of the data being mined. The work on mining Web logs is related to our work since Web requests can be thought of as client requests on data items in a server. Web logs contain various types of information, such as the IP addresses, the time, the type of action, etc. Joshi et. al. explained how Web logs can be organized and stored in a data warehouse environment to be used for data mining [17]. They also studied how the warehouse can be mined for association rules and clusters. Their work also included the development of a tool for querying the Web logs stored on the warehouse. Joshi and Khrisnapuram presented methods for efficiently organizing the sequential Web log into transactional form suitable for mining [18]. They used the temporal dimension of user access behavior and divided the sequence of Web logs into chunks where each chunk can be thought of a session encapsulating a user's interest span. Zaiane et. al. also worked on mining the Web [19]. They mainly worked on filtering the Web logs and storing the results in a data warehouse to be mined later on.

TABLE 1
Sample Database of User Requests

session no	requests
1	$e\ b$
2	$d\ a$
3	$e\ b\ c$
4	$d\ a\ f$

3 Motivation

Suppose that we have a set of data items $\{a, b, c, d, e, f\}$. A sample broadcast history over these items consisting of four sessions is shown in Table 1. The sequences extracted from this history with minimum support 50 percent are (e, b) and (d, a). The rules obtained out of these sequences with 100 percent minimum confidence are $e \rightarrow b$ and $d \rightarrow a$, as shown in Table 2. Two broadcast data organizations are depicted in Fig. 1. Fig. 1a shows a broadcast schedule without any intelligent preprocessing and Fig. 1b shows a schedule where related items are grouped together and sorted with respect to the order of reference. Assume that the disk is spinning counterclockwise and consider the following client request pattern, $e, b, d, a, c, f, e, b, d, e$, also

TABLE 2
Sample Rules

rule	confidence	support
$e \rightarrow b$	100%	50%
$d \rightarrow a$	100%	50%

shown in Fig. 1. For this pattern, if we have the broadcast schedule (a, b, c, d, e, f), which does not take into account the rules, the total waiting time for the client will be $4 + 3 + 2 + 3 + 2 + 3 + 5 + 3 + 2 + 3 = 30$ and the average latency will be $30/10 = 3.0$ broadcast units. However, if we partition the items to be broadcast into two groups with respect to the sequential patterns obtained after mining, then we will have $\{a, c, d\}$ and $\{b, e, f\}$. Note that data items that appear in the same sequential pattern are placed in the same group. When we sort the data items in the same group with respect to the rules $d \rightarrow a$ and $e \rightarrow b$, we will have the sequences (d, a, c) and (e, b, f). If we organize the data items to be broadcast with respect to these sorted groups of items, we will have the broadcast schedule presented in Fig. 1b. In this case, the total waiting time for the client for the same request pattern will be $3 + 1 + 2 + 1 + 1 + 3 + 4 + 1 + 2 + 1 = 19$ and the average latency will be $19/10 = 1.9$, which is much lower than 3.0.

Another example that demonstrates the benefits of rule based prefetching is shown in Fig. 2. The first two requests of the previous client request pattern are chosen as a snapshot. The first request is for data item e. While the client scans the broadcast disk, it checks whether the currently broadcast item is going to be requested in the future. This prediction is done by using the rules obtained from the history of previous requests. The current request is e and there is a rule stating that, if data item e is requested, then data item b will also be requested in the near future. Therefore, the prefetching decides to replace an item from the cache with the currently broadcast data item b. This way, the client sweeps the broadcast disk when searching for an item, prefetching the items on the way, that may be accessed in the future.

These simple examples show that, with some intelligent grouping and reorganization of data items and with predictive prefetching, average latency for clients can be considerably improved. In the following sections, we describe how we can extract the sequential patterns out of the client requests. We also explain how we group data items with respect to the sequential patterns and how we can sort the data items in the same group taking into account the ordering imposed by sequential patterns. We also discuss rule-based prefetching and cache replacement strategies.

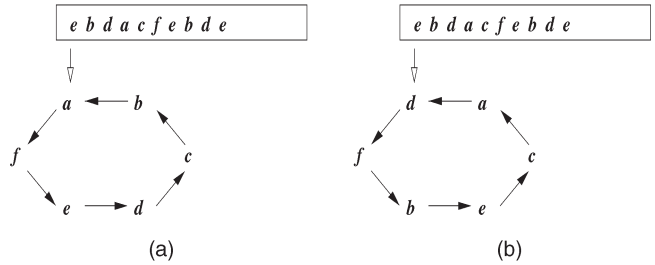

Fig. 1. Effect on broadcast data organization.

4 Mining Broadcast Histories

One of the major problems in data mining is to find patterns among a set of data items. We extract the useful information in the broadcast history in the form of sequential patterns. In the rest of this section, we discuss the issues related to the extraction of sequential patterns from broadcast histories and management of the resulting patterns.

4.1 Data and Rule Models

In order to mine for sequential patterns, we assume that the continuous client requests are organized into discrete sessions. Sessions specify user interest periods and a session consists of a sequence of client requests for data items ordered with respect to the time of reference. The whole database is considered as a set of sessions. Formally, we have a set of items, $I = i_1, i_2, \ldots, i_m$, and a set of sessions, D, such that $\forall S \in D, S \subseteq I$. A session S *supports* a sequence of items X if the items in X appear in S in the same order as they appear in X.

A sequential pattern p of size k consists of ordered data items, $p_1, p_2, \ldots, p_k$, and is represented as

$$p = < p_1, p_2, \ldots, p_k > .$$

A sequential pattern p has support s if $s\%$ of sessions in D supports p. A session supports a sequential pattern if that pattern appears in the session.

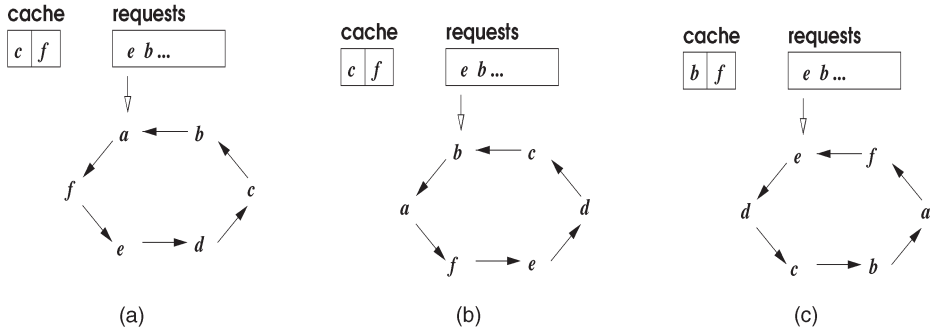

Fig. 2. Effect of prefetching.

Sequential rules are obtained from sequential patterns. For a sequential pattern $p =< p_1, p_2, ..., p_k >$, the possible sequential rules are:

$$< p_1 > \Rightarrow < p_2, p_3, \ldots p_k >,$$
$$< p_1, p_2 > \Rightarrow < p_3, p_4, \ldots p_k >,$$
$$\ldots$$
$$< p_1, p_2, \ldots, p_{k-1} > \Rightarrow < p_k > .$$

A sequential rule such as:

$$p_n =< p_1, p_2, \ldots, p_n > \Rightarrow < p_{n+1}, p_{n+2}, \ldots, p_k >,$$

where $0 < n < k$, has confidence c if $c\%$ of the sessions that support $< p_1, p_2, \ldots, p_n >$ also supports $< p1, p2, \ldots, p_k >$, i.e., $confidence(p_n) = \frac{support(<p_1,p_2,...,p_k>)\times 100}{support(<p1,p2,...,p_n>)}$.

For a sequential pattern $p =< p_1, p_2, \ldots, p_k >$, among the possible rules that can be derived from p, we are interested in the rules with the smallest possible antecedent (i.e., the first part of the rule). This is due to the fact that the rules are used for inferencing and inferencing should start as early as possible. The rest of the rules trivially meet the confidence requirement. This follows from the formulation of $confidence(p_n)$ as specified above. Support of a sequential pattern is less than the support of any of its subpattern. Therefore, as the antecedent of the rule grows, the support of the rule shrinks, making the confidence higher.

4.2 Finding Sequential Patterns in Data Broadcast Histories

In this section, we discuss how broadcast histories can be analyzed to come up with sequential rules describing the sequential order and frequency of occurrence of requests for data items in the data broadcast history.

The data mining problem in the context of broadcast disks is to extract useful information hidden in the broadcast history in the form of sequential rules. Mining of broadcast histories can be performed individually by each data server in a distributed fashion in case the broadcast history is distributed among the servers. Distributed rule mining may result in global sequential rules, as discussed in [20], or each server can maintain its own localized sequential rule set independent of the other sites. The choice of whether we should use a globalized or a localized approach is system dependent. A globalized approach is more beneficial in systems where the profiles of the data items kept in each server are basically the same. However, in case the data items stored in different servers are completely unrelated, a localized approach is more useful. Security issues also come into the picture when we deal with the analysis of private user access histories. Autonomous systems may sometimes not be willing to distribute user requests to the others. This situation can be handled by assigning symbolic ids to users and distributing the local broadcast results. However, some sites may not be willing to share valuable information with the other sites. In such a case, using a localized approach of obtaining sequential rules is the only solution.

We use a user-based partitioning approach to divide the broadcast history into subsets with respect to the user who requested them, as shown in Table 3. Table 3 shows the corresponding sessions of user requests for each user, such as $user_1$, who had k different request sessions. The analysis is done for each subset of the broadcast history corresponding to a user independent of the other subsets.

TABLE 3
User-Based Partitioning of the Broadcast History

$user_1$	(win_{1_1}: x z y; win_{1_2}: z y; ...; win_{1_k}: p q u)
$user_2$	(win_{2_1}: x p; win_{2_2}: u w v; ...; win_{2_l}: r s)
...	...
$user_n$	(win_{n_1}: x y; win_{n_2}: u v; ...; win_{n_m}: x y z)

We divide the request set into sessions and find the sequential rules using the data mining algorithms we have discussed in the preceding section. In the user-based partitioning approach, sessions are clusters of items requested according to the times of requests, i.e., if a user has requested some items consecutively with short periods of time in between, then these items should be put in the same session. We define a session as a group of requests where the time delay between two consecutive requests is less than a certain threshold. After requesting a set of items consecutively, if the user waits for a long period of time before starting another session, then the sequence of items requested in the new session can be put into another session. In Table 3, the requests of $user_1$, for instance, are divided into k sessions. The first session of $user_1$, denoted by win_{1_1}, has three requests, namely, the data items x, z, and y. We need to set a threshold value for the time delay in between two consecutive sessions. We may set this threshold to infinity to put all the items requested by the same user into the same session, which might be a reasonable approach when there are a lot of users and each user accesses a reasonable amount of (not too many) items. When the number of users is small and each user accesses a large amount of documents, then we need to set a reasonable threshold value.

The focus of our research is not the development of algorithms for finding the sequential patterns. Therefore, we used a simple and efficient sequential pattern mining algorithm on the set of sessions similar to the one described in [15]. The algorithm is based on the famous a priori principle described in Section 2.2 and it stores the candidate patterns in a hash structure for efficient support counting. The sequential patterns are then used to generate the sequential rules satisfying a minimum confidence threshold.

4.3 Elimination of Obsolete Rules and Incremental Mining

Rules that have been constructed through mining may become obsolete after a certain period of time. The rule set is dynamic in a sense. Therefore, we need to analyze the whole history periodically, eliminate the obsolete rules, and add new rules if necessary. Mining the broadcast history very frequently is a waste of server resources; however, using the same set of rules for a long time may affect the system performance negatively since the current rule set may no longer reflect the access patterns of the clients. Therefore, determination of the frequency of mining the broadcast requests is an important issue that needs to be

investigated. We may use feedback from the users to determine when the rules become obsolete. If the number of user requests is increasing, we can attribute this to obsolete rules and, when the number of user requests becomes significantly larger than the initial requests, we may decide to perform the mining process again. We can determine a threshold value for the time period to wait before restarting the mining of the broadcast history and find new sequential rules. For very huge histories, mining the whole history periodically is a waste of resources; therefore, some incremental methods can be applied to reduce the time spent for remining. Efficient methods for incremental rule maintenance are proposed in [21].

5 Broadcast Organization Using Sequential Patterns

Our main model for organizing the data items in a broadcast disk is sequential patterns. Sequential patterns are used to cluster the items that are accessed together frequently and also to impose an ordering on the data items inside a cluster.

5.1 Broadcast Organization By Clustering Data Items

Clustering in the context of data mining deals with grouping similar data items together such that the similarity among different clusters is minimized [22]. An evaluation of different clustering methods for grouping objects was studied by Bouguettaya in [13]. In the context of broadcast environments, clustering can be used to group data items that are similar to each other. Similarity of data items is defined by the client access patterns. Items that are frequently accessed together are considered to be similar. We cluster data items based on the sequential patterns obtained after mining. In other words, we infer items that are going to be accessed in the near future based on the rules obtained from sequential patterns. Clustering based on sequential patterns is therefore a natural method to use. Han et al. described a clustering method in the context of transactional data based on association rule hypergraph in [22]. A hypergraph is an extension of the standard graph model where a hyperedge connects two or more vertices, whereas, in standard graphs, an edge connects exactly two vertices. The hypergraph model perfectly fits to model sequential patterns in that the items in sequential patterns correspond to the vertices and the sequential patterns themselves correspond to the hyperedges connecting those vertices. The notion of similarity in [22] is defined by the associations among data items. They use a hypergraph model to cluster both the transactions and the data items. However, their methods are generic and not intended for a specific application like ours. Similar to the clustering method of Han et al., our method for clustering employs a hypergraph as the data model. However, due to the nature of the broadcast disk environment where the sequence of data items is important, we use sequential patterns to describe similarities among data items rather than associations. We believe that, in the case of broadcast, sequential patterns give more information than pure associations. The strength of similarity is determined by the support of the sequential pattern and confidence of the rules obtained by the sequential pattern.

TABLE 4
Sample Sequential Patterns

Sequence	Sup(%)	Rules	Conf(%)	Weight
$< a, b, c >$	2.0	$< a > \Rightarrow < b, c >$	60	220
		$< a, b > \Rightarrow < c >$	80	
$< b, c, a >$	1.0	$< b > \Rightarrow < c, a >$	50	160
		$< b, a > \Rightarrow < a >$	60	
$< b, c, d >$	2.5	$< b > \Rightarrow < c, d >$	70	260
		$< b, c > \Rightarrow < d >$	90	
$< c, e, f >$	2.0	$< c > \Rightarrow < e, f >$	60	210
		$< c, e > \Rightarrow < f >$	70	
$< e, f, h >$	1.5	$< e > \Rightarrow < f, h >$	80	230
		$< e, f > \Rightarrow < h >$	90	
$< f, g, h >$	1.0	$< f > \Rightarrow < g, h >$	60	170
		$< f, g > \Rightarrow < h >$	70	

The problem we deal with is to partition the set of data items at hand into k subsets such that the resulting subsets are balanced, the total similarities of data items in each subset are maximized, and the total similarities among the data items belonging to different subsets are minimized. This problem is similar to the *graph partitioning problem*, which was introduced by Kernighan and Lin in [23]: Given a graph G with costs on its edges, partition the nodes of G into subsets no larger than a *given maximum size* so as to minimize the total cost of the edges cut. It was shown in [23] that it is not feasible to find an exact solution to this problem. Therefore, Kernighan and Lin proposed heuristics for solving the graph partitioning problem.

The relationships among the data items are defined by the sequential patterns in our case, which may involve more than two data items. If we try to represent the relationships among the data items as a standard graph structure, then we lose the relations involving more than two data items. The hypergraph model is suitable for representing the relationships among multiple data items. A hypergraph $H = (V, E)$ is defined as a set of vertices V and a set of hyperedges E (also called nets). Every hyperedge h_i is a subset of vertices. In our model, the vertex set of the hypergraph consists of the data items to be broadcast and hyperedges correspond to sequential patterns. Determination of edge weights is a crucial point in clustering. We define edge weights based on both the support of the corresponding sequential pattern and the confidence of the rules obtained from that sequential pattern. The confidence of the rules is comparable to numeric value 100; however, the supports are usually close to one. To ensure a balance between the confidence and support in weight calculation, we scale the support values to the range $0..100$ by dividing the support to the maximum support value of the sequences and then multiplying the result by 100. Sample sequences and the rules obtained out of those sequences are provided in Table 4, together with the weights. For the sequential pattern $< b, c, d >$ in the table, the possible sequential rules are $< b > \Rightarrow < c, d >$ and $< b, c > \Rightarrow < d >$ with confidence 70 and 90, respectively. Support of $< b, c, d >$ is 2.5 percent. Since the maximum support among the sequences is also 2.5 percent, the contribution of the support of $< b, c, d >$ in the weight calculation of the hyperedge is $\frac{2.5}{2.5} \times 100 = 100$. The weight is calculated as $100 + 70 + 90 = 260$. Each hyperedge corresponds to one or more sequential patterns.

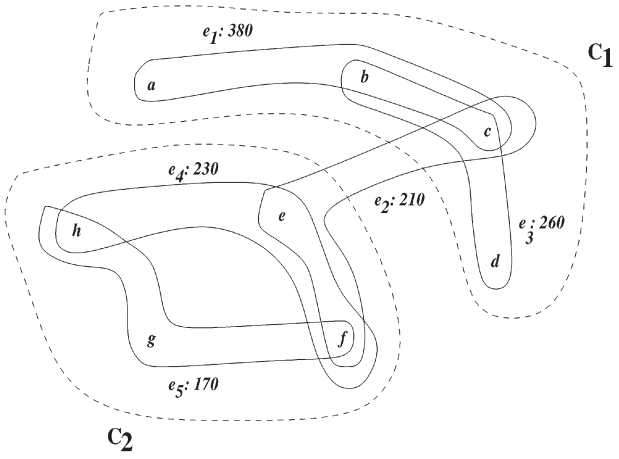

Fig. 3. The hypergraph structure for sequential rules.

The weight of a hyperedge is the sum of the support of the corresponding sequential patterns and the confidence of the rules obtained by that sequential patterns. The weight for this hyperedge is calculated by summing the weights of the corresponding sequential patterns and it is $220 + 160 = 380$. The hypergraph corresponding to the sequential patterns given in Table 4 is provided in Fig. 3. Hyperedge e_1, which connects three items, a, b, and c, corresponds to two sequential patterns, i.e., $< a, b, c >$ and $< b, c, a >$.

$P = \{V_1, V_2, \ldots, V_K\}$ is a K-way partition of $H = (V, E)$ if and only if the following three conditions are satisfied:

- $V_k \subset V$ and $V_k \neq \emptyset$ for $1 \leq k \leq K$,
- $\bigcup_{k=1}^{K} V_k = V$, and
- $V_k \cap V_l = O$ for $1 \leq k < l \leq K$.

The partitioning is sometimes referred to as bisection in the case of two-way partitioning. For $K > 2$, the partitioning is called multiway or multiple-way partitioning by some researchers. Partitioning is performed based on an objective function. The objective function in our case is to minimize the total weight of the hyperedges that span two or more partitions while performing a balanced partitioning. Partitioning is performed iteratively to obtain balanced partitions that satisfy the objective function. Recursive bisection is applied together with coarsening and uncoarsening steps for fast and effective partitioning of hypergraphs. The details of the state of the art hypergraph partitioning method we used in our work can be found in [24].

5.2 Weighted Topological Sorting of Items Inside a Cluster

As we have discussed above, clustering of data items based on sequential patterns produces sets of data items that are accessed together frequently. Besides the clustering, the ordering of the data items inside a cluster is also important. Therefore, we would like to order the data items inside a cluster such that they conform to the order imposed by sequential patterns.

In order to perform the ordering inside a cluster, we consider the set of sequential patterns of size two which form a directed graph structure. This graph may contain *cycles* and need not to be *connected*. For a given set S of sequential patterns over a set I of items, I constitutes the vertices of the graph. There is an edge $i \rightarrow j$ if and only if there is a sequence $(i, j) \in S$.

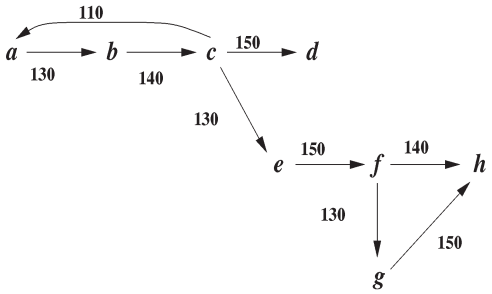

Fig. 4. Weighted graph of binary sequential rules.

The vertices of an acyclic directed graph G can be sorted by labeling the vertices with the integers from the set $\{1, 2, 3, \ldots, n\}$, where n is the number of vertices in G. If there is an edge from vertex i to vertex j in G, then the label assigned to i is smaller than that of j. Ordering the vertices in this manner is called the topological sorting of G [25].

The graph of sequential patterns may contain cycles and, in order to topologically sort the graph, we need to eliminate these cycles. Cycle elimination should be done in such a way that the edges removed have minimum impact on the overall ordering of the items. Therefore, we break a cycle by removing the edge with the minimum weight in the cycle. As described before, weights of the edges are determined by the support of the sequential pattern, and the confidence of the rule obtained form that pattern.

The graph structure for the sequential patterns given in Table 4 is provided in Fig. 4. The cycle $a \rightarrow b \rightarrow c \rightarrow a$ is broken by removing the minimum weight edge, which is (c, a). The algorithm used for weighted topological sort is presented in Fig. 5. Input of the algorithm is a weighted directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. The output of the algorithm is the topologically sorted list S of vertices in V. After the removal of the cycle, vertex a, which has zero indegree, is appended to the list of topologically sorted vertices. Vertex a is removed from the graph together with the edge (a, b). Similarly, vertex b is chosen next for removal and then

Break the cycles in G using minimum weight heuristic
$S \leftarrow \emptyset$ // S keeps the sorted list of vertices
$\forall v_i \in V, weight(v_i) = \sum_{v_j \in V, (v_j, v_i) \in E \wedge i \neq j} weight(v_i, v_j)$
$V' \leftarrow V$ // V' keeps track of the set of vertices to be removed
$E' \leftarrow E$ // E' keeps track of the set of edges to be removed
while $V' \neq \emptyset$ **do**
begin
  $Z \leftarrow Z \cup \{v_i | v_i \in V' \wedge indegree(v_i) = 0\}$
  select $v_i \in Z$ such that $weight(v_i)$ is maximum
  append v_i to S
  $\forall (v_i, v_j) \in E$ *remove* (v_i, v_j) from E'
  remove (v_i) from V'
  remove (v_i) from Z
end
Append rest of the vertices in Z to S in decreasing order of weight

Fig. 5. Weighted topological sorting algorithm.

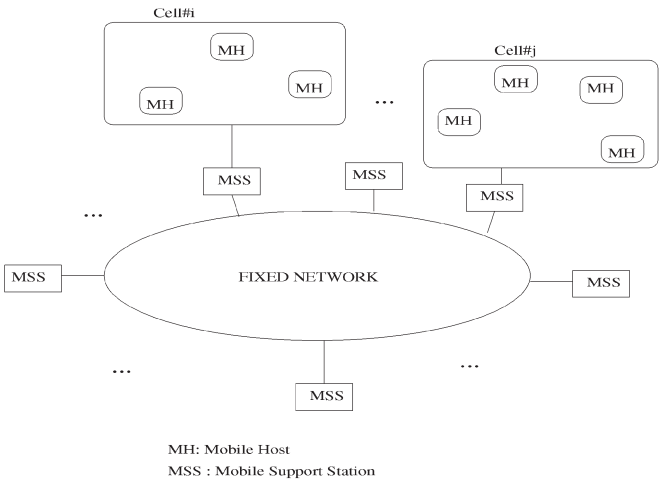

Fig. 6. A typical architecture for a mobile computing system.

vertex c. After the removal of vertex c, we have two vertices which are candidates for removal: d and e. We choose vertex d since the weight of edge (c, d) is higher than the weight of edge (c, e). The result of the topological sort of the graph is $< a, b, c, d, e, f, g, h >$.

6 Utilization of Sequential Rules in Prefetching and Cache Replacement

A typical architecture for mobile computing systems inspired from [26] is depicted in Fig. 6. In this architecture, there is a fixed network of *Mobile Support Stations* (MSSs). *Mobile Hosts* (MHs) are the computers which are portable and capable of wireless communication. Each MH is associated with an MSS and MHs are connected to MSSs via wireless links. An MSS is generally fixed and it provides MHs with a wireless interface inside a prespecified area, called a *cell*.

Both prefetching and cache replacement strategies exploit the sequential rules obtained by mining the broadcast history. The set of data items inferred by the sequential rules (that we denote by $inferred\_items$) is used as a base for rule-based prefetching and caching strategies. The set $inferred\_items$ is constructed by using the algorithm given in Fig. 7. The prefetching algorithm looks at the current n requests to predict what will be requested next. The predicted items are stored in the set of $inferred\_items$. In case the number of rules is large, we limit the number of inferred items by sorting them in decreasing order of priority and selecting the items with relatively high priority. The priority of each inferred item is determined by the confidence of the rule that has inferred it.

In order to perform rule-based prefetching and cache replacement, the mobile clients should be aware of the sequential rules in the current cell. This can be provided by ensuring that the rules also appear in the broadcast set, i.e., they are broadcast periodically. The broadcasting of rules together with the data items induces an overhead on the broadcast disk since the size of the broadcast disk increases with the addition of sequential rules. However, the number of rules in the system is usually much smaller than the number of broadcast items, therefore the induced overhead turns out to be negligible. When a mobile client enters a cell,

$inferred\_items = \emptyset$
for every *sequential rule* $S_1 \Rightarrow S_2$ **do**
  if there is a *match* for S_1 in current n requests **then**
    $inferred\_items = inferred\_items \cup S_2$
  endif
endfor

Fig. 7. Construction of the set $inferred\_items$.

it fetches the current rule set in its new cell at the time sequential rules are broadcast. This also induces an overhead on mobile clients, which may not be significant if the client does not move from one cell to another frequently. However, in case the cell sizes are small and the mobile client changes its cell frequently, loading and setting up the current rule set may take a considerable amount of time. This problem can be overcome by examining the profiles and mobility patterns of the users. The client on a mobile computer decides to load the current rule set if the user of the computer who enters a new cell will stay in this cell for a sufficiently long period of time. This information can be explicitly obtained from the user. A better approach could be to use user profiles if they are available. User profiles can contain information like [27]:

- probabilities of relocation between any two locations within a location server base (MSS),
- average number of calls per unit time, and
- average number of moves per unit time.

The problem of prefetching broadcast items is discussed in [8]. Prefetching is useful when the data items to be used subsequently are already on the air, ready to be retrieved. We suggest that sequential rules generated from broadcast histories can also be used in prefetching broadcast items.

The rule set loaded by the clients is utilized for prefetching data items in the broadcast set. Clients and the server symmetrically and synchronously utilize the sequential rules. The server uses the rules to organize the broadcast requests and clients use the rules to do prefetching. Clients consider the current n requests they have issued and do prefetching using the rules broadcast by the server. Prefetching, used with a rule-based cache replacement strategy, can decrease the waiting times of the mobile clients for the arrival of required broadcast items when those items are available on the broadcast channel. This is very intuitive since fetching a data item beforehand, when it is available, means that the client does not have to wait for the data to appear on the broadcast when it is actually required.

We have compared our approach with the prefetching method, called PT, that takes the broadcast environment into account. This method was proposed by Acharya et al. [8]. PT considers both the access probabilities of a data page and the time that the page will be broadcast again for prefetching. Each page is associated with a value, called the pt value, which is used for prefetching and cache replacement decisions. PT computes the pt value of a page by multiplying the probability of access (P) of the page with the time (T) that will elapse for the page to reappear on the broadcast. The probability of access of a data item is calculated by using the overall access frequencies of the individual data items. This is obtained after the first step of the mining process, which calculates the frequencies of

```
if bcast_item is not in cache then
    if bcast_item is in inferred_items − cached_items then
        pvalue = rule_conf(bcast_item)
    if minimum pvalue of cached_items > pvalue then
        perform prefetching
endif
```

Fig. 8. Prefetching algorithm.

patterns of size 1 (i.e., individual data items). The broadcast page is replaced with the page in the cache which has the minimum pt value if the minimum pt value is less than the pt value of the broadcast page.

The algorithm we use for prefetching is presented in Fig. 8. Similar to the PT method, a value is associated with each data item to be used for prefetching, which is called the $pvalue$ of the item. Prefetching of a data item is performed for data items that are not cache resident. Prefetching is performed if the $pvalue$ of the item on broadcast is greater than the minimum $pvalue$ of the cache resident data items. The $pvalue$ of a data item corresponds to the confidence of the rule that inferred that item (denoted by $rule\_conf(bcast\_item)$) if the item is in the list of $inferred\_items$. The $pvalues$ are halved at each broadcast tick to favor newly inferred items.

Cache replacement is also an important issue for mobile clients considering the limitations on wireless bandwidth and cache capacity of mobile computers. Clients can use the sequential rules broadcast by the servers (or MSSs) in determining which items should be replaced in their cache. Clients consider a window of current n requests in order to determine the next items that might be needed by the user and do the cache replacement according to their rule-based predictions. The rule-based cache replacement algorithm we propose is provided in Fig. 9. The algorithm first determines the data items that will probably be requested in the near future by using both the sequential rules and the last n requests issued by the client, as shown in Fig. 7. These data items are accumulated in the set $inferred\_items$. Any cache replacement strategy can be used on the set $cached\_items - inferred\_items$, i.e., the difference between the set of data items currently residing in the cache and the data items inferred by the sequential rules. Another possible approach is to use the set of inferred items for determining the replacement probabilities of cached items, instead of completely isolating them from the set of items to be considered for cache replacement. Both of the approaches can be classified as hybrid since they are taking advantage of both conventional and rule-based cache replacement strategies.

There might be a special case where the size of the set of items inferred by the rules as candidates to be requested in the near future may exceed the cache size. There are two possible approaches that might be adopted for cache replacement to handle such a case:

- Consider the inferred items for replacement in the order of timestamps of requests.
- Consider the inferred items for replacement in the order of confidences of the rules that have inferred them.

With the first approach, the temporal order of the requests is considered and the items inferred as a result of

```
if cached_items − inferred_items = ∅ then
    for each data item in
    inferred_items ∩ cached_item do
        replace the data item supported by a rule
        with the least confidence
else
    replace a data item in cached_items − inferred_items
    using their pvalues
endif
```

Fig. 9. Cache replacement algorithm.

requests which have been issued earlier are preferred to be kept in the cache. The second approach prioritizes the rules according to their confidences, i.e., among any two data items, the one inferred by the rule with a higher confidence has higher priority than the other item. The item with the lowest priority is replaced first.

7 Simulation and Experimental Results

We implemented the data mining algorithms and the rule-based scheduling and prefetching mechanisms to show the effectiveness of the proposed methods. A Web log was used for simulating the broadcast history. We think that requests for Web pages are actually a good approximation to the list of past requests by mobile clients. The simulation model we used and the experimental results are provided in Section 7.1 and Section 7.2, respectively.

7.1 Test Data and Simulation Model

We used the anonymous Web data from www.microsoft.com created by Jack S. Breese, David Heckerman, and Carl M. Kadie from Microsoft. Data was created by sampling and processing the www.microsoft.com logs and donated to the Machine Learning Data Repository stored at the University of California at Irvine Web site [28]. The Web log data keeps track of the use of the Microsoft Web site by 38,000 anonymous, randomly selected users. For each user, the data records list all the areas of the Web sites that the user visited in a one week timeframe. The number of instances is 32,711 for

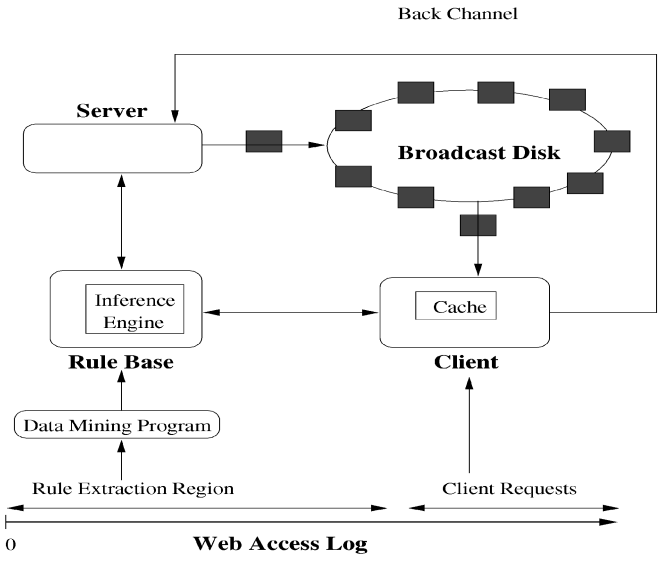

Fig. 10. System architecture.

TABLE 5
Main Parameters of Our System

Broadcast Size	Maximum size of the broadcast disk
Cache Size	Maximum size of the client cache
Minimum Support	Minimum support value of the extracted rules
Minimum Confidence	Minimum confidence value of the extracted rules
Max Inferred	Maximum number of items that can be inferred by rules
Queue Size	Size of the queue that stores the inferred items

training and 5,000 for testing. Each instance represents an anonymous, randomly selected user of the Web site. The number of attributes is 294, where each attribute is an area of the www.microsoft.com Web site and it is considered as a data item.

We selected the 50 most frequent data items for broadcasting which have more than 1 percent support. This is a reasonable method since, in broadcast environments, only the frequently accessed items are broadcast, while the others are requested through the back channel. However, since the set of 50 data items is too small for a broadcast disk, we replicated these 50 data items, together with the requests and the rules corresponding to these items, to construct a disk of 200 data items.

The whole system consists of three independent parts:

- rule extraction module,
- broadcast organization module, and
- broadcast simulation module.

The rule extraction module performs the task of extracting sequential rules from the Web log. Rule sets with different minimum confidence and support requirements can be constructed by the rule extraction module. The resulting sequential rules are written to a file in a specific format to be read later by the broadcast organization and broadcast simulation modules. The broadcast organization module performs clustering of data items using hypergraph partitioning based on sequential patterns. We used the PATOH hypergraph partitioning tool for our experiments [24]. The broadcast organization module also performs weighted topological sort of data items inside clusters. Details of the clustering and topological sorting of data items are provided in Section 5.

The architecture of our system is depicted in Fig. 10. In this architecture, we use the Microsoft Web logs to simulate the broadcast history. The sequence of Web logs that are organized into sessions is fed into the data mining program to be used for extracting sequential rules. The resulting rule set is fed into the rule base, which is then used for broadcast organization, cache replacement, and prefetching. The Web log provided for testing is used for simulating the requests of a client.

As we present in Fig. 10, our broadcast environment has a server that sends data items in the broadcast disk and a client that reads data items on the broadcast disk and sends data item requests whenever necessary. The broadcast disk is organized by the server using the sequential patterns. Clients use the sequential rules for prefetching from the broadcast disk.

We assume a simple broadcast structure as this would be sufficient for constructing an execution environment that would enable us to measure the effectiveness of extracted rules over client requests.

7.2 Experimental Results

We implemented the rule mining algorithms on our Web log. We extracted rules with different support values and evaluated their effect on the performance. Our main performance metric is the *average latency*. We also measured the *client cache hit ratio*. A decrease in the average latency is an indication of how useful the proposed methods are. The average latency can decrease as a result of both increased cache hit ratio via prefetching methods and better data organization in the broadcast disk. An increase in the cache hit ratio will also decrease the number of requests sent to the server on the backlink channel and, thus, lead to both saving of the scarce energy sources of the mobile computers and reduction in the server load.

Data mining is performed in main memory. The running time of the data mining algorithm does not exceed a few minutes. Considering that the mining process is not done very frequently, the running time is not significant. The running time of the rule checking algorithm is not significant either since the number of rules is not so large. We observed in our experiments that the optimum number of rules for the best cache hit ratio does not exceed a few hundreds.

The basic parameters of our system are presented in Table 5. As the broadcast size does not have a serious impact on the cache hit ratio, we assumed a fixed broadcast size of 200 data items in all our experiments.

We performed experiments under varying values of the cluster size and the best results for the test data were obtained with a cluster size of four. We set the minimum support threshold to 1 percent and minimum confidence threshold to 20 percent for the first part of the experiments which aims to evaluate the impact of cache size. We varied the cache size to observe how the average latency is affected. The cache size is measured in terms of the number of items, assuming that the items retrieved are html documents, possibly with images. The average latency is measured in terms of broadcast ticks.

We compared our rule-based prefetching method (RB) with the state of the art prefetching algorithm[2] (PT) for broadcast disks. We also evaluated the performance of a base algorithm (BASE) which does not do prefetching and only performs LRU-based cache replacement. PT is a very good heuristic when the locations of the data items in the disk and their relative access frequencies are known. However, with extra knowledge which describes the sequence of data item accesses by clients (i.e., by involving RB), further improvement in the performance is possible, as can be seen in Fig. 11. The cache hit ratios obtained with the

2. See Section 6.

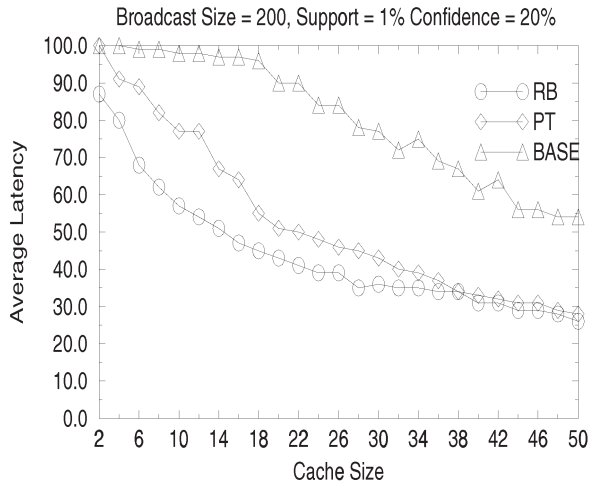

Fig. 11. Average latency as a function of the cache size.

methods RB, PT, and BASE are provided in Fig. 12. As can be observed from both figures, all three methods react similarly to the changes in the cache size. As expected, increasing the cache size leads to an increase in cache hit ratio and a decrease in average latency. For any cache size value tested, the cache hit ratio values obtained with RB and PT are much higher compared to that with BASE, which leads to much better performance for RB and PT in terms of the average latency. Comparing RB and PT, it can be observed that, except for large cache sizes tested, the cache hit ratio and the corresponding average latency values obtained with RB are consistently better than those with PT. Large cache size values (i.e., cache size $>$ 30) lead to comparable performance results for these two algorithms. RB does not provide an improvement in the performance for large cache size values because most of the data items inferred by the rules are already stored in the cache. For small cache sizes, the content of the cache is more dynamic and RB is more effective in this case. Performance improvement by RB is achieved via rule-based prefetching and cache replacement. Prefetching and cache replacement works hand in hand with RB as well as PT, i.e., these two

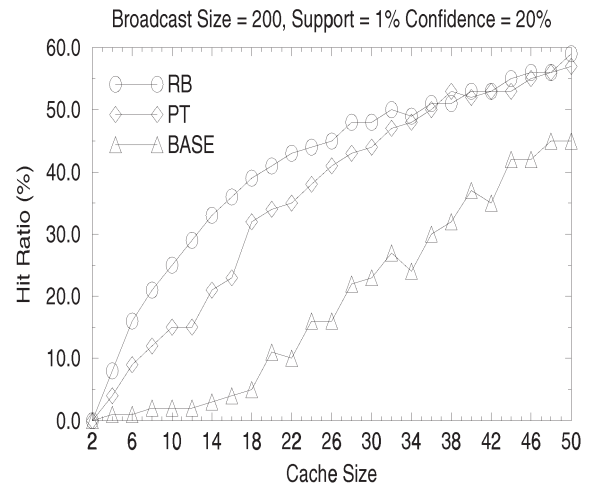

Fig. 12. Cache hit ratio as a function of the cache size.

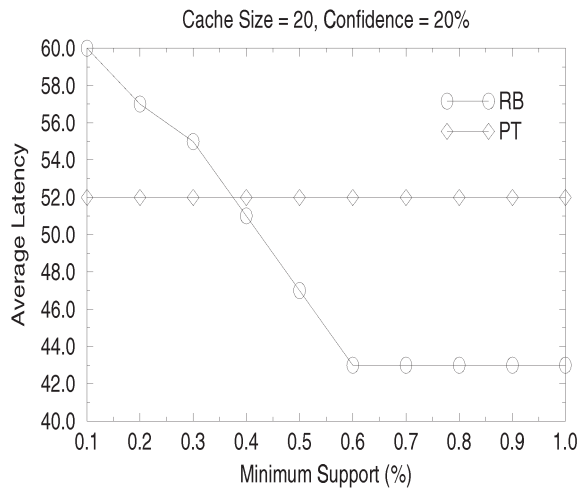

Fig. 13. Average latency as a function of the minimum support threshold (for small support values).

algorithms prioritize the cache items using the rule weights and pt values, respectively.

In order to evaluate the impact of the minimum support and confidence threshold on the performance of the rule-based prefetching, we also conducted experiments by varying these two parameters. The results obtained in these experiments are provided in Figs. 13, 14, and 15. We conducted two separate experiments in investigating the performance impact of different support values; one for small support values (in the range 0.1-1.0 in steps of 0.1) and one for large support values (in the range 1.0-10.0, in steps of 1.0). The performance results obtained for these two sets of support values are provided in Figs. 13 and 14, respectively. As support is increased for small support values, the average latency decreases since the rules chosen become more effective. The best results are obtained for the minimum support values of 0.6 through 2.0. When support is increased further, the average latency starts to increase since the number of rules decreases. The performance impact of the confidence was also examined by varying its value from 5 to 100 in steps of 5. The results are similar to

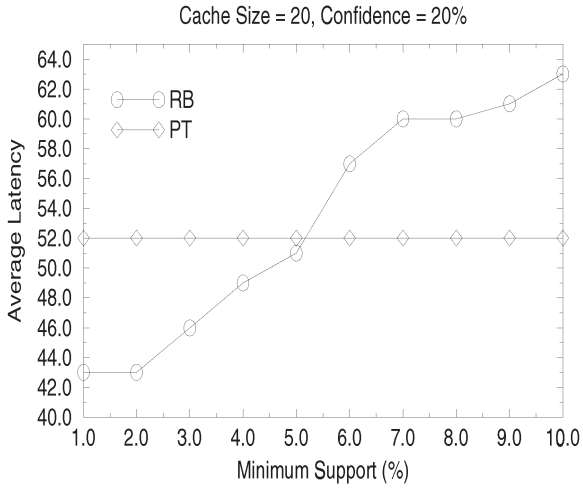

Fig. 14. Average latency as a function of the minimum support threshold (for large support values).

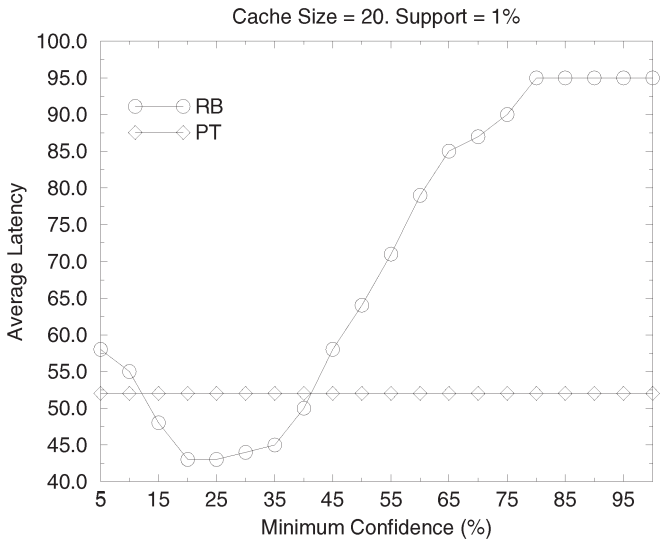

Fig. 15. Average latency as a function of the minimum confidence threshold.

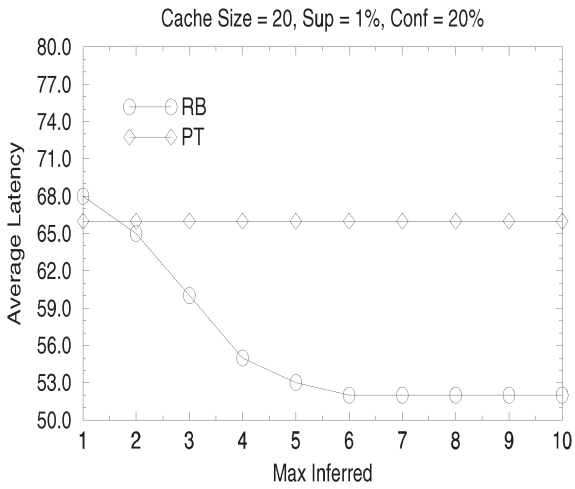

Fig. 16. Average latency as a function of the maximum number of items that can be inferred by rules.

those obtained with the minimum support experiments, as shown in Fig. 15. For small confidence values, it is possible to improve the performance by increasing the confidence threshold. However, after a certain confidence threshold (i.e., 25 percent in this experiment), increasing confidence leads to an increase in the average latency since a smaller number of rules and, thus, a smaller number of inferred items, are involved for large confidence values.

We also conducted experiments to investigate the performance impact of the maximum number of items inferred at each step (i.e., *max inferred*) and size of the queue that stores the inferred items (i.e., *queue size*). The results of these two experiments are displayed in Figs. 16 and 17, respectively. As the maximum number of inferred items is increased, the average latency decreases up to a certain point (i.e., *max inferred* = 5 in this experiment). A further increase in the maximum number of inferred items does not improve the performance because the number of items inferred cannot be increased any more. A similar performance pattern is also observed by varying the size of the queue that stores the inferred items (Fig. 17). Some of the items stored in the queue may become obsolete after a while and they are pushed out of the queue as new items are inferred. For large queue sizes, increasing the queue size does not lead to better performance because this would just increase the fraction of obsolete items stored in the queue and such items do not have any effect on the performance.

8 Conclusions and Future Work

In this paper, we have discussed some data mining techniques to be used for broadcasting data in mobile environments. We have proposed a new method of organizing the data broadcast in mobile environments using the sequential rules obtained by mining the broadcast history. The sequential rules are used as a base for clustering data items and, thus, for data organization in the broadcast disk. Our expectation with this approach is to decrease the delay experienced by the clients while waiting for the required data items since the items that are likely to be requested in the near future will be placed close together in the broadcast.

We have also aimed to show that sequential rules are beneficial for prefetching of data items by mobile clients. A cache replacement mechanism for mobile computers has been proposed which utilizes the sequential rules to determine the items to be replaced in the cache. In order to exploit sequential rules for prefetching and cache replacement, mobile clients need to have access to the current sequential rules that exist in the system. Servers can handle this problem by periodically broadcasting the current sequential rule set.

The proposed methods have been evaluated through performance experiments on a Web log. The rules resulting from mining a Web log have been used to test the effectiveness of the proposed methods. The performance of our methods has been compared with a state of the art prefetching technique, as well as a base algorithm. It has been observed through performance experiments that a considerable increase in the cache hit ratio can be obtained when the rule-based broadcast organization, cache replacement, and prefetching techniques are used together. The

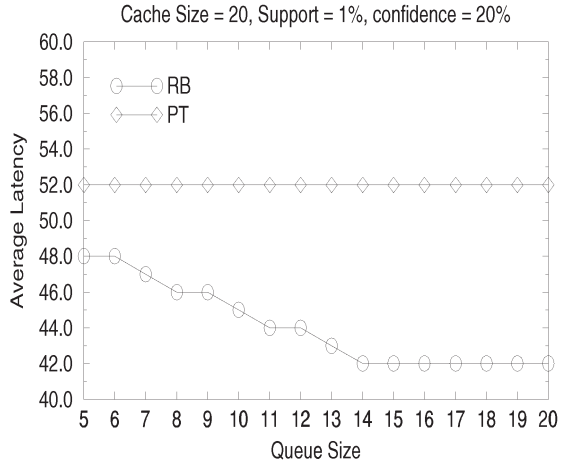

Fig. 17. Average latency as a function of the size of the queue that stores the inferred items.

increase in the cache hit ratio leads to lower average latency for rule-based methods especially for small cache sizes, which is typical for mobile devices.

In this work, we have not dealt with temporality issues. However, temporal information can also be exploited for scheduling broadcast requests. Temporal sequential rules can be obtained by mining the broadcast history, considering the relative times of the broadcast requests as well. The issue of mining temporal patterns is discussed in [14]. Temporal sequential rules can improve the effectiveness of rule-based scheduling by considering the time of the requests as well as the sequence of requests. The issue of utilization of temporal sequential rules in broadcasting is left as future work.

ACKNOWLEDGMENTS

The authors wish to thank Dr. Cevdet Aykanat and Bora Uçar for setting up their hypergraph partitioning tool for our use. This research is supported by the Research Council of Turkey (TÜBITAK) under grant number EEEG-246.

REFERENCES

[1] J. Jing, A. Helal, and A.K. Elmagarmid, "Client-Server Computing in Mobile Environments," *ACM Computing Surverys,* vol. 31, no. 2, 1999.
[2] S. Zdonik, M. Franklin, R. Alonso, and S. Acharya, "Are 'Disks in the Air' Just Pie in the Sky," *Proc. IEEE Workshop Mobile Computing Systems and Applications,* Dec. 1994.
[3] S. Acharya, M. Franklin, and S. Zdonik, "Balancing Push and Pull from Data Broadcast," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* May 1997.
[4] K. Stathatos, N. Roussopoulos, and J.S. Baras, "Adaptive Data Broadcast in Hybrid Networks," *Proc. 23rd VLDB Conf.,* 1997.
[5] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Data on Air: Organization and Access," *IEEE Trans. Knowledge and Data Eng.,* vol. 9, no. 3, July/Aug. 1997.
[6] A.P. Sistla and O. Wolfson, "Minimization of Communication Cost through Caching in Mobile Environments," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 4, Apr. 1998.
[7] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management for Asymmetric Communication Environments," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* June 1995.
[8] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a Broadcast Disk," *Proc. 12th Int'l Conf. Data Eng. (ICDE '96),* Feb. 1996.
[9] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Databases,* Sept. 1994.
[10] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. ACM SIGMOD Conf. Management of Data,* May 1993.
[11] M. Houtsma and A. Swami, "Set-Oriented Mining of Association Rules," technical report, IBM Almaden Research Center, San Jose, Calif., Oct. 1993.
[12] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. Int'l Conf. Data Eng. (ICDE),* Mar. 1995.
[13] A. Bouguettaya, "On-Line Clustering," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 2, 1996.
[14] C. Bettini, X.S. Wang, S. Jajodia, and J.-L. Lin, "Discovering Frequent Event Patterns with Multiple Granularities in Time Sequences," *IEEE Trans. Knowledge and Data Eng.,* vol. 10, no. 2, 1998.
[15] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovery of Frequent Episodes in Event Sequences," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining,* Aug. 1995.
[16] B. Mobasher, N. Jain, E.-H. Han, and J. Srivastana, "Web Mining: Pattern Discovery from World Wide Web Transactions," Technical Report 96-050, Dept. of Computer Science, Univ. of Minnesota, Sept. 1996.
[17] K.P. Joshi, A. Joshi, Y. Yesha, and R. Krishnapuram, "Warehousing and Mining Web Logs," *Proc. ACM CIKM Second Workshop Web Information and Data Management (CIKM '99),* 1999.
[18] A. Joshi and R. Krishnapuram, "On Mining Web Access Logs," *Proc. SIGMOD Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD),* 2000.
[19] O.R. Zaiane, M. Xin, and J. Han, "Discovering Web Access Patterns and Trends by Applying Olap and Data Mining Technology on Web Logs," *Proc. Advances in Digital Libraries Conf.,* 1998.
[20] D.W.-L. Cheung, V. Ng, A.W.-C. Fu, and Y. Fu, "Efficient Mining of Association Rules in Distributed Database," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, no. 6, 1996.
[21] D.W.-L. Cheung, J. Han, V. Ng, and C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," *Proc. 12th Int'l Conf. Data Eng. (ICDE),* pp. 106-114, 1996.
[22] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher, "Clustering Based on Association Rule Hypergraphs," *Proc. SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '97),* 1997.
[23] B.W. Kerninghan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical J.,* vol. 49, no. 2, 1970.
[24] U.V. Catalyurek and C. Aykanat, "Hypergraph-Partitioning Based Decompostion for Parallel Sparse-Matrix Vector Multiplication," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, pp. 673-693, 1999.
[25] K. Thulasiraman and M.N.S. Swamy, *Graphs: Theory and Algorithms.* Wiley and Sons, 1992.
[26] T. Imielinski and B.R. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Comm. ACM,* pp. 19-27, Oct. 1994.
[27] T. Imielski and B.R. Badrinath, "Querying in Highly Distributed Environments," *Proc. 18th VLDB Conf.,* 1992.
[28] University of California at Irvine Machine Learning Repository, http://www.ics.uci.edu/~mlearn, 2002.

Yücel Saygin received the PhD degree in computer engineering from Bilkent University, Turkey, in 2001. He is currently an asistant professor in the Faculty of Engineering and Natural Sciences, Sabanci University, Turkey. His main research interests include mobile data management, data mining, and application of data mining technology to database management systems. He is a member of the IEEE.

Özgür Ulusoy received the PhD degree in computer science from the University of Illinois at Urbana-Champaign. He is currently an associate professor in the Computer Engineering Department of Bilkent University in Ankara, Turkey. His research interests include wireless data access, data management for mobile systems, web query languages and data models, multimedia database systems, and real-time and active database systems. Professor Ulusoy has served on numerous program committees for conferences, including the International Conference on Very Large Databases, the International Conference on Data Engineering, and the International Conference on Scientific and Statistical Database Management. He was the program cochair of the International Workshop on Issues and Applications of Database Technology that was held in Berlin in July 1998. He coedited a special issue on real-time databases in *Information Systems Journal* and a special issue on current trends in database technology in the *Journal of Database Management.* He also coedited a book on *Current Trends in Data Management Technology.* He has published more than 50 articles in archived journals and conference proceedings. He is a member of the IEEE and the IEEE Computer Society.