Cache-Miss-Initiated Prefetch in Mobile Environments[*]

Hui Song and Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802
{hsong, gcao}@cse.psu.edu

Abstract

Prefetching has been widely used to improve system performance in mobile environments. Since prefetching also consumes system resources such as bandwidth and power, it is important to consider the system overhead when designing a prefetching scheme. This paper proposes a cache-miss-initiated prefetch (CMIP) scheme to address this issue. The CMIP scheme relies on two prefetch sets: the always-prefetch set and the miss-prefetch set. The always-prefetch set consists of data that should always be prefetched if possible. The miss-prefetch set consists of data that are closely related to the cache-missed data item. When a cache miss happens, instead of sending an uplink request to ask for the cache-missed data item only, the client also requests for the data items which are within the miss-prefetch set. This reduces not only future cache misses but also the number of uplink requests. Note that the client can ask for several data items in one uplink request with little additional cost. We propose novel algorithms to mine the association rules and use them to construct the two prefetch sets. Simulation results show that our CMIP scheme can greatly improve the system performance in terms of improved cache hit ratio, reduced uplink requests, and negligible additional traffic.

1. Introduction

Many studies show that data broadcasting is an effective dissemination technique in mobile environments[2, 3, 14, 18]. With this technique, clients access data by simply monitoring the broadcasting channel until the requested data items appear on the channel. The data broadcasting model exploits the asymmetric nature of the wireless channel, where more bandwidth is available for the downlink (server-to-client) but less bandwidth is available for the uplink (client-to-server). In addition, the broadcasting model is scalable since the bandwidth consumption does not depend on the number of mobile clients in the system.

Although data broadcasting has advantages, it also introduces some problems. For example, waiting for a data item to appear on the broadcast channel may increase the query delay. One way to alleviate this problem is to cache frequently accessed data on the client side. In this way, the clients can serve many requests from the local cache without sending uplink requests. This not only reduces the average query delay but also reduces the uplink and downlink bandwidth consumption.

To further reduce reduce the query delay and improve the cache hit ratio. Prefetching techniques can be used. Prefetching has many advantages in mobile environments since wireless networks, such as wireless LANs or cellular networks, support data broadcasting. When the server broadcasts data on the broadcast channel, clients can prefetch their interested data without increasing bandwidth consumption. However, if the requested data item is not prefetched earlier, the client has to send an uplink request to ask for it when a query comes. This not only increases the query delay but also increases the uplink bandwidth consumption. Since the uplink bandwidth is a limited and precious resource in mobile environments, prefetching should be used frequently. However, prefetching also consumes many system resources. For example, prefetching can make use of the broadcast channel but the clients still need to consume power to process the prefetched data. Since most mobile clients are powered by batteries, prefetching the right data is critical when designing prefetching schemes.

In this paper, we propose a cache-miss-initiated prefetch (*CMIP*) scheme to help mobile clients prefetch the right data. The CMIP scheme relies on two prefetch sets: the *always-prefetch set* and the *miss-prefetch set*.

[*] This work was supported in part by the National Science Foundation (CAREER CCR-0092770 and ITR-0219711).

The always-prefetch set consists of data that should always be prefetched if possible. The miss-prefetch set consists of data that are closely related to the cache-missed data item. When a cache miss happens, instead of sending an uplink request to ask for the cache-missed data item only, the client also asks for several other items, which are within the miss-prefetch set. In this way, we can prevent some future cache misses and reduce the number of uplink requests. Note that the client can ask for several data items in one uplink request with little additional cost. In this paper, we propose a novel algorithm to mine the association rules and use them to construct the two prefetch sets in two steps. First, we mine the access history of the clients to obtain the association rules. Second, we use the *confidence* parameter of the rules to construct the prefetch sets. Detailed experiments are used to evaluate the performance of the proposed scheme. Compared to the UIR scheme [7] and the UIR scheme without prefetch, our CMIP scheme can greatly improve the system performance in terms of improved cache hit ratio, reduced uplink requests and negligible additional traffic.

The remainder of the paper is organized as follows. In Section 2, we present the cache-miss-initiated prefetch (CMIP) scheme. Section 3 evaluates the performance of our CMIP scheme. Section 4 reviews the previous work on prefetching and association rules mining. Section 5 concludes the paper.

2. The Cache-Miss-Initiated Prefetch Scheme

Our scheme is motivated by two observations. First, a mobile client may query some data items frequently. If the requested data items are in the cache, the mobile clients can save bandwidth and power by serving the quests without sending uplink requests. Therefore, it is important to always prefetch frequently requested data. Second, data items queried during a period of time are related to each other. Hence, cache misses are not isolated events; a cache miss is often followed by a series of cache misses. For instance, suppose a meaningful information consists of four data items, $\{i_1$, i_2, i_3, and $i_4\}$. To get the information, a client needs to access all of them. Now suppose a cache miss happens when the client is accessing data item i_1. The client can expect a series of cache misses because of sequential accesses to i_2, i_3, and i_4. Thus, when the cache miss of item i_1 happens, the client should also prefetch items i_2, i_3, and i_4 from the server. Because of the cache miss of item i_1, the client will need to send an uplink request. Piggybacking the requests for items i_2, i_3, and i_4 in the uplink request has little bandwidth overhead but saves 3 future uplink requests. The query delay for getting the information is also reduced. Based on these two observations, we propose to mine the access history of the clients to find the relationships among data items and use them to prefetch.

Association rule based data mining techniques [4, 5] have been proposed to find relationships among data items by analyzing a large collection of data. We propose to use these techniques to discover the association rules in the access history and use the rules to construct two prefetch sets, the *always-prefetch set* and the *miss-prefetch set*, motivated by the above two observations respectively. As the name suggests, the always-prefetch set consists of data that should always be prefetched if possible. The miss-prefetch set consists of data items that will be prefetched after a cache miss. In the following sections, we will present our cache-miss-initiated prefetch (CMIP) scheme based on these two prefetch sets.

2.1. Mining the Access Trace to Obtain Association Rules

In this section, we present the algorithm which can generate association rules from the access trace. Subsection 2.1.1 formalizes the problem. The formalization is inspired from [4] and [5]. Subsection 2.1.2 and 2.1.3 present the algorithm which generates association rules.

2.1.1. The Problem Statement. Suppose a client's access trace S consists of a set of consecutive parts: $\{p_1, p_2, \ldots, p_i, \ldots, p_n\}$. Let $\Gamma = \{i_1, i_2, \ldots, i_m\}$ denote the set of data items accessed by the client. Let S_i denote the data items contained in part p_i. S_i is called a *session* and $S_i \subset \Gamma$. We say that a session S_i *contains* X if $S_i \supseteq X$, where $X \subset \Gamma$. An *association rule* is defined as an implication of the form $X \Longrightarrow Y$, where $X \subset \Gamma, Y \subset \Gamma$, and $X \cap Y = \emptyset$. X is called the *antecedent* of the rule and Y is called the *consequent*.

A set of data items is called an *itemset*. The number of data items in an itemset is called the *size* of the itemset. An itemset of size k is called a *k-itemset*. The *support* of an itemset X, *support(X)*, is defined as the percentage of sessions which contains X in the client's access trace S. While the *support* of an association rule R, *support(R)*, is defined as the support of the itemset, which consists data items in both the antecedent and the consequent of the rule. For instance, the support of an association rule $R : X \Longrightarrow Y$ is

$$support(R) = support(\{X, Y\})$$

The *confidence* of an association rule R, *confidence(R)*, is defined as the support of the rule divided by the support of the antecedent. For example, the confidence of the association rule $R : X \Longrightarrow Y$ is

$$confidence(R) = \frac{support(\{X, Y\})}{support(X)} * 100\%$$

Table 1. Notations

k-itemset	an itemset with k items.
F_k	the set of frequent k-itemsets (those with minimum support).
f_i, f_j	any of the frequent (k-1)-itemsets within F_{k-1}.
$f_j.item_m$	itemset f_j's $m-$th item.
c	a new frequent k-itemset got by combining a frequent (k-1)-itemset with an item.

Given an access trace S, the problem of mining association rules is to find all the association rules that have support and confidence greater than the user-specified minimum support (*minsup*) and minimum confidence (*minconf*) respectively.

The problem of discovering association rules can be decomposed into two subproblems [5]:

1. Find all the itemsets with minimum support: $support(X) \geq minsup$. Itemsets with minimum support are called *frequent itemsets*.
2. Use the frequent itemsets to generate association rules with minimum confidence.

2.1.2. The Algorithm to Generate Frequent Itemsets. In this section, we present the algorithm for generating frequent itemsets from the client's access trace, which is similar to [4]. Table 1 shows the notations used in the algorithm. Figure 1 illustrates the main steps of our frequent itemset generation algorithm. The algorithm accepts an access trace S and a minimum support (*minsup*) as parameters. Line 1 is the first step. In this step, S is analyzed to generate the frequent 1-itemsets. This is done by calculating the support of each data item and comparing the support to the minimum support. Each Data item which has minimum support forms a frequent 1-itemset.

The second step is to find all the frequent 2-, 3-, ..., k-itemsets. This is done by using a loop shown from line 3 to line 21. Each iteration of the loop, say iteration k, generates frequent k-itemsets based on the (k-1)-itemsets generated in the previous iteration. The loop stops when no larger frequent itemsets can be generated. Inside the second step, lines 3-15 generate all the new candidate frequent k-itemsets out of the frequent (k-1)-itemsets. Lines 16-19 remove those candidate frequent k-itemsets which do not have minimum support. The algorithm returns all the frequent itemsets in line 22.

```
1) $F_1 \leftarrow \{$frequent 1-itemsets$\}$
2) $k \leftarrow 2$
3) while $F_{k-1} \neq \emptyset$ do
4)     $F_k \leftarrow \emptyset$ /*initialize $F_k$*/
5)     for each itemset $f_i \in F_{k-1}$ do
6)         for each itemset $f_j \in F_{k-1}$
7)             if $f_i.item_1 = f_j.item_1$&
                   $f_i.item_2 = f_j.item_2$&...
8)                 & $f_i.item_{k-2} = f_j.item_{k-2}$&
                   $f_i.item_{k-1} < f_j.item_{k-1}$
9)             then $c \leftarrow f_i \cup \{f_j.item_{k-1}\}; F_k \leftarrow F_k \cup \{c\}$
10)                for each (k-1)-subsets $s \in c$ do
11)                    if $s \notin F_{k-1}$
12)                    then $F_k \leftarrow F_k - \{c\}; break$
13)                end
14)        end
15)    end /*loop generates new candidate sets*/
16)    for each itemset $f_i \in F_k$ do
17)        if $support(f_i) < minsup$
18)        then $F_k = F_k - \{f_i\}$
19)    end /*loop removes candidates without minsup*/
20)    k++
21) end
22) return $\cup_k F_k$
```

Figure 1. The algorithm to generate frequent itemsets

2.1.3. The Algorithm to Generate Association Rules. The proposed algorithm generates association rules based on the frequent itemsets. Only two simple kinds of rules are generated for our purpose: one is of the form '$\Longrightarrow i_j$' and the other is of the form '$i_j \Longrightarrow Y$', where $i_j \in \Gamma$ and $Y \subset \Gamma$. Table 2 shows the notations used in our algorithm. Figure 2 illustrates the main idea of the algorithm. The algorithm accepts the frequent itemsets and a minimum confidence (*minconf*) as parameters. The rules are generated in two steps. Lines 1-6 are the first step. This step generates the first kind of rules, which are of the form '$\Longrightarrow f_j.item_1$', where $f_j.item_1 \in f_j$ and $f_j \in F_1$. These rules are generated from the frequent 1-itemsets F_1 in this way. For each 1-itemset f_j, the support of $f_j.item_1$, which is the only one data item contained in f_j, is compared with the minimum confidence. If the support of $f_j.item_1$ is not less than the minimum confidence, we generate a rule of the form: '$\Longrightarrow f_j.item_1$'. Note that the 1-itemset f_j here is also of minimum support (*minsup*), because all the 1-itemsets with less than minimum support have already been filtered in the frequent-itemset generating phase described in Subsection 2.1.2. In addition, the number of rules of the

Table 2. Notations

R	the set of association rules.
I	the set of items which could be the antecedent of a rule.
F_k	the set of frequent k-itemsets.
n	the largest size of the frequent itemsets.
f_j	any of the frequent k-itemsets within F_k.
$f_j.item_i$	itemset f_j's i-th item.

```
1) $R \leftarrow \emptyset; I \leftarrow \emptyset$
2) **for each** itemset $f_j \in F_1$
3)     $I \leftarrow I \cup \{f_j.item_1\}$
4)     **if** $support(f_j.item_1) \geq minconf$
5)     **then** $R \leftarrow R \cup \{" \Longrightarrow f_j.item_1"\}$
6) **end**
7) $k \leftarrow n$
8) **while** $F_k \neq \emptyset$ **and** $k > 1$ **do**
9)     **for each** itemset $f_j \in F_k$ **do**
10)         **for each** item $f_j.item_i \in f_j$
11)             **if** $f_j.item_i \in I$ **and**
                $\frac{support(f_j)}{support(f_j.item_i)} \geq minconf$
12)             **then** $R \leftarrow R \cup$
                $\{``f_j.item_i \Longrightarrow \{f_j - \{f_j.item_i\}\}"\}$
13)             $I \leftarrow I - \{f_j.item_i\}$
14)         **end**
15)     **end**
16)     k - -
17) **end**
18) **return** $R$
```

Figure 2. The algorithm to generate the association rules

form '$\Longrightarrow f_j.item_1$' is limited by the cache size. If there are too many such kind of rules, we stop generating this kind of rule after the cache size reaches. This situation suggests that we should use a larger cache size. The first step also initializes the set I. Set I is used to optimize the process of generating rules in the second step. Set I consists of data items which have potential to generate rules where they are the antecedent of the rules.

Lines 7-17 are the second step. In this step, the second kind of rules of the form '$i_j \Longrightarrow Y$', where $i_j \in \Gamma$ and $Y \subset \Gamma$, are generated from frequent itemsets of size larger than 1. For each frequent itemset, the rules are generated as follows. Of all the data items within the frequent itemset, one item becomes the antecedent of the rule, and all the other items become the consequent. Thus, a frequent k-itemset can generate at most k rules. For example, suppose $\{i_1, i_2, i_3, i_4\}$ is a frequent 4-itemset. It can generate at most four rules: $i_1 \Longrightarrow \{i_2, i_3, i_4\}$, $i_2 \Longrightarrow \{i_1, i_3, i_4\}$, $i_3 \Longrightarrow \{i_1, i_2, i_4\}$, and $i_4 \Longrightarrow \{i_1, i_2, i_3\}$. After the rules have been generated, their confidences are calculated to determine if they have the minimum confidence. Only the rules with the minimum confidence are kept in the rule set R. For example, for the rule $i_1 \Longrightarrow \{i_2, i_3, i_4\}$, we need to calculate its confidence: $conf = \frac{support(\{i_1, i_2, i_3, i_4\})}{support(\{i_1\})}$. If $conf \geq minconf$, the rule holds and it will be kept in the rule set R. Using this method, different frequent itemsets may generate rules with the same antecedent. For example, suppose there are two frequent itemsets: a frequent 4-itemset $\{i_1, i_2, i_3, i_4\}$ and a frequent 3-itemset $\{i_1, i_2, i_3\}$. Both of them can generate a rule for i_1. The rule generated by the frequent 4-itemset is '$i_1 \Longrightarrow \{i_2, i_3, i_4\}$' and the rule generated by the frequent 3-itemset is '$i_1 \Longrightarrow \{i_2, i_3\}$'.

Among the rules with the same antecedent, the one we need should have the largest itemset in the consequent of the rule. Note that a larger frequent itemset can generate a rule with the same antecedent but a larger consequent than that generated by a smaller frequent itemset. Thus, for a data item, the rules should be generated from the largest frequent itemsets first. If a rule has already been generated for a data item from a frequent itemset, say a k-itemset, one need not generate rules for the data item in any smaller frequent itemsets, such as (k-1)-itemsets, (k-2)-itemsets and so on. As in the above example, data item i_1 has two rules and the rule '$i_1 \Longrightarrow \{i_2, i_3, i_4\}$' has a larger itemset in the consequent than the rule '$i_1 \Longrightarrow \{i_2, i_3\}$'. Since $\{i_2, i_3, i_4\}$ contains $\{i_2, i_3\}$, the former rule has a larger view of i_1's relationships with other data items than the latter one. Thus, we only need the former rule for data item i_1 and do not need to generate rules, such as '$i_1 \Longrightarrow \{i_2, i_3\}$', from smaller frequent itemsets.

Based on this observation, in our algorithm, the process of rule generating starts by analyzing the largest frequent itemsets, F_n, first, and then F_{n-1}, F_{n-2}, ..., until reaching the frequent 2-itemsets F_2. In order to prevent the algorithm from generating rules for those data items which already have a rule, set I is used. Set I is initialized in the first step and it is used in the following way in our algorithm. Before generating a rule for a data item, say i_j, the data item i_j is checked to see if it is in set I. If i_j is found in set I, a rule is generated for i_j. At the same time, i_j is deleted from the set I. If i_j is not in set I, no rule will be generated. By using the set I, only one rule, which has the largest consequent, is generated for each data item.

2.2. Constructing Prefetch Sets

The two prefetch sets, *always-prefetch set* and *miss-prefetch set*, are constructed using the association rules generated above. The always-prefetch set is constructed from the first kind of rules which are of the form '$\Longrightarrow i_1$', by collecting the data items in the consequent of these rules. Since the data items in the consequent of such rules appear in most of the sessions, it is a good idea to always keep a fresh copy of them in the cache. By doing so, we can improve the cache hit ratio and reduce the uplink requests.

The miss-prefetch set is constructed from the second kind of rules which are of the form '$i_j \Longrightarrow Y$', where $i_j \in \Gamma$ and $Y \subset \Gamma$. A miss-prefetch set for the cache-missed data item is constructed when a cache miss happens. The miss-prefetch set is constructed in this way: among all the rules, the rule whose antecedent is the cache-missed data item is located and all the data items in the consequent of the rule form a miss-prefetch set of the cache-missed data item. For example, suppose data item i_1 is cache missed and we have a rule like this: '$i_1 \Longrightarrow \{i_2, i_3, i_4\}$'. The miss-prefetch set of data item i_1 is $\{i_2, i_3, i_4\}$.

2.3. The Cache-Miss-Initiated Prefetch (CMIP) Scheme

Our CMIP scheme prefetches data items based on the two prefetch sets. The following is a description of our CMIP scheme. Each client maintains an always-prefetch set. If the data items within the always-prefetch set are not in the cache, the client will prefetch them when they appear on the broadcast channel. If they have been updated by the server and the copies in the cache become invalid, the client prefetches them when they become available. Since the always-prefetch set is small, keeping it in the cache does not consume too many system resources.

When a cache miss happens, the miss-prefetch set for the cache-missed data item is constructed. In case of a cache miss, the client needs to send an uplink request to the server to ask for the cache-missed data item. Instead of requesting only the cache-missed data item, all the data items within the miss-prefetch set are requested. When the server receives the request, it broadcasts all these data items on the broadcast channel. The client will prefetch and store them in its cache. By piggybacking the request of items in the miss-prefetch set, the client can save future uplink requests and reduce the access latency.

The association rules, on which the two prefetch sets are based, are highly related to the client's access pattern and the client's access pattern may change from time to time. For example, at one time, the client is interested in stock quotes. After some time, the user may like to browse the news. When the client's access pattern changes, the relationships among data items also changes. Some new relationships may show up and some old relationships may disappear. Thus, the association rules should be re-mined and updated so that they are in accordance with the client's changing access pattern. In the CMIP scheme, we re-mine and update association rules periodically to keep them fresh. This is done by adding recent accesses to the access trace and cutting off the oldest part. After re-mining the association rules, the always-prefetch set is updated and the miss-prefetch sets for oncoming cache misses are constructed using the new rules.

We want to point out that, in our CMIP scheme, the data within the always-Prefetch set will never be replaced or deleted from the cache, since the client has a high probability to query them in the future. But we note one exception: when association rules are updated, some of them may become replaceable. For example, if a data item is within the old always-prefetch set, but not in the new one, it will be marked as replaceable.

To make our CMIP scheme adaptable, we need to decrease the number of prefetches when the power becomes low, since downloading a data item from the channel consumes power. Our strategy is to reduce the miss-initiated-prefetch level. In this case, we need to re-mine the association rules with a higher minimum confidence (*minconf*) parameter, and then the rule mining algorithm can generate fewer and smaller rules. As a result, the miss-prefetch set becomes smaller and we only need to prefetch fewer data items from the broadcast channel. Using this strategy, we can decrease the power consumption and only prefetch the most important data items.

3. Performance Evaluation

3.1. The Simulation Model and Parameters

To evaluate the performance of the proposed methodology, we compare the CMIP scheme with the UIR scheme [7] and the UIR scheme without prefetch (called NOPRE) under various workload and system settings. Our simulation is base on a simulation trace transformed from a real client-side web trace provided by Boston University [11].

3.1.1. The Simulation Trace. The web trace provided by Boston University [11] was collected for the purpose of understanding client requests for web documents. Traces of actual executions of NCSA Mosaic have been collected to reflect over half a million client requests for WWW documents in the Internet. There are three type of traces: condensed, window and structure. The condensed trace contains the sequence of object (URL) requests. Each line of a condensed trace corresponds to a single URL requested by the user; it contains the machine name, the time stamp when the request was made, the URL, the size of the documents and the object retrieval time in seconds. An example of a line from a condensed trace of a client is:
$cs20 \quad 785526142 \quad 920156 \quad ``http : //cs - www.bu.edu/lib/pica/bu - logo.gif'' \quad 1804 \quad 0.484092$

We utilize the condensed trace in our simulation because the client/web-server model in the web is very similar to the client/server model in mobile environments. To fit the web trace into mobile environments, we use the following strategies to transform it:

1. Clients in the web trace are mapped to mobile clients in the mobile environment.
2. URLs requested by users in the web trace are mapped to data items requested by mobile clients in the mobile environment.
3. The access time of a URL in the web trace is mapped to the query time in our simulation. Let the first requested URL's access time in the web trace be zero, other requested URLs' query time will be the seconds elapsed from the first request. For example, one month's real time is mapped to $60 * 60 * 24 * 30 = 2,592,000$ simulation seconds.

After the transformation, the trace can be used to simulate the client requests in mobile environments. Table 3 shows a sample trace selected from the trace transformed from the web trace using the above strategies. It contains 39 data requests. The access time is in simulation seconds and each data item ID represents a URL request.

3.1.2. Session Identification. Before applying the association rule algorithm, we first need to construct sessions out of the simulation trace. [10] uses a timeout to divide the page accesses of each user into individual sessions. If the time between page requests exceeds a certain limit, it is assumed that the user is starting a new session.

In this paper, we determine the session boundaries using an approach similar to [10], called *gap-based approach*. In this approach, we assume that a new session starts when the time delay between two consecutive requests is greater than a pre-specified time threshold-*gap*. In order to find an appropriate gap to identify the session boundaries, we have done some study on the web trace. We notice that if the time interval between two URL requests is larger than 1000 seconds, then the later requested data item has little relationship with the former one. Based on this observation, we use 1000-second as the gap to identify sessions in our simulation. Table 4 shows that the sample trace (in Table 3) can be divided into five sessions. Table 5 shows the frequent itemsets generated from the sample trace by applying our frequent itemsets generating algorithm to the five sessions shown in Table 4. Here, the minsup = 60%. From the table we can see that the largest frequent itemset is a 5-itemset. Table 6 shows the rules generated from the sample trace by applying our rule generating algorithm to the frequent itemsets shown in Table 5. Here, the minconf = 80%. As can be seen, the frequent 2- and 3-itemsets do not generate any rules. This is because after analyzing the frequent 5- and 4-itemsets, all the available rules have been generated. Thus, there is no need to generate any rules from the frequent 2- and 3-itemsets.

Table 4. The sample trace in sessions

Session #	Data Item Requests
1	0 1 2 422 1 2 423 10369 10370 10371
2	0 1 2 422 1 2 1384 1390 1385
3	0 1 2
4	0 1 2 422 1 2 1384 1385
5	0 1 2 422 1 2 1384 9800 8744

Table 5. The frequent itemsets

1-itemset	{0}, {1}, {2}, {422}, {1384}

2-itemset	{0, 1}, {0, 2}, {0, 422}, {0, 1384}, {1, 2}
	{1, 422}, {1, 1384}, {2, 422}, {2, 1384}
	{422, 1384}

3-itemset	{0, 1, 2}, {0, 1, 422}, {0, 1, 1384}
	{1, 2, 422}, {1, 2, 1384}, {2, 422, 1384}

4-itemset	{0, 1, 2, 422}, {0, 1, 2, 1384}
	{1, 2, 422, 1384}

5-itemset	{0, 1, 2, 422, 1384}

3.1.3. The Simulation Model. The architecture of our simulation model is shown in Figure 3. The simulation model is similar to the one used in [7, 15]. It consists of a single server that serves multiple clients.

The Server Model: To insure cache consistency, the server broadcasts invalidation reports (IRs) every L seconds. IR contains the update history of the past w broadcast intervals. To reduce the query latency, a number of replicated updated invalidation reports (UIRs) [7] are inserted into each IR interval. UIR contains the data items that have been updated after the last IR has been broadcasted. Other messages are served on a first-come-first-serve basis. If the

Table 6. The association rules

1-itemset	$\Longrightarrow 0(100\%)$ $\Longrightarrow 1(100\%)$ $\Longrightarrow 2(100\%)$ $\Longrightarrow 422(80\%)$
2-itemset	
3-itemset	
4-itemset	$0 \Longrightarrow \{1, 2, 422\}(80\%)$ $1 \Longrightarrow \{0, 2, 422\}(80\%)$ $2 \Longrightarrow \{0, 1, 422\}(80\%)$ $422 \Longrightarrow \{0, 1, 2\}(100\%)$
5-itemset	$1384 \Longrightarrow \{0, 1, 2, 422\}(100\%)$

Table 3. A sample trace

Access time	0	20	140	280	420	500	600	820	920	1000
Data Item	0	1	2	422	1	2	423	10369	10370	10371
Access time	2100	2200	2300	2400	2500	3000	3900	4000	4500	6000
Data Item	0	1	2	422	1	2	1384	1390	1385	0
Access time	6200	6400	7500	7600	7700	7800	7900	8000	8100	8200
Data Item	1	2	0	1	2	422	1	2	1384	1385
Access time	9300	9400	9500	9600	9700	9800	9900	10000	10800	
data item	0	1	2	422	1	2	1384	9800	8744	

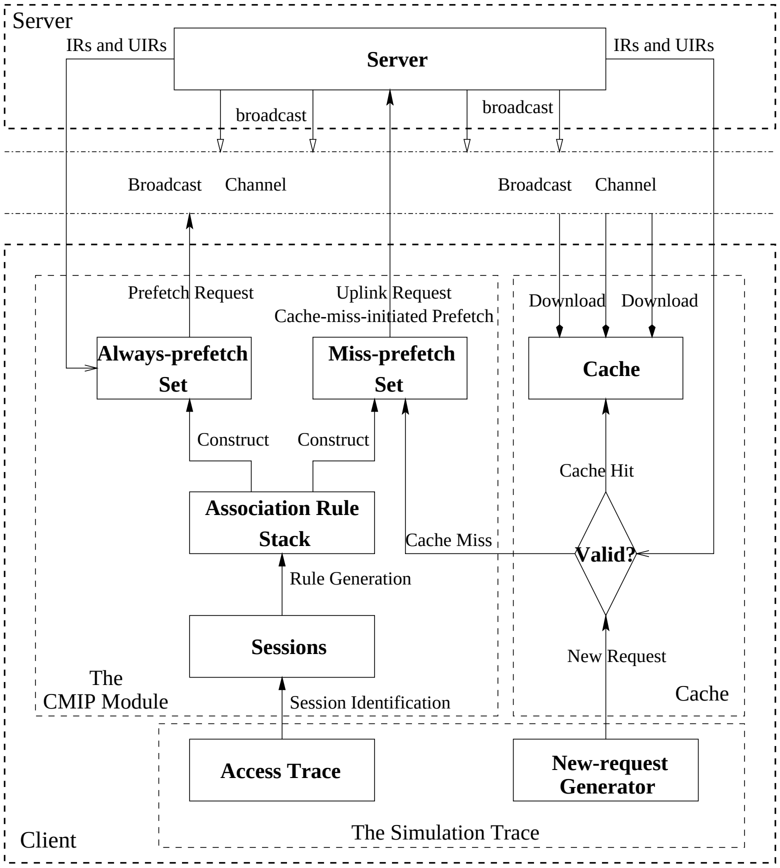


Figure 3. The system model

server is in the middle of a transmission when an IR or UIR has to be sent, the IR/UIR broadcast is deferred till the end of the current packet transmission.

The server generates a single stream of updates separated by an exponentially distributed update inter-arrival time. All updates are randomly distributed inside all the data items and only the server can update the data items. It is assumed that the bandwidth is fully utilized for broadcasting IRs and UIRs and serving client requests. The server processing time is considered to be negligible.

The Client Model: Every client, if active, listens to the IRs and UIRs to invalidate its cache accordingly. When a new request is generated, the client listens to the next IR or UIR to decide if there is a valid copy of the requested item in the cache. If there is one, the client answers the query immediately. Otherwise, a cache miss happens and the client sends an uplink request to the server. Before sending the uplink request, the requested item is sent to the CMIP module and a miss-prefetch set is constructed. After that, the uplink request is sent to the server, piggybacking the requests of the data in the miss-prefetch set. After receiving the uplink request, the server broadcasts the requested data on the broadcast channel. Then, the client can download them and answer the query.

The client model consists of two parts: the simulation trace and the CMIP module. The simulation trace consists of an access trace and a new-request generator. The access trace is used by the CMIP model to generate association rules and the new-request generator is used to simulate clients' requests. In the CMIP module, the access trace is divided into sessions. Then the sessions are mined using the association rule mining algorithms. The association rules with the minimum confidence (minconf) and the minimum support (minsup) are saved in the association rule stack. The always-prefetch set is constructed after obtaining the association rules. A miss-prefetch set for the cache-missed data item is constructed when a cache miss happens. To keep the association rules fresh, the client updates the access trace by feeding the recent query history from the new-request generator, and re-mines the rules periodically. The always-prefetch set will be updated consequently and the new rules are used to construct miss-prefetch sets for oncoming cache misses. Most of the system parameters are listed in Table 7.

Table 7. Simulation parameters

Number of clients	200
Number of data items	13,833 items
Data item size	1024 bytes
Broadcast interval (L)	20 seconds
Broadcast bandwidth	10kb/s
Cache size	10 to 300 items
Broadcast window (w)	10 intervals
UIR replicate times $(m-1)$	4 (5 - 1)
Rule Minimum Support ($minsup$)	60 (%)
Rule Minimum Confidence ($minconf$)	80 (%)

3.2. Simulation Results

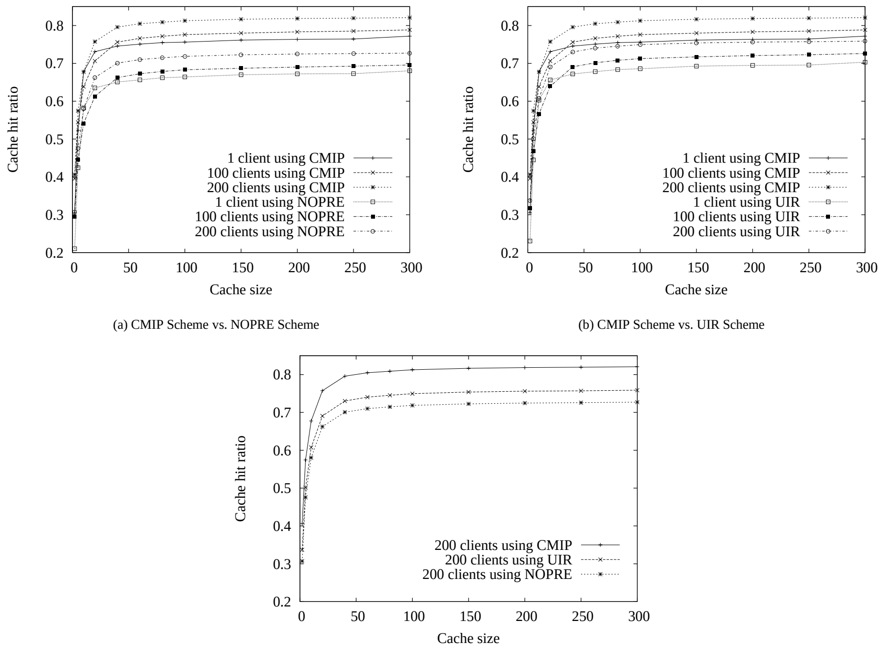
We compare the performance of our CMIP scheme with two other schemes: the UIR scheme [7] and the UIR scheme without prefetching (called NOPRE scheme). Three metrics are used to evaluate their performance: the percentage of additional traffic, the cache hit ratio, and the percentage of reduced uplink requests.

3.2.1. The Cache Hit Ratio. The performance metrics such as access latency and the uplink cost have strong relations with the cache hit ratio. For example, if the cache hit ratio is high, the access latency can be reduced since many of the requests can be served within the cache without sending uplink requests to the server. In our simulation, we evaluate the cache hit ratio of the three schemes using various cache sizes and different number of mobile clients. Figure 4 compares the cache hit ratio of the CMIP scheme to the UIR scheme and the NOPRE scheme. As shown in Figure 4(a), our CMIP scheme can greatly improve the cache hit ratio, compared to the NOPRE scheme. Since the NOPRE scheme does not prefetch data items, less data item requests can be served in the cache. When the cache is full, it uses LRU scheme to do cache replacement. However, LRU does not consider the relationships among data items and important data items may be replaced by others using LRU scheme. For the CMIP scheme, it has mined the relationships among data items and known which data items have higher future access probabilities. So it can keep important data items longer in the cache. Thus, more client's requests can be served locally in the cache and the cache hit ratio is improved. This explains why our CMIP scheme can achieve a better performance than the NOPRE scheme in term of cache hit ratio.

Figure 4(b) shows that the CMIP scheme is also better than the UIR scheme. This can be explains as follows. The UIR scheme is based on the cache locality: a client has a large chance to access the invalidated cache items in the near future; downloading these data items in advance should be able to increase the cache hit ratio [7]. However, the UIR scheme does not differentiate the items in the cache. Although, the UIR scheme classifies data items into hot data and cold data and treat them differently during cache management, they are treated equally in prefetching and assumed to have the same access probability in the future. However, as stated above, some of the data items within the cache are important, while others are not. The CMIP scheme differentiates the importance of the data items and the important data items are kept longer in the cache. As a result, the CMIP scheme can achieve a higher cache hit ratio than the UIR scheme.

Figure 4(c) compares three schemes when the number of mobile clients is 200. We put it here to show a clear picture of the comparison of these three schemes. As can be seen, CMIP is about 15% better than the NOPRE scheme and about 9% better than UIR. Although UIR scheme achieves a better cache hit ratio than NOPRE scheme, later we will see that the high cache hit ratio of the UIR scheme is at the cost of high additional traffic.

3.2.2. The Percentage of Reduced Uplink Requests. The percentage of reduced uplink requests is defined as the ratio of the number of saved uplink requests to the total number of requests. Since the NOPRE scheme does not prefetch, it will not be compared. Figure 5 shows the percentage of reduced uplink requests using the CMIP



(a) CMIP Scheme vs. NOPRE Scheme

(b) CMIP Scheme vs. UIR Scheme

(c) CMIP Scheme vs. UIR Scheme vs. NOPRE Scheme when clients =200

Figure 4. The comparison of cache hit ratio using three schemes

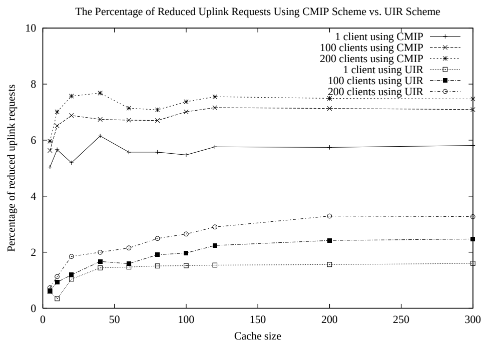


Figure 5. The percentage of reduced uplink requests

scheme and the UIR scheme. Generally speaking, using our CMIP scheme, the percentage of reduced uplink requests increases as the cache size increases. After the cache size reaches 120, the percentage no longer changes. This can be explained as follows. As the cache size increases, more and more important data items having high access probability can be stored in the cache. When the cache size is still not large enough, the cache hit ratio will increase sharply when cache size increases. As a result, the rate of reduced uplink requests is higher than the arrival rate of new requests. Thus, the percentage of reduced uplink requests has a trend of increasing. When the cache size is big enough to hold all the items within our prefetch sets, the rate of reduced uplink requests is no long significant to the arrival rate of new requests. Hence, the percentage of reduced uplink requests no longer changes after reaching a certain cache size.

From Figure 5, we also notice that although the trend of the percentage of reduced uplink requests is increasing, there are some ups and downs. For example, when the number of clients is 200, the percentage of reduced uplink requests reaches the peak when the cache size is about 50. As the cache size continues increasing, the percentage begins to decrease a little bit, and then it increases again. This can be explained as follows. As the cache size increases, more queries can be served within the cache. So, the number of reduced uplink requests keeps increasing. Depending on the access patterns of the clients, the number of reduced uplink requests may increase at a rate higher or lower than the increase of the number of requests. If the rate is higher, the percentage of reduced uplink requests will increase. If the rate is lower, the percentage will decrease instead. Thus, there are some ups and downs in the percentage of reduced uplink requests, although the trend is increasing.

Figure 5 also shows that our CMIP scheme outperforms the UIR scheme with various cache sizes and various number of mobile clients. The percentage of reduced uplinks using the CMIP scheme is twice as much as that of using the UIR scheme. In term of percentage of reduced uplink request, even the worst case of the CMIP scheme is better than the best case of the UIR scheme. For instance, the CMIP scheme has the worse performance when there is one mobile client and the percentage of reduced uplinks is about 6% (7.5% at best). But for the UIR scheme, it achieves its best performance when there are 200 mobile clients with about 3% of reduced uplinks. This is because the CMIP scheme can better predict the future access of the clients and prefetch them in advance than the UIR scheme. Hence, the CMIP scheme can reduce the uplink requests at a percentage much higher than the UIR scheme.

3.2.3. The Percentage of Additional Traffic. The percentage of additional traffic is defined as the ratio of the number of prefetches from the broadcast channel to the number of requests. As we know, downloading a data item from the channel also consumes a lot of system resources, such as the bandwidth and the power. So the prefetch scheme should not do aggressive prefetching; otherwise, it will consume too much system resources. This is especially important in mobile environments, where the system resource is very limited.

One argument about this metric is that the percentage of additional traffic depends on the number of requests that are quite application-dependent and it does not make much sense to compare on this metric. We argue that this metric, when combined with other metric, such as the cache hit ratio, can well describe a prefetch scheme's efficiency and predictability. A good prefetch scheme should be able to improve the cache hit ratio without incurring too much additional traffic (or prefetches) to the system. Figure 6 compares the two schemes in term of the percentage of additional traffic.

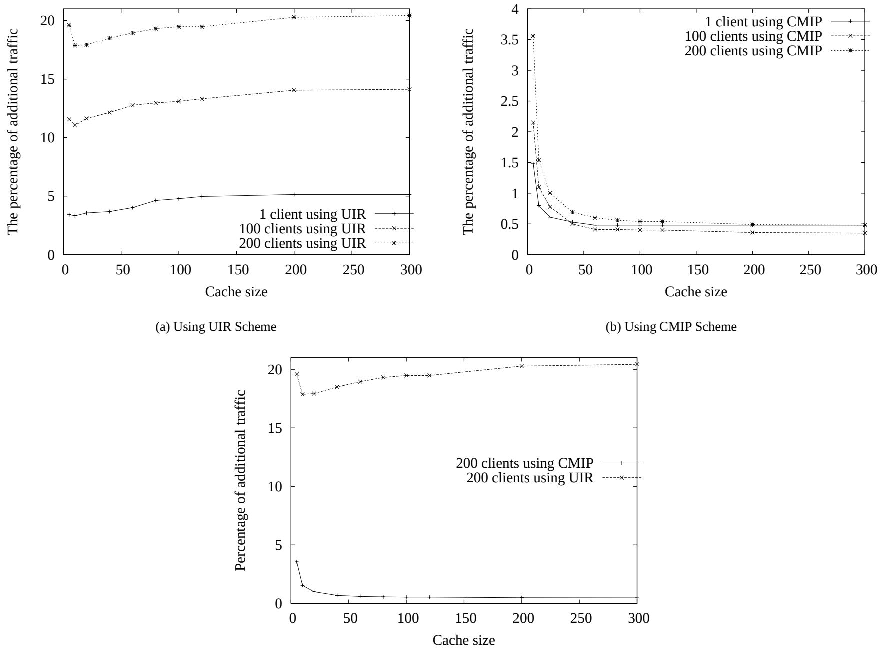
Figure 6(a) shows that the UIR scheme incurs much more additional traffic to the system. For example, the percentage of additional traffic for UIR scheme is up to 20% when there are 200 clients. This is because the UIR scheme is an aggressive prefetch scheme. Whenever a data item within the cache has been updated and is broadcasted on the channel, the client will download it and update the cache. But for the CMIP scheme, the percentage of additional traffic to the system is negligible, as shown in Figure 6(b). For example, the percentage is lower than 0.5% when cache size becomes larger than 100. Why the percentage of additional traffic is so small is due to the characteristic of our CMIP scheme. Using the CMIP scheme, only those data items which are within our prefetch sets are prefetched. The data items within prefetch sets are got from association rules with a high confidence and support. So the set of data items to be prefetch is small and the number of prefetches is also small. This explains why the percentage of additional traffic is negligible.

Figure 6(c) compares the two schemes in term of the percentage of additional traffic when there are 200 mobile clients. From 6(c), we can see that our CMIP scheme incurs only a fraction of 20 of the percentage of additional traffic incurred by the UIR scheme. By far, we can say that our CMIP scheme is much better than the UIR scheme and the NOPRE scheme in terms of increased cache hit ratio, reduced uplink requests and negligible additional traffic.

4. Related Work

In the literature, prefetch technique is widely employed to reduce the access latency in WWW environments [17, 16, 8, 12, 9]. [17] presents a predictive prefetching scheme for the World Wide Web in which the server tells the clients which files are likely to be requested by the user, and the clients decide whether to prefetch these files or not based on local considerations (such as the contents of the local cache). In [16], an adaptive network prefetch scheme is proposed. This scheme predicts the files' future access probabilities based on the access history and the network condition. The scheme allows the prefetching of a file only if the access probability of the file is greater than a function of the system bandwidth, delay and retrieval time. In [9], Cohen and Kaplan investigate three other types of prefetching in web: pre-resolving host-names (pre-performing DNS lookup); preconnecting (prefetching TCP connections prior to issuance of HTTP request); and pre-warming (sending a "dummy" HTTP HEAD request to Web servers). [12] develops a new method for prefetching Web pages into the client cache. Clients send reference information to the Web server, which aggregates the reference information in near-real-time and then disperses the aggregated information to all clients, piggybacked on GET responses. The information indicates how often hyperlink URLs embedded in pages have been previously accessed relative to the embedding page. Based on the knowledge about which hyperlinks are generally popular, clients initiate prefetching of the hyperlinks and their embedded images according to any algorithm they prefer.

Most of these work were not designed for mobile environments and did not consider the constraints of mobile environments. Recently, several prefetch schemes have been proposed as a client-side technique to reduce the access latency in mobile environments [1, 13, 6, 19]. In [1], a simple prefetching heuristic, called PT, computes the value of a page by taking the product of the probability (P) of accessing of the page with the time (T) that will elapse before that page appears on the broadcast again. PT finds the page in the cache with the lowest pt value and replaces it with the current broadcast page if the latter has a higher pt value. However, this time-based prefetch scheme is expensive to implement since it computes the pt for each item in



(a) Using UIR Scheme

(b) Using CMIP Scheme

(c) CMIP Scheme vs. UIR Scheme when clients = 200

Figure 6. The percentage of additional traffic

the cache at every clock tick. A similar scheme has been proposed in [13], which uses fv, a function of the access rate of the data item only, to evaluate the value of each data item *i* that becomes available to the client on the channel. If there exists a data item *j* in the client's cache such that $fv(i) > fv(j)$, data item j is removed from the cache and replaced with i.

A prefetch scheme based on the cache locality, called UIR scheme, was proposed in [7]. It assumes that a client has a large chance to access the invalidated cache items in the near future. It proposes to prefetch these data items if it is possible to increase the cache hit ratio. In [6], Cao improves the UIR scheme by reducing some unnecessary prefetches based on the prefetch access ratio (*PAR*). In this scheme, the client records how many times a cached data item has been accessed and prefetched, respectively. It then calculates the PAR, which is the number of prefetches divided by the number of accesses, for each data item. If the PAR is less than one, it means that the data item has been accessed a number of times and hence the prefetching is useful. The clients can mark data items as non-prefetching when $PAR > b$, where b is a system tuning factor. The scheme proposes to change the value of b dynamically according to power consumption. This can make the prefetch scheme adaptable, but no clear methodology as to how and when b should be changed. Yin et al. [19] proposed a power-aware prefetch scheme, called value-based adaptive prefetch (*VAP*) scheme, which can dynamically adjust the number of prefetches based on the current energy level to prolong the system running time. The VAP scheme defines a value function which can optimize the prefetch cost to achieve better performance.

These existing schemes have ignored the following characteristics of a mobile environment: (1) a mobile client may query some data items frequently, (2) data items queried during a period of time are related to each other, (3) cache miss is not a isolated events; a cache miss is often followed by a series of cache misses, (4) piggybacking several requests in one uplink request consumes little additional bandwidth but reduces the number of future uplink requests. In this paper, we addressed these issues using a cache-miss-initiated prefetch scheme, which is based on association rule mining technique. Association rule mining is a widely used technique in finding the relationships among data items. The problem of finding association rules among items is clearly defined by Agrawal et al. in [5]. However, in the mobile environment, one cannot apply the existing association rule mining algorithm [4] directly because it is too complex and expensive to use.

This makes our algorithm different from that of [4] in twofold. First, we are interested in rules with only one data item in the antecedent and several data items in the consequent. Our motivation is to prefetch several data items which are highly related to the cache-miss data item within the cache-miss initiated uplink request. We want to generate rules where the antecedent is one data item, but the cache-missed data item and the consequent is a series of data items, which are highly related to the antecedent. If we have such rules, we can easily find the data items which should also be piggybacked in the uplink request. Second, in mobile environment, the client's computation and power resources are limited. Thus, the rule-mining process should not be too complex and resource expensive. It should not take a long time to mine the rules. It should not have high computation overhead. However, most of the association rule mining algorithms [4, 5] have high computation requirements to generate such rules.

5. Conclusions

Client-side prefetching technique can be used to improve system performance in mobile environments. However, prefetching also consumes a large amount of system resources such as computation power and energy. Thus, it is very important to only prefetch the right data. In this paper, we proposed a cache-miss-initiated prefetch (*CMIP*) scheme to help the mobile clients prefetch the right data. The CMIP scheme relies on two prefetch sets: the always-prefetch set and the miss-prefetch set. Novel association rule based algorithms were proposed to construct these prefetch sets. When a cache miss happens, instead of sending an uplink request to only ask for the cache-missed data item, the client requests several items, which are within the miss-prefetch set, to reduce future cache misses. Detailed experimental results verified that the CMIP scheme can greatly improve the system performance in terms of increased cache hit ratio, reduced uplink requests and negligible additional traffic.

References

[1] S. Acharya, M. Franklin, and S. Zdonik. Prefetching From a Broadcast Disk. *Proc. Int'l Conf. on Data Eng.*, pages 276–285, Feb. 1996.

[2] S. Acharya, M. Franklin, and S. Zdonik. Balancing Push and Pull for Data Broadcast. *Proc. ACM SIGMOD*, pages 183–194, May 1997.

[3] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data Management for Asymmetric Communication Environments. *Proc. ACM SIGMOD*, pages 199–210, May 1995.

[4] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[5] R. Agrawal, Tomasz Imielinski, and Arun Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.

[6] G. Cao. Proactive Power-Aware Cache Management for Mobile Computing Systems. *IEEE Transactions on Computers*, 51(6):608–621, June 2002.

[7] G. Cao. A Scalable Low-Latency Cache Invalidation Strategy for Mobile Environments. *IEEE Transactions on Knowledge and Data Engineering*, 15(5), September/October 2003 (A preliminary version appeared in ACM MobiCom'00).

[8] K. Chinen and S. Yamaguchi. An Interactive Prefetching Proxy Server for Improvement of WWW Latency. In *Proc. INET 97*, June 1997.

[9] E. Cohen and H. Kaplan. Prefetching the means for document transfer: A new approach for reducing web latency. In *Proceedings of IEEE INFOCOM*, pages 854–863, 2000.

[10] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, 1999.

[11] C. R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW Client Based Traces. Technical Report TR-95-010, Boston University, CS Dept, Boston, MA 02215, July 1995.

[12] D. Duchamp. Prefetching hyperlinks. In *USENIX Symposium on Internet Technologies and Systems (USITS'99)*, 1999.

[13] V. Grassi. Prefetching Policies for Energy Saving and Latency Reduction in a Wireless Broadcast Data Delivery System. In *ACM MSWIM 2000*, Boston MA, 2000.

[14] S. Hameed and N. Vaidya. Efficient Algorithms for Scheduling Data Broadcast. *ACM/Baltzer Wireless Networks (WINET)*, pages 183–193, May 1999.

[15] Q. Hu and D. Lee. Cache Algorithms based on Adaptive Invalidation Reports for Mobile Environments. *Cluster Computing*, pages 39–48, Feb. 1998.

[16] Z. Jiang and L. Kleinrock. An Adaptive Network Prefetch Scheme. *IEEE Journal on Selected Areas in Communications*, 16(3):1–11, April 1998.

[17] V. Padmanabhan and J. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. *Computer Communication Review*, pages 22–36, July 1996.

[18] N. Vaidya and S. Hameed. Scheduling Data Broadcast in Asymmetric Communication Environments. *ACM/Baltzer Wireless Networks (WINET)*, pages 171–182, May 1999.

[19] L. Yin, G. Cao, C. Das, and A. Ashraf. Power-Aware Prefetch in Mobile Environments. *IEEE International Conference on Distributed Computing Systems (ICDCS)*, July 2002.