EXPLOITING DATA MINING TECHNIQUES FOR BROADCASTING DATA IN MOBILE COMPUTING ENVIRONMENTS[1]

Yücel Saygın and Özgür Ulusoy

Department of Computer Engineering and Information Science, Bilkent University, Turkey.

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. It is crucial for mobile clients to use their scarce resources efficiently while retrieving the required data from the data broadcast by the server machine. In this paper, we propose some methods for predicting and pushing the data that might be requested subsequently by mobile clients. 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. We also propose cache replacement and prefetching techniques for mobile clients based on the association rules obtained by mining the broadcast history. 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 cache hit ratio of mobile clients.

Key words: Broadcast disks, broadcast histories, mobile databases, data mining, caching, prefetching.

1 Introduction

Recent advances in computer hardware technology have made it 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* area with the aim of enabling mobile (portable) computers to access efficiently to a large number of shared databases on stationary/mobile data servers.

The bandwidth limitation of 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. An important issue in broadcasting data is to determine which data items should be broadcast and in what sequence. The set of data items to be broadcast can be either:

- *static*, i.e., they do not change depending on user requests, or

[1]This research is supported by the Research Council of Turkey (TÜBİTAK) under grant number EEEAG-246 and the NATO Collaborative Research Grant CRG 960648.

- *dynamic*, i.e., they may change over time, according to user requests.

Static broadcast sets are used when the user profiles are well known and do not change over time. The set of items to be broadcast is precompiled before starting the broadcast process. Dynamic broadcast sets, on the other hand, are reconstructed when the user requirements change over time, and these changes are reflected in the broadcast sets. Dynamic broadcasting requires monitoring of data items requested by the clients.

In this paper, we investigate the issues related to broadcasting data to mobile clients. We assume that data access requirements of users may change over time and therefore the broadcast data should be reorganized and reconstructed dynamically. 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 scheduling future data broadcast requests. 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. We propose some methods for predicting and pushing the data that will probably be requested subsequently by mobile 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. The information in a broadcast history is extracted in the form of association rules. The problem of finding associations among a set of items has been studied for a long time, however, to the best of our knowledge, no research results have been published so far on automated use of the resulting association rules in any application. Agrawal and Psaila introduced the notion of active data mining in [AP95], but their work is different from ours in that they define triggers on the set of resulting association rules whereas we actually use the resulting association rules for inferencing. In this paper, we discuss the automated use of association rules for scheduling the broadcast of data to mobile clients. The resulting association rules are also shown to be a useful tool for prefetching and cache replacement when the rules are available to mobile clients through broadcasting. 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 both for extracting the association rules and for 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. Section 3 describes the problem of mining for association rules, presents an algorithm for mining association rules, and discusses how this algorithm can be used for mining broadcast histories. Utilization of association rules in prefetching, cache replacement, and scheduling of broadcast data in mobile systems is discussed in Section 4. Section 5 describes the experimental set up and performance results for the proposed methods. Finally, Section 6 concludes the paper and outlines future research directions.

2 Background and Related Work

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 [ZFAA94]. There are two main approaches for data dissemination through broadcast [AFZ97]:

1. *Push based approach* where data is broadcast according to predefined user profiles. This approach does not consider the current client requests.

2. *Pull based approach* 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 with 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* [AFZ97]. 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[2] was also proposed by Sthathatos et al. [SRB97].

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 [AFZ97], but the data to be broadcast is determined by both the user requests and the association rule set constructed by mining the broadcast history. Sending the data inferred by the rules may be described as predictive push.

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 [IVB97]. The difference is in that the disk on air is uni-directional 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 current requests of the clients and the previous broadcast schedules (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 requested often together close to each other. Similarly, the broadcast items that are requested together frequently should be broadcast close to each other in time.

The problem of determining the portion of the broadcast schedule that will be spared for the requests (pull) and the portion for the data sent regularly (push) is discussed in [AFZ97]. We need to further determine the portion of the broadcast schedule that will be spared based on the estimates done by mining the broadcast histories.

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 [SW98]. 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 [AAFZ95]. These strategies take into account the access

[2]The authors call them broadcast and unicast, respectively.

probabilities of the cached items together with the frequency of broadcast. A prefetching technique for broadcast disks was proposed by Acharya et al. [AFZ96]. 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 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.

3 Mining Broadcast Histories

In our work, we provide an analysis of the broadcast history by using data mining techniques. One of the major data mining problems is to find association rules among a set of data items. We extract the useful information in the broadcast history in the form of association rules. In the rest of this section, we discuss the issues of the extraction of association rules from broadcast histories and management of the resulting association rules.

3.1 Extracting Association Rules: A Data Mining Problem

The advance of data storage and processing techniques made the storage and processing of large amounts of data possible. 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 problems are usually grouped into the following three categories:

- classification,
- association,
- sequence.

In classification, we try to partition the data into disjoint groups [AGI+92]. Some applications of the classification algorithms include the store location problem, finding the best N candidates, and credit card approval [AIS93a]. In the association problem, we try to find some correlations among data items [RS95]. Catalog design, store layout and finding discount items are few of the applications of finding associations. Finally, the sequence problem tries to find the sequences among data items [AS95]. It was shown by Agrawal et al. that all these problems can be mapped to a unified framework [AIS93a]. In our work, we deal with the association problem; i.e., we try to find association rules among items with a given confidence.

The problem of finding association rules among items is formally defined by Agrawal et al. [AS94]. The definition of this problem is given below for stating the problem clearly:

Let $I = i_1, i_2, ..., i_m$ 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 a support and a confidence greater than some thresholds specified by the user [AS94]. 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 [AIS93b], SETM [HS93], Apriori [AS94], and AprioriTid [AS94].

Given the set of items I, the power set of I, denoted by $P(I)$, contains all subsets of I. Each element of $P(I)$ is called an *itemset*. The number of transactions that contain an itemset is called the *support* of that itemset. An itemset in $P(I)$ can be either:

- *large*, if its support is greater than or equal to *minsup*, or
- *small*, if its support is less than *minsup*.

Given the above definitions that are provided in [AS94], the problem of finding association rules is decomposed into two main steps:

1. Find all *large itemsets*.
2. Generate association rules only from *large itemsets*.

The second step of the problem can be solved easily. For every large itemset l, we generate all non-empty subsets of l, i.e., $P(l) - \emptyset$. For each a in $P(l) - \emptyset$, a rule of the form, $a \Rightarrow (l - a)$ is produced only if,

$$\frac{support(l)}{support(a)} \geq minconf$$

The number and the size of large itemsets are manageable. Therefore, the time complexity of the second step is not considerable. However, the generation of large itemsets from the set of transactions, D, is not an easy task since all the subsets in D need to be generated and tested. Considering a huge set of transactions, one can easily see that the time complexity of the first step is exponential.

The algorithms for discovering association rules should have the following steps [AS94]:

1. Find the *large itemsets* of size one, i.e., *1-itemsets*.
2. Initialize *seed* to *1-itemsets*. *Seed* is used to generate new *candidate* (i.e., potentially large) itemsets.
3. Count the support of the itemsets in the set of new candidates, and eliminate the ones with support less than *minsup*.
4. Initialize the seed with the new large items and continue with Step 3 until no further large itemsets are left.

3.2 Finding Association Rules in Data Broadcast Histories

Association rules were proposed to be a useful tool to determine the locations of objects in a store and to boost the sales of one item by decreasing the price of another item [AS94]. Finding association rules in the context of web was also studied [MJHS96]. 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 been studied. The data mining research done in the context of web is closer to our research of analyzing histories in terms of the nature of data being mined. In this subsection, we discuss how broadcast histories can be analyzed to come up with association rules describing the associations in the data broadcast requests of mobile clients.

$uscr_1$	(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)

Table 1: User-based partitioning of the 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 association 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 association rules as discussed in [CNFF96], or each server can maintain its own localized association 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 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 distribute 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 association rules is the only solution.

There can be two basic approaches for mining the broadcast history depending on how the history of user requests is partitioned:

- *Flat* approach.
- *User-based partitioning* approach.

The flat approach tries to extract data item request patterns regardless of who requested them. The user-based partitioning approach, on the other hand, divides the broadcast history into subsets with respect to the user who requested them as shown in Table 1. Table 1 shows the corresponding windows of user requests for each user, such as $user_1$ who had k different request windows. The analysis is done for each subset of the broadcast history corresponding to a user independent of the other subsets.

In both approaches, we divide the request set into windows and find the association rules using the data mining algorithms we have discussed in the preceding subsection. However, the construction of the windows is different for the two approaches. In the user-based partitioning approach, windows 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 window. We define a window 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 window. In Table 1, the requests of $user_1$, for instance, are divided into k windows. The first window of $user_1$, denoted by win_{1_1}, has 3 requests, namely the data items x, z, and y. We need to set a threshold value for the time delay in between two consecutive windows. We may set this threshold to infinity to put all

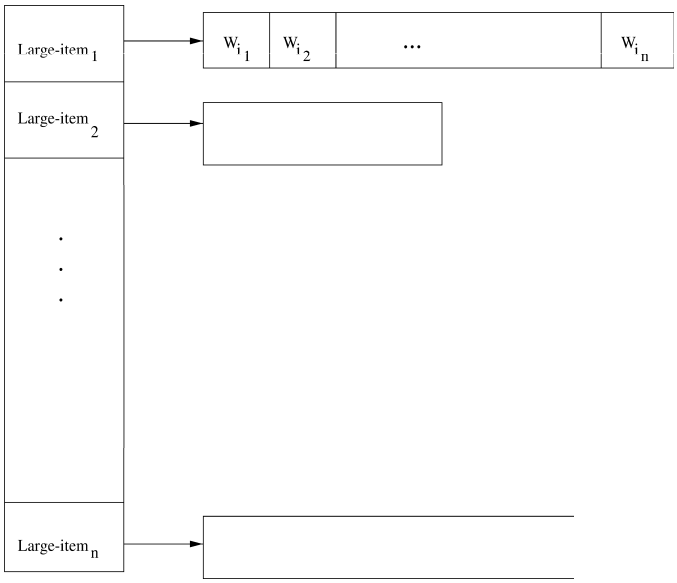

Figure 1: Inverted list data structure for mining broadcast histories

the items requested by the same user into the same window, 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. For the flat approach, we determine a fixed window length and use a sliding window approach to construct the windows. The sliding window approach is based on moving the window one item further and consider the items in the current window for finding the large itemsets.

The advantage of the flat approach is that there is no overhead of partitioning the items into users and clustering the items into windows. But the information about the users is lost in this approach. In user-based partitioning, the rules obtained can be considered to be more reliable, as they are based on finding association rules for the same user by partitioning the broadcast history into users.

In order to be able to use the data mining algorithms described in Section 3.1 to obtain association rules, we need to map the windows to the *transactions* and the data items to the *items* described in Section 3.1.

An inverted list data structure is appropriate for extracting association rules out of the collection of windows as shown in Figure 1. Initially all the large data items are determined by counting their supports. Large data items are characterized by the fact that the percentage of the windows that support them is greater than the minimum support value. The inverted list structure keeps the large items and the corresponding windows that support them. The inverted list structure can be used to simplify the calculation of the support values of itemsets. This can be achieved by implementing the list of windows that support a large item as a set. Set operations are then used for the calculation of support values of itemsets. For example, the support of an item set consisting of $Large\text{-}item_1$ and $Large\text{-}item_2$ is calculated by taking the intersection of the supporting sets of $Large\text{-}item_1$ and $Large\text{-}item_2$.

3.3 Implementation of Association Rules as Active Rules

The association rules obtained as a result of mining the broadcast history can be implemented as active rules. Active rules are used to declaratively express system responses and they are usually called Event-Condition-Action (ECA) rules [Day88, SUC98]. An ECA rule is composed of an *event* that triggers the rule, a *condition* describing a given situation, and an *action* to be performed if the condition is satisfied. We can implement association rules as event-action (EA) rules by dropping the condition (C) part. EA rules have an event and an action. The conditions of EA rules are assumed to be true by default. An association rule of the form $S_1 \Rightarrow S_2$ can be defined as an EA rule with event S_1 and action S_2. This way we can exploit features of active systems in broadcast environments. Assuming that we have a set of n data items $D = \{d_1, d_2, ..., d_n\}$, our set of primitive events E has n elements as well where each event corresponds to the arrival of a client request. The primitive event set is defined as $E = \{e_1, e_2, ..., e_n\}$ where e_i is the *arrival of a request for* d_i. The composite events which are constructed by using primitive events can be described by using the association rules. For an association rule $S_1 \Rightarrow S_2$, the head of the rule (i.e., S_1) can be specified as a composite event to be detected. Actions of the rules just affect the new broadcast schedule by inserting the items that are specified at the tail of the rule (i.e., S_2) to the current schedule. For example, for a rule $S_1 \Rightarrow S_2$ where $S_1 = d_3\ d_2\ d_5$ and $S_2 = d_6\ d_9\ d_7$, if the primitive events for the arrival of requests for data items d_3, d_2, and d_5 are e_3, e_2, and e_5 respectively, then the composite event for the rule is $e_c = e_3 \wedge e_2 \wedge e_5$. The action of the rule can be specified as "append data items, d_6, d_9, d_7 into the *new_broadcast_schedule*" where *new_broadcast_schedule* is the schedule constructed with the help of the association rules.

When we implement the association rules as EA rules, the system automatically detects the rules whose events are signaled by the arrival of requests, and then updates the broadcast schedule accordingly. However, the issue of cascaded rule triggering should be carefully examined since cascaded rule firing may cause cycles of triggered rules. Termination of a set of ECA rules was studied in [AWH92]. The cascaded rule firing and the termination problem are beyond the scope of our work.

3.4 Elimination of Obsolete Rules

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 the 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 association 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 [CHNW96].

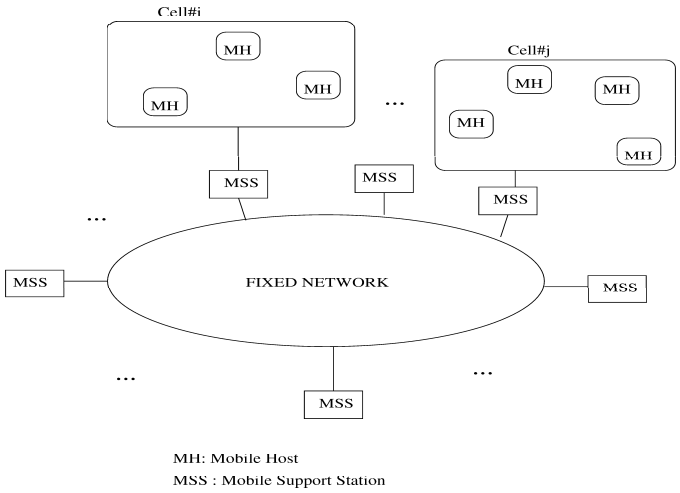

Figure 2: A general architecture of a mobile computing system

4 Utilization of Association Rules in Prefetching, Cache Replacement, and Scheduling

A typical architecture for mobile computing systems inspired from [IB94] is depicted in Figure 2. 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*.

We assume that there is a fixed set of data items that is periodically broadcast to mobile clients. Mobile clients issue requests for data items when the required items are not in the current broadcast set. The data server responds to client requests by placing the requested items on the broadcast disk. The server should carefully decide what portion of the broadcast disk will be dedicated to requests and what portion will be used for the fixed set of broadcast items. The portion dedicated to client requests is called pull bandwidth [AFZ97].

The following subsections discuss the automated use of association rules for prefetching, cache replacement, and scheduling of broadcast items.

4.1 Rule-based Prefetching and Cache Replacement Strategies

Both prefetching and cache replacement strategies exploit the association rules obtained by mining the broadcast history. The set of data items inferred by the association 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 Figure 3.

In order to perform rule-based prefetching and cache replacement, the mobile clients should be aware of the association 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. Broadcasting of rules together with the data items induces an overhead on the broadcast disk since the size of broadcast disk increases with the addition of association rules. Usually the number of the rules in the system is much smaller than the number of broadcast items, therefore the induced overhead turns out to be negligible. When a mobile client enters a cell, it fetches the current rule set in its new cell at the time association rules are broadcast. This also induces an overhead on mobile clients which

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

Figure 3: Construction of the set $inferred\_items$

if *bcast_item* is not in cache **then**
    if *bcast_item* is in $inferred\_items - cached\_items$ **then**
        $pvalue = rule\_conf(bcast\_item)$
    else
        $pvalue = access\_prob(bcast\_item) \times bcast\_time(bcast\_item)$
    endif
    if *minimum pvalue of cached_items* $> pvalue$ **then**
        perform prefetching
endif

Figure 4: Prefetching algorithm.

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 [IB92]:

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

The problem of prefetching broadcast items is discussed in [AFZ96]. Prefetching is useful when the data items to be used subsequently are already on the air, ready to be retrieved. We suggest that association 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 association rules. The server uses the rules to schedule the broadcast requests and clients use the rules to do prefetching. Clients consider 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 required data to appear on the broadcast when it is actually requested.

Rule-based prefetching strategy can be used together with the other prefetching methods. A prefetching method called PT that takes the broadcast environment into account was proposed by Acharya et al. in [AFZ96]. PT considers both the access probabilities of a data page and the time that the page will be broadcast again for prefetching. PT computes the 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 calculated value of a page is called its pt value and this value is used for cache replacement. 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. Rule-based prefetching strategy can be used together with the PT heuristic. Assume that we want to calculate the pt value of a cache item i. If item i is supported by a rule r with confidence c, then we assign value c for the pt value of i. This is a reasonable heuristic since the confidence of a rule gives us statistical information about how often the rule correctly predicts the requests for an item. A hybrid algorithm that combines rule-based prefetching and PT heuristic of Acharya et al. is provided in Figure 4. As described in the algorithm, prefetching is performed for data items that are not cache resident. Prefetching is performed if the prefetch value, i.e., $pvalue$ of the data item on broadcast is greater than the minimum $pvalue$ among the cache resident data items. The $pvalue$ of a data item is the confidence of the rule that inferred that item (denoted by $rule\_conf(bcast\_item)$) if the item is in the list of $inferred\_items$. Otherwise, $pvalue$ is set to the pt value as explained in [AFZ96].

Cache replacement is an important issue for mobile clients considering the limitations on wireless bandwidth and cache capacity of mobile computers. Clients can use the association 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 Figure 5. The algorithm first determines the data items that will probably be requested in the near future by using both the association rules and the last n requests issued by the client. 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 association rules. Another 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 the advantages of both conventional and rule-based cache replacement strategies. Some caching strategies for broadcast environments, namely PIX and LIX, were proposed by Acharya et al. in [AAFZ95]. The broadcast environment assumed by Acharya et al. consists of multiple broadcast disks with different sizes. Sizes of the disks determine their speed, i.e., disks with smaller sizes are faster than the larger disks. PIX takes into account both the probability (P) of access of the cache items and their frequency (X) of broadcast. The acronym PIX stands for P inverse X since the PIX value of a cache item is P/X. LIX is an approximation to PIX and it is an adaptation of LRU(Least Recently Used) cache replacement strategy to multiple disks. PIX can be used together with our rule-based cache replacement strategy by setting the probability of access value of a cache item in $inferred\_items$ to the confidence of the rule that inferred it, as in

if $cached\_items - inferred\_items = \emptyset$ **then**
    for the data items in $inferred\_items \cap cached\_item$ **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 PIX values
endif

Figure 5: Cache replacement algorithm.

the rule-based prefetching strategy.

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 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.

The first approach considers the temporal order of the requests, and the items inferred as a result of 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., data items inferred by a rule with a high confidence has higher priority than the items inferred by a rule with a lower confidence. The item with the lowest priority is replaced first.

4.2 Rule-based Scheduling of Data Broadcast Requests

The association rules extracted by mining the broadcast history can also be utilized in scheduling of broadcast requests. Our intuition in proposing this claim is based on the principal of proximity of requests. Let's say that we are given a set of data items $I = \{i_1, i_2, ...i_m\}$ which are requested frequently over the past in the same time frame. If we have an association rule $S_1 \Rightarrow S_2$ where $S_1, S_2 \subset I$, $S_1 \cup S_2 = I$, and $S_1 \cap S_2 = \emptyset$, it tells us with high confidence that upon the request of the items in S_1, the items in S_2 will also be requested in the same time frame. A very good example of this case is the request of html documents. A user who has requested a document will follow the links in these documents as well. Situations like this can be captured in association rules. Another example that can be given here is the processing of continuous spatial queries. Continuous spatial queries are issued over spatial data and the results may change depending on the user's location. A mobile user in a car, who has issued a continuous spatial query asking for the nearby hotels could also be interested in nearby restaurants. A problem that might arise here is that the user who is interested in the hotel information may leave its current cell. During the handoff process[3], the information about the user, such as the near past requests of him/her should be transferred to the new server, so that the new server can be aware of the request patterns of that user.

The current set of requests is preprocessed before running the data mining algorithms by partitioning the user requests with respect to user identifiers as shown in Table 2. The inferencing

[3]Handoff process refers to the jobs that need to be performed when a user leaves a cell and enters another one.

Users	Requested Items
$uscr_1$	u w x
$user_2$	v y
...	...
$user_n$	x u z

Table 2: User-based partitioning of requests.

Rule Name	Rule
$rule_1$	w x $\Rightarrow$ y z
$rule_2$	u $\Rightarrow$ v
...	...
$rule_m$	y $\Rightarrow$ x

Table 3: Current association rules.

mechanism should examine the current set of requests and predict the future requests of users. For example if we have the list of requests u w x by $user_1$ as shown in Table 2 and $rule_1$ as shown in Table 3, then we should put data items y and z in the next schedule as a result of the inference mechanism because requests w and x have occurred and $rule_1$ tells us that if a user requests data items w and x then he/she will also request data items y and z in the near future with a certain confidence.

The outline of our scheduling mechanism for determining which items to place in the broadcast schedule is provided in Figure 6. The scheduling mechanism, which is window-based, can be described as follows. Assume that the mobile computing system starts functioning, and new requests arrive consecutively. The scheduling program starts scanning the requests and tries to find the matching items in the current request window to the association rules set. The inference algorithm shown in Figure 7 does the matching for the current request window. The new broadcast schedule is constructed by adding the items that are matched to some rules for the current request window. The window slides one step and inferencing is performed again. In order to avoid multiple matching of the rules, we use a method based on marking already fired rules, and counting the number of times that the rules are considered as marked.

In case we have temporal information about the relative issuing of the broadcast requests and the data mining algorithms allow the mining of temporal association rules, we may also use the temporal information for scheduling. A possible example of temporal information is: "user1 requested $item_1$ and two time steps later he/she requested $item_2$". In this paper, we do not deal with temporality issues. The only temporal information we assume to have is the timestamps of the requests which is used to derive the sequence of requests.

While the server is broadcasting the data items using the association rules extracted from the broadcast history, the client performs the following steps to get the data item it needs:

1. It first looks at its cache. If the required data item is in the cache, then the client directly reads it from there.

2. Otherwise, it checks the index on air to see whether the data item will be broadcast. If the

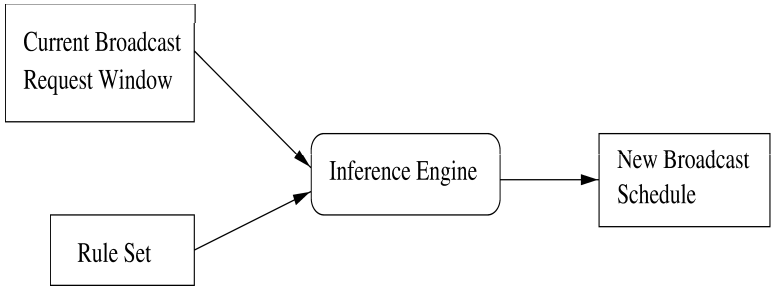

Figure 6: Scheduling Broadcast Requests with Association Rules.

initialize all the *association rules* as *unmarked*
initialize the *counters* of all *association rules* to 0
for each *association rule* **do**
    if the *association rule* is *unmarked* **then**
        if there is a *match* for the rule in the current *request window* **then**
            mark the rule
            set *rule counter* to *0*
            add the items to the *broadcast schedule*
        endif
    else
        increment the *counter* of the rule
        if *counter* is bigger than the *window size* **then**
            set the rule as *unmarked*
        endif
    endif
endfor

Figure 7: Inference algorithm.

item is in the schedule then the client waits for the item to arrive.

3. The client requests the item if it is not in the current broadcast schedule.

Conflicts among the association rules may arise when the rules with the same head are matched. Conflict resolution among such rules can be handled by either:

- choosing the rule with the highest confidence, or
- scheduling both of the rules in the order of confidence.

The second approach seems more reasonable if both of the rules are considered to be useful.

Another problem arises when we have some association rules of the form: $u \Rightarrow v$ and $y \Rightarrow v$. When both of the rules are matched, we may either put the item v only once in the schedule or we may wait for some time for scheduling v for the second time. This issue is related to periodic broadcasting of items.

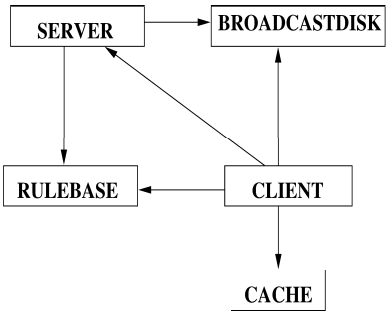

Figure 8: Object Relationship Diagram of the Simulation Program

5 Simulation and Experimental Results

We have implemented the data mining algorithms and the rule based scheduling and prefetching mechanisms to show the effectiveness of the proposed methods. A web log is 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 5.1 and Section 5.2, respectively.

5.1 Simulation Model

The whole system consists of two independent parts:

- rule extraction part, and
- broadcast simulation part.

Rule extraction part performs the task of extracting the association rules from the web log. Rule sets with different minimum confidence and support requirements can be constructed by the rule extraction program. The resulting association rules are written to a file in a specific format to be read later by the broadcast simulation program. Both parts of the system have been implemented using an object oriented programming language, C++.

The main objects of the broadcast simulation program and their relationship are depicted in Figure 8. The main objects are the *SERVER, CLIENT, CACHE, BROADCASTDISK*, and *RULEBASE*. Both the server and client objects have a common rule base and a broadcast disk. The client object interacts with the server object by sending broadcast requests to it. The server and client objects interact with the broadcast disk by placing new items to the broadcast disk and fetching data items from the broadcast disk, respectively. The rulebase object is constructed by reading the rule file generated by the data mining program and is shared by client and server objects. The rulebase object is contacted by the client and server objects for prefetching and broadcast scheduling, respectively. The client has a cache object to store a limited amount of requested items.

The architecture of our system is depicted in Figure 9. In this architecture, we divide the whole history into two equal parts. The first half of the history is fed into the data mining program to be used for extracting association rules. The resulting rule set is fed into the rule base which is then used for scheduling, cache replacement, and prefetching. The second part of the history is used for simulating the requests of a client. Rules are extracted out of the first half of the broadcast history

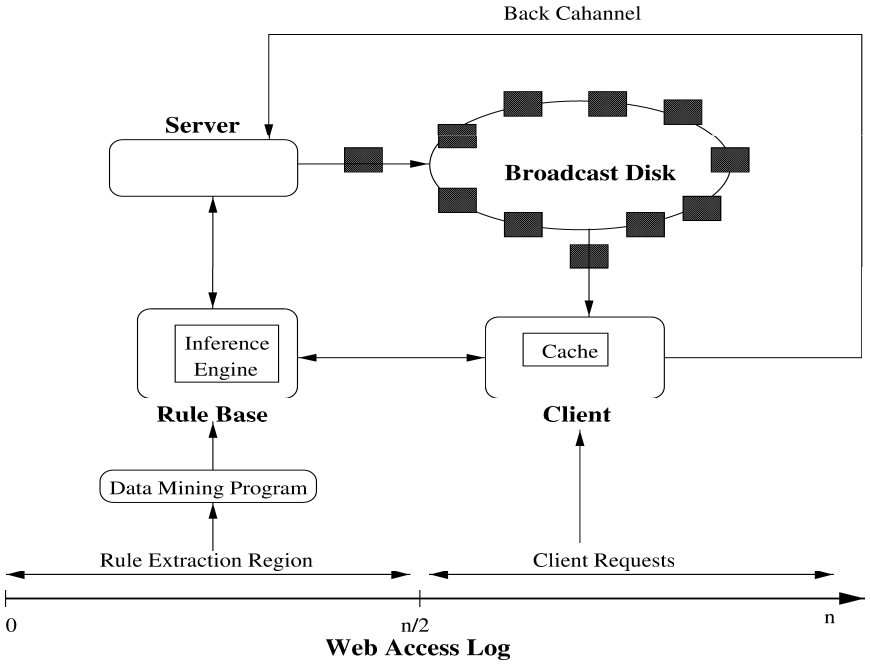

Figure 9: System Architecture

using set oriented association rule extraction algorithms which are very similar to the well known data mining algorithm called Apriori [AIS93a].

As we present in Figure 9, 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 requests are evaluated by the server and the requested data items are placed on the broadcast disk. However, to simulate the server workload, broadcast disk spins for a random amount of time before the server actually places the first the data item on the disk. The client continuously scans the disk after issuing the request until it accesses the requested item.

We assumed 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. There is no indexing or clustering in our broadcast structure. We assume that each bucket contains only one data item which consists of an identifier and the real data. We also assume a pure pull based broadcast system where only the requests of the clients are broadcast.

5.2 Experimental Results

We implemented and run both gap based and window based rule mining algorithms on our web log. The experiments were performed on a Pentium II MMX with a main memory of 64M, running Solaris operating system. We extracted rules with different support values and evaluated their effect on the performance. Our performance metric is the *client cache hit ratio*. An increase in the cache hit ratio is an indication of how useful the proposed methods are. If an increase in the hit ratio of the mobile client cache is achieved, the mobile client would not be subject to frequently performing an expensive look up operation over 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, therefore, lead to saving of the scarce energy sources of the mobile computers and reduction of the server load.

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
GAP SIZE	gap between the transactions
WINDOW SIZE	size of the window for rule extraction and inferencing

Table 4: Main Parameters of Our System.

Some statistics about the web log which was used for the experiments are as follows:

- the total number of requests is around 116.000,
- the requests are distributed among 7600 different data items,
- the total number of users that issued the requests is 11.600.

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 times are not significant. A detailed discussion on the running time of the rule extraction part is provided in Section 5.2.1 The running time of the rule checking algorithm is not significant either since the number of rules is not so large. We have also observed that the optimum number of rules for the best cache hit ratio does not exceed 150.

The basic parameters of our system are presented in Table 4. 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. The gap size of 600 seconds was observed to give the best results for rule extraction. Similarly, we assumed a window size of 100 items, as this value produced reasonable results. The results obtained by varying the values of the gap size and the window size are not presented due to space limitations.

5.2.1 Results for Gap Based Methods

Gap based methods for mining the web log assume that after a certain threshold value, the clients start a new transaction (i.e., internet session). This threshold is called the gap between two transactions and measured in seconds. As we have just mentioned, the gap value chosen for our experiments is 600 seconds which is 5 minutes.

The results we obtained for the CPU time required to extract rules with different minimum support values using gap based methods are shown in Figure 10. As can be observed from the figure, CPU time decreases sharply when the support value is changed from 0.2 to 0.4 since the number of rules found for a minimum support value of 0.2 is much higher than that with minimum support value 0.4. This considerable decrease continues until the point where the support is 0.6 and then levels as the support value is increased more. Even for very low support values like 0.2, the CPU time required for mining the web log does not exceed a few minutes.

The experimental results for the association rules extracted using a gap based approach are depicted in Figure 11. The support value of the rules was kept constant at 1.0 and the confidence was varied from 0 to 100. The values of cache hit ratio increase (CHRI) in response to the rule based prefetching techniques are depicted in Figure 11. We observed that a confidence value in the

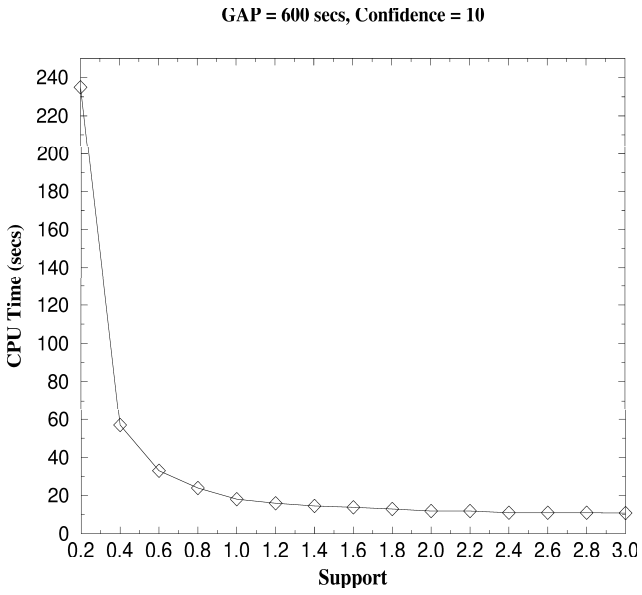

Figure 10: CPU time spent as a function of the support value

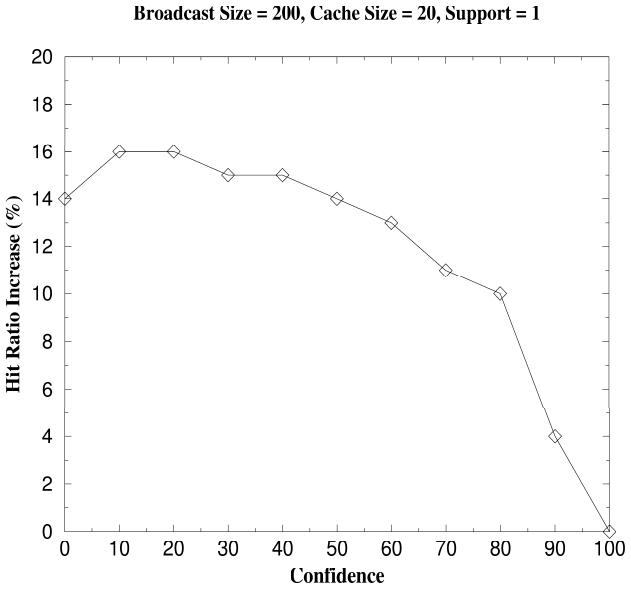

Figure 11: Increase in the cache hit ratio as a function of the confidence value

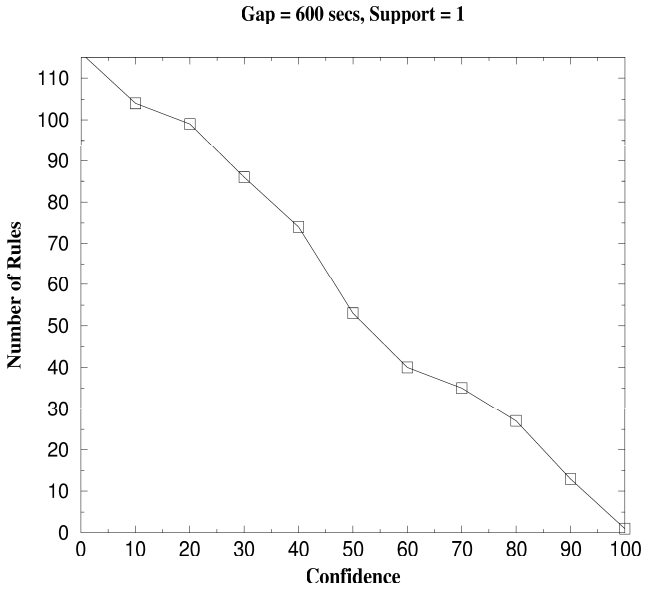

Figure 12: Number of rules as a function of the confidence value

range 10 to 20 gives the best value for CHRI. A further increase in the confidence value results in a decrease in CHRI value. An increased confidence decreases the number of rules, and thus reduces the effectiveness of the rule based methods. It should also be noted that a very low confidence does not mean the best performance as can be observed for the 0-confidence case, since the rules in this case do not reflect the client access patterns and mislead the inferencing mechanism. Figure 12 depicts the number of rules for different confidence values.

Figure 13 displays CHRI results under varying values of the minimum support. The corresponding numbers of rules for these support values are displayed in Figure 14. As can be observed from Figure 13, for very low support values, like 0.2 and 0.4, although the number of extracted rules is very high, CHRI values are not as high as those obtained with the support values in the range from 0.8 to 1.8. For the support values greater than 1.8, increasing the support value leads to a decrease in CHRI. Therefore we can conclude that the best performance could be achieved with a minimum support value that is not too small or too large.

Cache size is another important parameter that we need to evaluate whose impact on the performance. The results we obtained for CHRI under different cache sizes are displayed in Figure 15. As we can observe from the figure, in general, rule based prefetching and cache replacement methods are more effective for small cache sizes. Since the mobile devices are characterized by limited cache size, we could state that our methods are effective especially for mobile computers rather than powerful fixed computers. The cache size is measured in terms of the number of items, assuming that the items retrieved are html documents, possibly with images. A cache size of 20 seems to be a reasonable value.

5.2.2 Results for Window Based Methods

Window based methods for mining the web log assume that a certain number of data requests, called *window*, constitutes a transaction. A sliding window approach is used to mine the web log. Each window is assumed to be a transaction, therefore the number of transactions is much higher

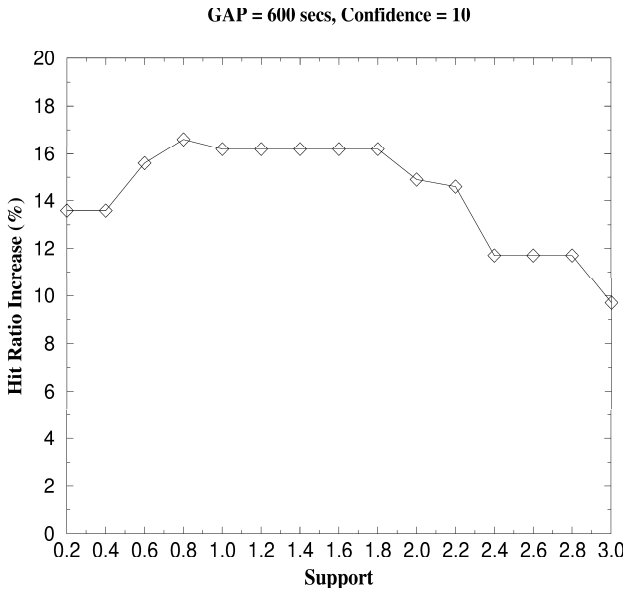

Figure 13: Increase in the cache hit ratio as a function of the minimum support value

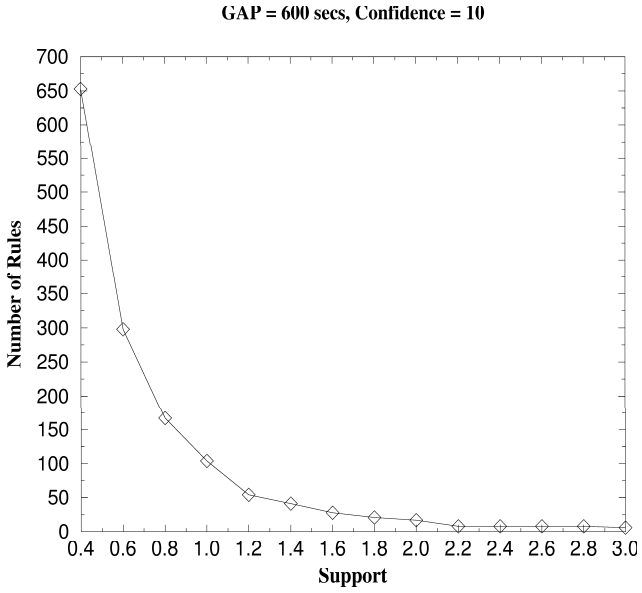

Figure 14: Number of rules as a function of the minimum support value

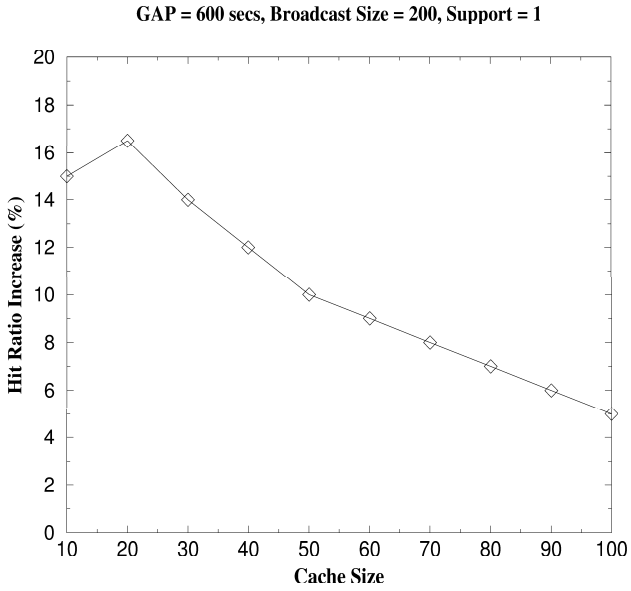

Figure 15: Increase in the cache hit ratio as a function of the cache size

than that in the gap based methods. A higher number of transactions means a higher execution time compared to the gap based methods. In the sliding window approach, a sequence of consecutive windows overlap depending on the window size; therefore, for a given minimum support value, the number of rules obtained is higher than that with the gap based methods. This situation makes the window based approaches to reach to optimum performance in terms of CHRI for support values around 2.0 instead of 1.0 as in gap based methods (see Figure 16). The results for the CPU time required for mining the web log using window based methods are shown in Figure 17. The CPU time spent stays constant at around 10 seconds for the support values in the range 1.4-4.0. As the support value goes below 1.4, the required CPU time increases sharply. The time required for mining could be considered to be not significant for the support values around 2.0 which lead to the best CHRI performance.

The experiment was repeated with different cache sizes and confidence values. The results are not displayed here since the performance pattern obtained for CHRI was not sensitive to varying the values of these parameters.

6 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 exploited the methods for the data mining problem of finding association rules and applied the association rule extraction algorithms to the broadcast history of a mobile database system. We have proposed a new method of scheduling broadcast requests in mobile environments using the association rules obtained by mining the broadcast history. The resulting association rules mimic the future requests of mobile clients. An inference mechanism is used to determine the future items to be scheduled for broadcasting, given the current request window and association rules derived from the broadcast history. Our expectation with this approach is to decrease the delay experienced by the clients while waiting for the required data items, since

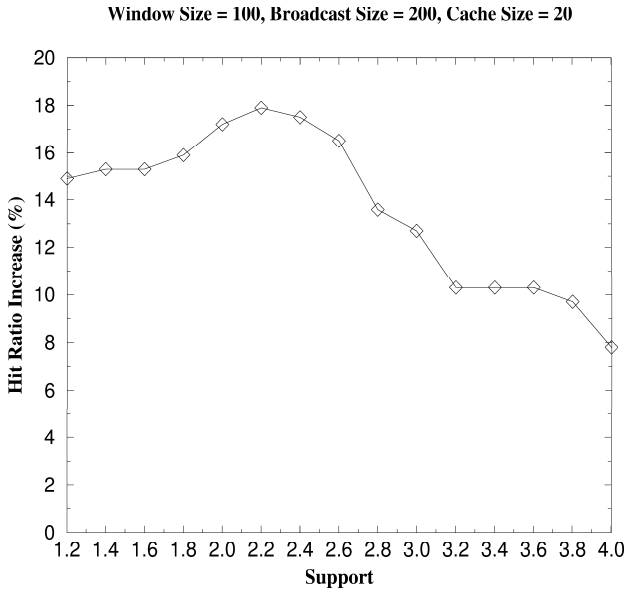

Figure 16: Increase in the cache hit ratio as a function of the support value for window based methods

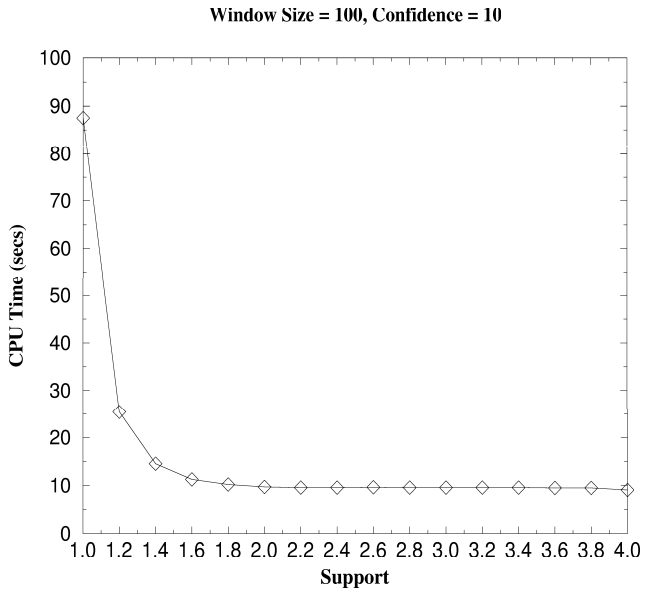

Figure 17: CPU time spent as a function of the support value

the items that are likely to be requested in the near future will be placed in the broadcast schedule.

We have also aimed to show that association rules are beneficial for prefetching of data items by mobile clients. A cache replacement mechanism for mobile computers has been proposed which utilizes the association rules to determine the items to be replaced in the cache. In order to exploit association rules for prefetching and cache replacement, mobile clients need to have an access to the current association rules that exist in the system. Servers can handle this problem by periodically broadcasting the current association 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. It has been observed in the experiments that a considerable increase in the cache hit ratio can be obtained when the rule based cache replacement and prefetching techniques are used together. The best results in terms of the cache hit ratio have been obtained for small cache sizes of clients. Considering the fact that mobile clients are characterized by limited cache size, we can say that our methods of predictive push, cache replacement and prefetching are useful especially in mobile environments. The performance results have been obtained by mining the web log without any preprocessing. We think that the improvement in the performance of the system can be even more by preprocessing the history and eliminating the irrelevant data items and transactions.

In this work, we have not dealt with the temporality issues. However, temporal information can also be exploited for scheduling broadcast requests. Temporal association 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 [BWJL98]. Temporal association 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 association rules in broadcasting is left as a future work.

References

[AAFZ95] Swarup Acharya, Rafael Alonso, Michael Franklin, and Stanley Zdonik. Broadcast disks: Data management for asymmetric communication environments. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, San Jose, California, June 1995.

[AFZ96] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Prefetching from a broadcast disk. In *Proceedings of the 12th International Conference on Data Engineering(ICDE'96)*, New Orleans, LA, February 1996.

[AFZ97] Swarup Acharya, Michael Franklin, and Stanley Zdonik. Balancing push and pull fro data broadcast. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, Tucson, Arizona, May 1997.

[AGI+92] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. of the 18th Int'l Conference on Very Large Databases*, pages 560–573, Vancouver, August 1992.

[AIS93a] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering, Special issue on Learning and Discovery in Knowledge-Based Databases*, 5(6), December 1993.

[AIS93b] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Washington D.C., May 1993.

[AP95] R. Agrawal and G. Psaila. Active data mining. In *Proc. of the 1st Int'l Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, August 1995.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September 1994.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the Int'l Conference on Data Engineering (ICDE)*, Taipei, Taiwan, March 1995.

[AWH92] Alexander Aiken, Jennifer Widom, and Joseph M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *Proceedings of ACM-SIGMOD Conference on Management of Data*, pages 59–68, San Diego, California, June 1992.

[BWJL98] Claudio Bettini, X. Sean Wang, Sushil Jajodia, and Jia-Ling Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2), 1998.

[CHNW96] David Wai-Lok Cheung, Jiawei Han, Vincent Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pages 106–114, 1996.

[CNFF96] David Wai-Lok Cheung, Vincent Ng, Ada Wai-Chee Fu, and Yongjian Fu. Efficient mining of association rules in distributed database. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 1996.

[Day88] Umeshwar Dayal. Active database management systems. In *Proceedings of the Third International Conference on Data and Knowledge Bases*, pages 150–169, Jerusalem, June 1988.

[HS93] M. Houtsma and A. Swami. Set-oriented mining of association rules. Technical report, IBM Almaden Research Center, San Jose, California, October 1993.

[IB92] T. Imielski and B. R. Badrinath. Querying in highly distributed environments. In *Proceedings of the 18th VLDB Conference*, Columbia, Canada, 1992.

[IB94] Tomasz Imielinski and B. R. Badrinath. Mobile wireless computing: Challenges in data management. *Communications of the ACM*, pages 19–27, October 1994.

[IVB97] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering*, 9(3), 1997.

[MJHS96] Bamshad Mobasher, Namit Jain, Eui-Hong Han, and Jaideep Srivastana. Web mining: Pattern discovery from worl wide web transactions. Technical Report 96-050, Department of Computer Science, University of Minnesota, September 1996.

[RS95] R. Agrawal R. Srikant. Mining generalized association rules. In *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[SRB97] Konstantinos Stathatos, Nick Roussopoulos, and John S. Baras. Adaptive data broadcast in hybrid networks. In *Proceedings of the 23rd VLDB Conference*, Athens, Greece, 1997.

[SUC98] Yucel Saygin, Ozgur Ulusoy, and Sharma Chakravarthy. Concurrent rule execution in active databases. *Information Systems*, 23(1), 1998.

[SW98] A. Prasad Sistla and Ouri Wolfson. Minimization of communication cost through caching in mobile environments. *IEEE Transactions on Parallel and Distributed Systems*, 9(4), April 1998.

[ZFAA94] S. Zdonik, M. Franklin, R. Alonso, and S. Acharya. Are "Disks in the Air" just pie in the sky. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.