ASSOCIATION RULES FOR SUPPORTING HOARDING IN MOBILE COMPUTING ENVIRONMENTS

Yücel Saygın*, [1] Özgür Ulusoy* , and Ahmed K. Elmagarmid†

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

†Department of Computer Sciences , Purdue University, USA

Abstract

Mobile computers equipped with wireless communication devices are becoming more and more popular each day. One of the features that a mobile computer should provide is disconnected operation which is performed by hoarding. The process of hoarding can be described as loading the data items needed in the future to the client cache prior to disconnection. However, letting the user specify which files he/she may need in the future may be too much of a burden. Therefore, the automating process must be performed seamlessly. Automated hoarding is the process of predicting the hoard set without any user intervention. In this paper, we describe an application independent and generic technique for determining what should be hoarded prior to disconnection. Our method utilizes association rules that are extracted by data mining techniques for determining the set of items that should be hoarded to a mobile computer prior to disconnection. The proposed method was implemented and tested on synthetic data to estimate its effectiveness. Performance experiments determined that the proposed rule-based methods are effective in improving the system performance in terms of the cache hit ratio of mobile clients especially for small cache sizes.

Key words: Mobile databases, disconnected operation, hoarding, association rules.

1 Introduction

Recent advances in computer hardware technology have made it possible the production of small computers, such as notebooks and palmtops, which can be carried around by users. These portable computers are often 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 efficiently access a large number of shared databases on stationary/mobile data servers [AK93, Chr93, DH95]. A detailed description of the issues related to the recent research on client server computing in mobile environments can be found in [JHE99].

Mobile devices are beginning to support applications such as multimedia and World Wide Web. Users would like to surf the Internet and read their email while traveling. However, mobile computers with wireless communication often frequently disconnect from the network due to the cost of wireless communication or the unavailability of the wireless network. Consider a businessperson who frequently travels around by plane. Before getting on the plane he/she would have to disconnect and then continue his/her work on the air without wireless network support. The same businessman may travel by car and knows that after a certain point, no wireless network support in a certain region exists. Thus, his or her operation must go on into disconnected mode. Another

[1] Corresponding author Yücel Saygın is currently visiting Purdue, and can be contacted via `saygin@cs.purdue.edu`.

person may not be willing to pay for a continuous wireless connection, but would prefer to connect intermittently, in order to access files of interest. Such scenarios suggest that the disconnection mode may be in high demand and the system should provide support for disconnected operation. The process of hoarding involves loading the files which may be needed in the future to the client cache prior to disconnection. The early work on disconnected operation, done by the CODA project group in CMU [KS92], was based on user's manual specification of the hoard set before disconnection via a script language. They also had some limited form of automated hoarding.

Automated hoarding is the process of predicting the hoard set without any user intervention. An approach for automated hoarding based on clustering was proposed by Kunning and Popek [KP97]. Their clustering method was based on the notion of semantic distance. The semantic distance between two operations was calculated by using the number of intervening references to other objects. Another approach based on identifying the working sets of different programs was proposed by Tait et al. [TLAC95]. In that work, the authors constructed program trees together with the files used by those programs. For a program, each execution resulted in a tree. The trees belonging to the same program were merged to obtain a final hoard set for that program.

In previously proposed hoarding methods, it is assumed that users run a program while disconnected and would like to have access to other programs and data files needed by the original program in the cache prior to disconnection. This assumption enables researchers to use semantic information about the data and executable files such as the file extensions (.txt, .exe), and the directories these files are located. The difference between our approaches and the previously proposed approaches is that we do not assume any knowledge about file extensions or directory information. We propose a more generic and application independent approach to hoarding. In our approach, we assume that no information is available regarding which programs are running when the client requests are issued and what operations are issued on the data. We introduce the notion of a session which consists of client requests and is independent of the other sessions. Our approach to automated hoarding is inferencing with association rules that simply exploits the access patterns of clients to guess the future requests. Association rules provide us with sound priority measures like the support, confidence, and size of the rules which can be utilized for limiting the set of items to be loaded to the client cache.

We performed experiments and compared our association rule based hoarding method with LRU (Least Recently Used) caching. The results showed that rule based hoarding had a higher cache hit ratio especially for small cache sizes which is typical for thin clients and palm top computers.

Hoarding through association rules can be achieved through the following three phases of execution: 1) mining of the history, 2) construction of the candidate set, and 3) construction of the hoard set. In phase 1, the history of client requests are preprocessed and analyzed to come up with the association rules using data mining techniques. In phases 2 and 3, these rules are used to construct the candidate set and the hoard set, respectively.

The rest of the paper is organized as follows: Section 2 and Section 3 describe the first and second phases of the hoarding process, respectively. Section 4 describes the simulation model and provides the experimental results we obtained, and finally Section 5 concludes the paper while discussing possibilities for future research.

2 Mining Histories for Association Rules

The history of previous client requests should be mined to come up with association rules. Mining algorithms for association rules assume that the input data is a collection of sets of items. In what follows, we provide a formal definition of association rules and discuss how the client request history is preprocessed so that state of the art, efficient, and incremental association rule finding algorithms are utilized.

2.1 Association Rules: An Overview

The problem of finding association rules among items has been clearly defined by Agrawal et al. [AS94]. This definition is provided for clarity:

Let $I = i_1, i_2, ..., i_m$ be a set of literals, called items and D be a set of transactions[2], 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 the association rules that have a support and a confidence greater than a user-specified thresholds [AS94]. The thresholds for confidence and support are called *minconf* and *minsup*, respectively. The problem of mining association rules has been well studied and is outside the scope of this paper. In this paper, we focus on how a request history can be analyzed to construct association rules that capture the affinity among mobile client requests. We will examine the utilization of the resulting association rules in hoarding. In this approach, we assume that past data access requests of clients issued to the server are stored in a history log. Association rules are obtained by analyzing the history of requests issued by clients.

To date, association rules have been used in various contexts. For example, they are useful in determining the placement and pricing of merchandise at megamarkets [AS94], World Wide Web [MJHS96], and broadcasting data [SU99]. However, to the best of our knowledge, the automated use of association rules in hoarding has not yet been considered by any researcher.

2.2 The Use of Association Rules for Hoarding

Data mining techniques can be involved in hoarding algorithms in order to extract association rules from client request histories. The extracted association rules which represent client access patterns, can be used to predict future client requests. The predicted request set is what should be loaded prior to disconnection so that the future client requests are satisfied locally without requiring a connection to the server. Association rules provide guides such as support, confidence and size of the rules which are crucial in limiting the size of the data set to be hoarded.

In order to describe our approaches for hoarding, we first need to introduce the notion of a session. A *session* consists of a group of continuous client requests and represents a period of user interest for a particular issue. In theory, such sessions are independent of each other. We assume that user requests consist of sessions and that client requests could be partitioned into sessions. Each session contains some patterns of client requests. Data mining techniques find these patterns and produce rules that can be used to build a rule base of associations. When disconnection occurs, rules are used to infer a user's future requests. The rest of the session is stored locally at the client's side. It is assumed here that hoarding sets are limited to one session. Therefore, histories must be divided into a sequence of sessions.

[2]In the context of data mining, a transaction is a set of items purchased at a time in market basket data and it should not be confused with the notion of transaction in the database context. Transactions will be mapped to sessions for hoarding.

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

Table 1: User-based partitioning of the client request history.

2.3 Partitioning the History into Sessions

There can be two basic approaches for mining the client request history depending on how the history of user requests is partitioned: 1) the flat approach and 2) the 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 client request history into subsets with respect to the user who requested them, as shown in Table 1. Table 1 presents the windows of requests corresponding to each user, such as $user_1$ who had k different request windows. The analysis is done for each subset of the client request history corresponding to a user independent of the other subsets.

In order to make use of the existing data mining algorithms, we need to construct sessions out of the existing history. When we cannot assume any session boundaries, we need to use a sliding window approach for simulating the sessions. We used a gap-based approach to determine the session boundaries. In gap-based approach, we assume that a new session starts when the time delay between two consecutive requests is greater than a prespecified threshold called *gap*.

In both flat and user-based partitioning approaches, we divide the request set into windows and find the association rules using the data mining algorithms mentioned previously. These windows correspond to the sessions for the data mining algorithm. However, the construction of the windows is different for the two approaches. In the user-based partitioning approach, we can observe the temporal patterns of a user to divide his/her request set into sessions. In the gap-based method, windows are clusters of item/s requested according to the times of requests; if a user has requested items consecutively and relatively quickly, 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 an extended 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 three requests, namely 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 in order to put all items requested by the same user into the same window; this might be a reasonable approach if there are multiple users and each user accesses a reasonable amount of items. When the number of users is small and each user accesses many 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 item sets.

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 users is lost in this approach. In user-based partitioning, the rules obtained are considered to be more reliable, as they are based on finding the association rules for the same user by partitioning the client request history

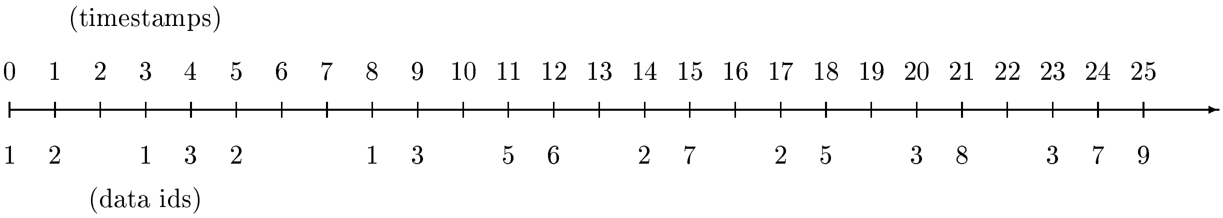

Figure 1: A sample history

into users.

In order to be able to use data mining algorithms to obtain association rules, windows are mapped to *sessions* and data item requests are mapped to *items*. Standard association rule finding algorithms are then used to find the association rules in the history. The state of the art association rule finding algorithm called *Apriori* is appropriate for our purposes [AS94]. Initially, the Apriori algorithm finds all the large data items and computes their supports. Large data items are characterized by the fact that the percentage of the windows that supports them is greater than the minimum support value. Then using these large items, 2-itemsets (i.e., itemsets of size 2) are found by combining large items by checking their support values against the minimum support. Incrementally, item sets of size n are constructed by using item sets of size $n-1$ (where $n > 1$) until no larger item sets could be found. We extended the Apriori algorithm with an inverted list data structure. The inverted list structure maintains 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 $Large\text{-}item_2$.

Figure 1 shows a sample history generated for clarifying the proposed methods. The numbers above the line are the time stamps and the numbers below the line are requested items. If we do not assume any session boundaries, then we need to use a sliding window algorithm which results in the following sessions if the window size is set to 3: $\{1,2,1\}$, $\{2,1,3\}$, $\{1,3,2\}$, $\{3,2,1\}$, $\{2,1,3\}$, $\{1,3,5\}$, $\{3,5,6\}$, $\{5,6,2\}$, $\{6,2,7\}$,$\{2,7,2\}$, $\{7,2,5\}$, $\{2,5,3\}$, $\{5,3,8\}$, $\{3,8,3\}$, $\{8,3,7\}$, $\{3,7,9\}$. If the gap-based methods will be used and the gap is set to 2 units, then we have 8 sessions which are $\{1,2\}$,$\{1,3,2\}$, $\{1,3\}$, $\{5,6\}$, $\{2,7\}$, $\{2,5\}$, $\{3,8\}$, $\{3,7,9\}$. If a length-based approach is going to be used and length is set to 3, then we have; $\{1,2,1\}$, $\{3,2,1\}$, $\{3,5,6\}$, $\{2,7,2\}$, $\{5,3,8\}$, $\{3,7,9\}$. Cluster-based approach depends on the clustering methods we use and will not be discussed in this paper. Cluster-based methods generate session patterns similar to the sessions generated by the gap-based approaches.

3 Utilization of the Association Rules for Hoarding

The association rules obtained after mining the request history are used for determining the *candidate set* and the *hoard set* of the client upon disconnection. The candidate set is defined as the set of all candidates for hoarding for a specific client. Hoard set is the set of data items actually loaded to the client prior to disconnection. A candidate set is constructed using inferencing on association rules as explained in Section 3.1. Some other heuristics are used to prune the candidate set to the hoard set so that it fits to the cache of the mobile client as explained in Section 3.2.

The process of automated hoarding via association rules can be summarized as follows:

- Requests of the client in the current session are used through an inferencing mechanism to construct the candidate set prior to disconnection,
- Candidate set is pruned to form the hoard set,

- Hoard set is loaded to the client.

The need to have separate steps for constructing the candidate set and the hoard set arises from the fact that users also move from one machine to another that may have lower resources. The construction of the hoard set must adapt to such potential changes.

Details about how the hoarding process constructs candidate and hoard sets are provided in Sections 3.1 and 3.2 below.

3.1 Construction of the Candidate Set

An inferencing mechanism is used to construct the candidate set of data items that are of interest to the client to be disconnected. The candidate set of the client is constructed in two steps:

1. The inferencing mechanism finds the association rules whose heads (i.e., left hand side) match with the client's requests in the current session,

2. The tails (i.e., right hand side) of the matching rules are collected into the candidate set.

The inferencing mechanism examines the current requests and predicts future ones. The rules obtained from the history shown in Figure 1 as a result of gap-based partitioning approach are: $1 \rightarrow 2$ and $1 \rightarrow 3$ with a minimum support of 20% and minimum confidence of 80%. If we have the list $6, 7, 1, 5$ belonging to a current client session, then the data items 2 and 3 should be placed into the candidate set as a result of the inference mechanism because data item 1 was requested and our rules tell us with a certain confidence level that if a user requests data item 1, then he/she will also request data items 2 and 3 in the near future.

In order to have an effective hoarding process, the set of rules should be large. This way, we could come up with a sufficient hoard set that will satisfy the client's needs for most of the client request patterns. To obtain a large rule set after mining the history, we need to have low minimum support and confidence values for the association rules. Priorities need to be assigned for the items obtained as a result of the inferencing. Our priority metric is based on the rule confidence and support values; i.e., items inferred by a rule with a high confidence or support. To include both the support and confidence value, the priority of a rule is set to be the multiplication of the confidence and support value for that rule.

3.2 Construction of the Hoard Set

The client that issued the hoard request has limited resources. The storage resource is of particular importance for hoarding since we have a limited space to load the candidate set. Therefore, the candidate set obtained in the first phase of the hoarding set should shrink to the hoard set so that it fits the client cache. Each data item in the candidate set is associated with a priority. These priorities together with various heuristics must be incorporated for determining the hoard set. The data items are used to sort the rules in descending order of priorities. The hoard set is constructed out of the data items with the highest priority in the candidate set just enough to fill the cache.

For an effective hoarding, the cache misses during disconnection should be recorded and reflected to the history upon reconnection. In this manner, we can capture the whole picture of client requests, both connected and disconnected.

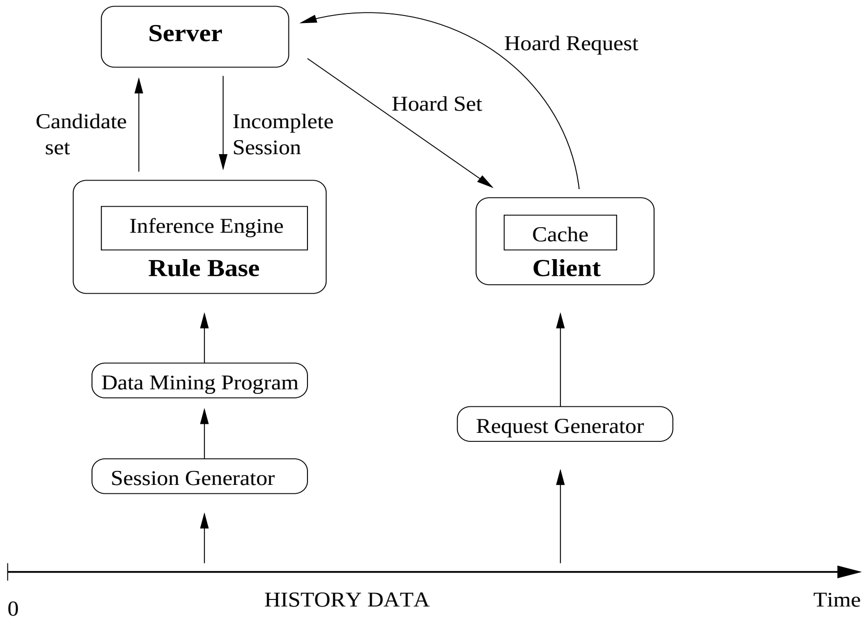

Figure 2: Architecture of the Simulation

4 Simulation Model and Experimental Results

We implemented the data mining algorithms and the rule based hoarding mechanism to show the effectiveness of the proposed methods. The data set generator of the IBM Quest project was used for simulating the history of sessions [AAB+96]. The simulation model we used and the experimental results we obtained are provided in Section 4.1 and Section 4.2, respectively.

4.1 Simulation Model

In our simulation model, we make use of the history of sessions for both extracting rules and testing the effectiveness of the rule based hoarding method. The architecture of our simulation model is shown in Figure 2.

We assume that the whole set of past requests are stored in a history. Our experiments consist of two stages:

1. *Rule extraction*, where the the history is mined for extracting association rules.

2. *Rule performance evaluation*, where the history is used to test the effectiveness of the extracted rules.

First, the sessions are generated, then the rule extraction mechanism performs the task of extracting the association rules from the collection of sessions. 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 simulation program. The simulation program was written in C++.

The second stage of our experiments is performed as follows: First of all, the client requests are divided into two groups:

1. Requests issued while disconnected,

2. Requests issued while connected.

A request generator program takes the data generated by the IBM benchmark and converts it to a stream of client requests consisting of sessions. The beginning and end of the sessions are marked and disconnect requests are inserted. Hoard requests are also inserted as a client request that marks disconnection times. Hoard requests are issued relative to the beginning of the sessions and have a normal distribution with mean $average\_session\_length/2$.

The main subsystems of the broadcast simulation program are the *server, client, cache*, and *rulebase*. Server has a rule base used for constructing the hoard set. The client interacts with the server by sending both data and hoard requests to it. Rule base loads the association rules from a file generated by the data mining program. The client has a cache to store a limited amount of requested items.

4.2 Experimental Results

We used the IBM Almaden Quest data generator to simulate the history of client requests [AAB+96]. The data generated by this benchmark is suitable for our purposes because we could finetune the generated data. Basically, the benchmark produces a collection of sets of items. The sets of items correspond to the sessions in the context of hoarding.

We implemented the Apriori algorithm for mining association rules. We extended the mining algorithm with an inverted list so that the implementation could be performed with set operations like union and intersection. Data mining is performed in main memory. The running time of the data mining algorithm does not exceed a few seconds for a thousand sessions. Considering that the mining process is not done frequently, these running times are not significant.

The client request log is partitioned into disconnected and connected periods. The length of these periods is calculated in terms of number of requests. The client part of the system loads the requests into a queue together with the disconnection period and frequency information. When the disconnection period is reached, the hoard set is sent to the client, and the cache hit ratio is measured for the disconnection period. Overall, the disconnected cache hit ratio is calculated by averaging the individual disconnected cache hit ratios. The resulting hit ratios can be compared to that of the LRU approaches.

We have performed various experiments to see the effect of different parameters on the performance. Local storage done by the mobile client is referred to as its cache. Our main performance criterion is the cache hit ratio since the main purpose of hoarding process is to decrease the number of cache misses during disconnection.

The number of sessions does not have any impact on the performance; the value selected for this parameter in our experiments is 200. The number of different data items generated is 150. We assume 50 different patterns in the data set and the average pattern length is taken to be 10 items. Cache size and its impact on the performance is an important parameter that we need to evaluate. We measure the cache size in terms of the number of data items that the cache can hold since we did not simulate the size of data items. Considering that the data items are most likely html documents or multimedia applications, data items would consist of one or more pages. Therefore, the locality in the number of pages would not affect the system performance. As a result, the cache size in terms of the number of data items does not deviate very much from real life where cache size is measured in terms of kilo bytes.

We targeted small size devices like palmtops, which have a limited memory and potentially short sessions of connection. A cache size of 50 or at most 100 items (or objects) used in our experiments seems reasonable, considering the amount of space multimedia or web items consume. Since we are

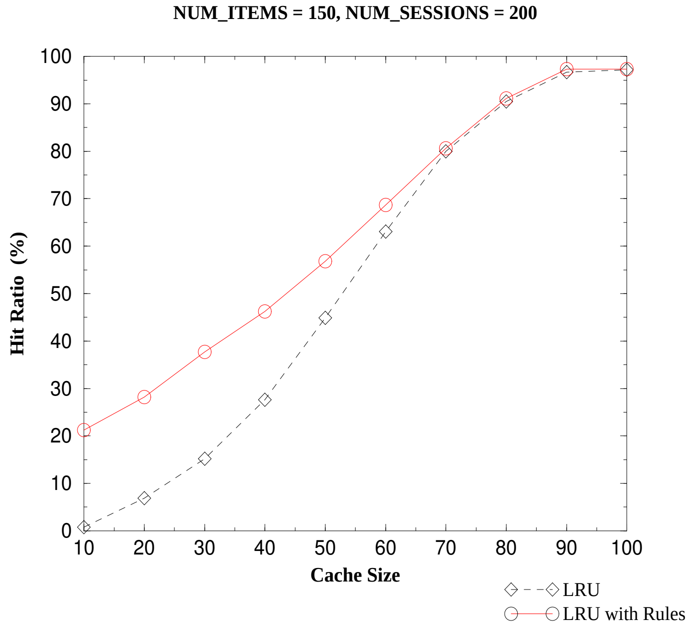

Figure 3: Cache Hit ratio as a function of the cache size

assuming thin clients with a small cache size, we varied the cache size from 10 to 100 items. The results we obtained for the cache hit ratio under different cache sizes are displayed in Figure 3. As we observe from the figure, in general, rule-based hoarding methods are more effective for small cache sizes. As the cache size gets closer to the number of distinct data items, the cache hit ratio gets closer to the LRU cache hit ratio. Since the mobile devices are characterized by a limited cache size, we could state that our methods are effective for mobile 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 50 seems to be a reasonable value.

Rule confidence also affects the cache hit ratio. The performance results obtained by varying the rule confidence are depicted in Figure 4. As can be seen from the figure, very high minimum confidence values cause the cache hit ratio to decrease. This is due to the fact that a higher minimum confidence requirement means that we will have a lower number of rules, which decreases the number of predicted items. In the extreme case of a confidence of 100%, the number of rules we can have is the minimum, and the cache hit ratio is close to that of the LRU without rule extension. With lower confidence values, we have a larger number of rules. However, since we are using the rule confidence and support values for limiting the number of inferred items, the cache hit ratio does not change much for the minimum confidence value in the range of 0 to 70. This is due to the fact that we are already eliminating the low confidence rules by the priority mechanism.

The number of different patterns in the data set also has an effect on the cache hit ratio, as shown in Figure 5. LRU with rules consistently has a higher hit ratio as compared to LRU, for all different numbers of patterns. However, as the number of patterns increases, both LRU and LRU with rule extension inhibit a decreased cache hit ratio and the difference between the hit ratios also decreases. This is due to the fact that rules better represent a dataset with a lower number of patterns, and we would extract more rules with a smaller number of patterns for the same minimum confidence and support.

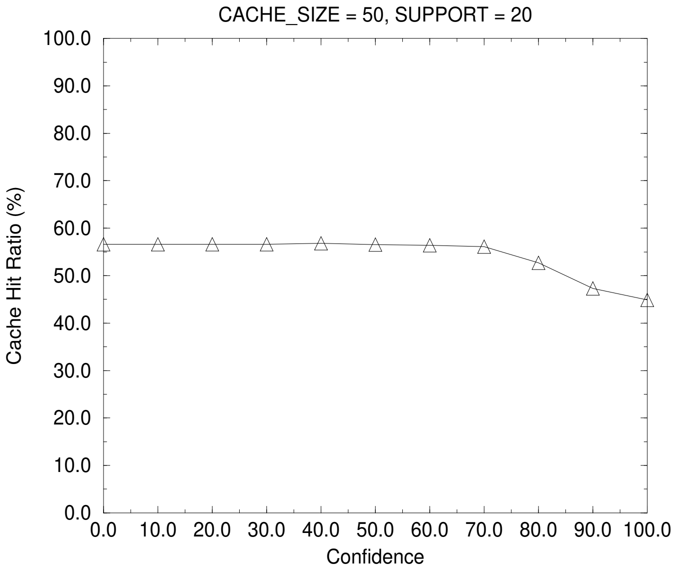

Figure 4: Cache Hit ratio as a function of the minimum confidence

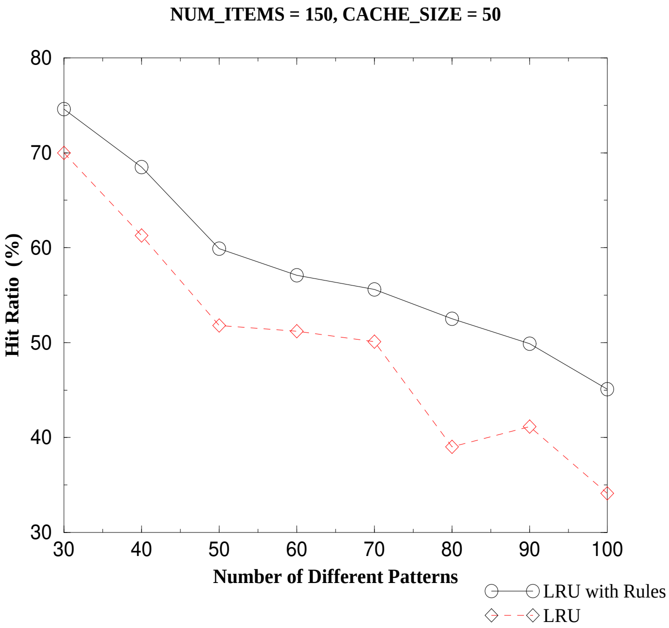

Figure 5: Cache Hit ratio as a function of the number of different patterns

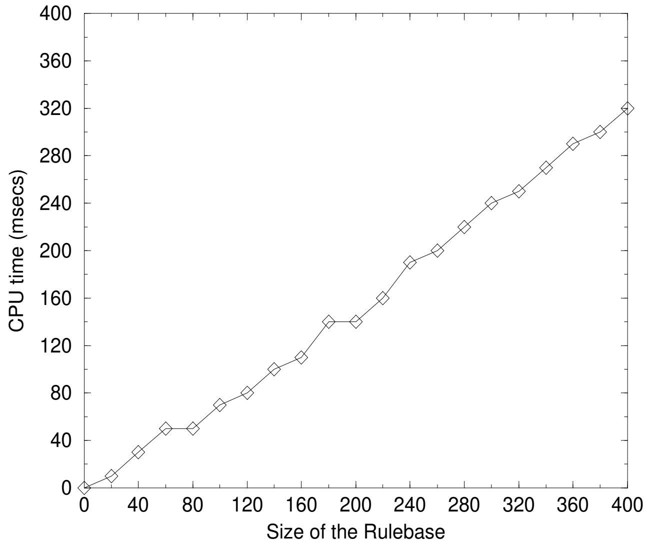

Figure 6: CPU time for inferencing as a function of the number of rules in the rulebase

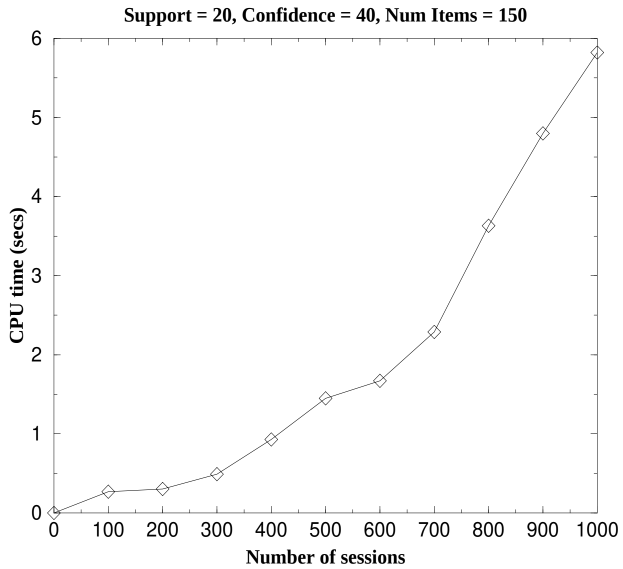

Figure 7: CPU time for rule extraction as a function of the number of session

Overhead of mining and inferencing: Rule extraction should be performed periodically to reflect changes in request patterns. However, this would lead to a processing overhead on the server in addition to the inferencing it causes. We can determine a threshold value for the waiting period before restarting the history mining and development of new association rules. For 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]. Some timing measurements are presented in Figure 6 and Figure 7. Figure 6 shows the time required for inferencing as a function of the number of rules. This timing measurement is performed over 2000 sessions and *CPU time* gives the time required to perform the inferencing 2000 times. For the time being, we search the whole rulebase to find out the matching rules. Matching is performed by a subset operation and can be performed in constant time assuming that the rule and session sizes are bounded. More efficient rule searching mechanisms could be employed to improve the performance since the inferencing time is directly proportional to the size of the rulebase. The processing overhead of rule extraction, in terms of the CPU time required to mine for rules from histories of different sizes, is presented as a function of the number of sessions in Figure 7.

Inferencing with rules is performed at the server; therefore, mobile clients are not affected by the inferencing overhead except for the server delay. With efficient incremental mining and inferencing techniques, the server load and thus client waiting times for hoarding can be decreased.

5 Conclusions and Future Work

Disconnected operation is vital for mobile clients since users disconnect frequently from the fixed part of the system, due to either the high cost of wireless communication or the unavailability of wireless links. Disconnected operation support is an important function that a mobile computing system should provide to the users, and an effective and transparent disconnected operation is the most desirable one. Hoarding is the process of loading data that will potentially be needed in the future to clients' local storage before the disconnection so that users can continue their operation as if they are connected to the network.

In this paper, we proposed a new method for automated hoarding to support seamless disconnected operation. We first analyzed the user access behavior and introduced the concept of sessions which are groups of related user requests. Our method basically utilizes the association rules via an inferencing mechanism to decide what data items to hoard. As an initial step, a method for constructing sessions out of the client request history to feed into the data mining algorithm as transactions was described. This allowed us to use data mining algorithms to analyze the history of previous client requests. We described how the hoarding process could be performed via association rules together with an inferencing mechanism to determine the candidate set of data items that may be of interest to the user in the future. A priority assignment method was proposed which exploits the confidence and support values of the association rules. The priorities were used to prune the candidate set of data items in order to develop a smaller set that will fit the limited storage of mobile clients.

Experiments were also conducted to gauge the effectiveness of the proposed methods in terms of the client cache hit ratio. We observed that the use of association rules considerably improves the hit ratio of the client cache especially for small cache size. The results are promising at this stage and can be improved further with some extensions. We also performed experiments to measure the processing overhead of rule extraction and inferencing. These results support our claim that neither infrequent mining nor inferencing introduces a considerable overhead to the system. We believe that the performance could be improved even further with a more efficient incremental mining and inferencing process.

One of the more appealing aspects of this work is that we bring old problems of production rules into the arena with a new method of rule gathering and application. We proposed the usage of the data mining concepts for different applications, such as obtaining production rules. Temporality in association rules is an important aspect which captures the temporal behavior of associations. We are planning to investigate how temporal association rules could be used for automated hoarding. The problems of termination and confluence which need to be addressed while dealing with rules are also left as future work. Efficiency of the inferencing mechanism is very important from the performance perspective of rule based hoarding. The problem we face with inferencing is the presence of multiple clients, each sending hoard requests at the same time. To handle such a case, we plan to devise an efficient inferencing mechanism that will make use of the common patterns in client request windows and perform the inferencing for all the clients in parallel. Indexing mechanisms should be used for efficient accessing and inferencing with rules. Indexing becomes an important factor in case of a large rule base.

Listed below are some more heuristics for limiting the hoard set that we are planning to test:

- Considering the size of data items for determining what to hoard: Hoarding 10 small items inferred by a lower priority rule might be better than hoarding a huge file that may consume half of the client cache inferred by higher priority rules.

- Considering the time to load a data item on the cache: In case the hoarding for weak connection, we can hoard big data items and let smaller ones be loaded by the client during the weak connection time. The "size" and "time to load" heuristics seem to contradict each other. However, for some items transmission time might be high because of their location in the network.

- Considering, if available, expiration times of data items for the hoarding process: We may not want to hoard data that will expire in the near future.

User request patterns may change over time. For that reason, the history should be analyzed to eliminate those patterns that no longer reflect more recent user request patterns. The frequency of mining the history depends on the particular system in concern. Efficient incremental mining algorithms are proposed in [CHNW96]. Dynamic nature of the client request patterns will be considered in the future by using incremental association rule mining algorithms.

References

[AAB+96] R. Agrawal, A. Arning, T. Bollinger, M. Mehta, J. Shafer, and R. Srikant. The quest data mining system. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[AK93] R. Alonso and H. F Korth. Database system issues in nomadic computing. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1993.

[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.

[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.

[Chr93] P. K Chrysanthis. Transaction processing in mobile computing environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, 1993.

[CMS97] R. Cooley, B. Mobasher, and J. Srivastana. Grouping web page references into transactions for mining world wide web browsing patterns. In *Proceedings of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop*, November 1997.

[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.

[DH95] M. H. Dunham and A. S. Helal. Mobile computing and databases: Anything new? *SIGMOD Record*, 24(4):5–9, 1995.

[JHE99] J. Jing, A. Helal, and A. K. Elmagarmid. Client server computing in mobile environments. *ACM Computing Surverys*, June 1999.

[KP97] G. Kuenning and G. Popek. Automated hoarding for mobile computers. In *Proceedings of the ACM Symposium on Operating Systems Principles*, St. Malo, France, 1997.

[KS92] James J. Kistler and Mahadev Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

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

[SU99] Yucel Saygin and Ozgur Ulusoy. Exploiting data mining techniques for broadcasting data in mobile computing environments. *Submitted for Journal Publication*, 1999.

[TLAC95] Carl Tait, Hui Lei, Swarup Acharya, and Henry Chang. Intelligent file hoarding for mobile computers. In *Proceedings of Mobicom'95*, Berkeley, CA, November 1995.