DATA MINING TECHNIQUE: MODIFICATION IN SEQUENTIAL PATTERN MINING AND DATA BROADCASTING

Vikas Jain*1, Dr.Sachin Kumar2

1Research Scholar, IFTM University, Moradabad, India.
2AKGEC, Ghaziabad, India.

ABSTRACT

Ideally the server should find out the relevant data items to broadcast on the channel so that the maximum number of user requests can be fulfilled. Several techniques some frequency based, some access probability based where frequent items are placed before the infrequent ones are in use. In our work we exploit data mining techniques; sequence mining and clustering to generate this broadcast sequence by looking into user request logs. The latency can be reduced further if there is some caching at the client side. The client can then cache data items currently on the channel which are likely to be required in the near future.

Keywords: Data Mining, Sequential Pattern, Cache management, Broadcasting

1. DATA MINING

Simply stated, it refers to extracting or mining information from large amounts of data. It deals with finding relationships among data items and grouping the related items together.

1.1 Sequential pattern is a sequence of item sets that frequently occur in a specific order, all items in the same item set are supposed to have the same transaction time value or a time interval. Usually, all the transactions of a customer are together viewed as a sequence, usually called the customer sequence, where each transaction is represented as an item set in that sequences, all the transactions are listed in a certain order with respect to transaction time.

1.2 Sequential pattern mining is the process of extracting certain sequential patterns whose support exceeds a predefined minimum support threshold. Since the number of sequences can be very large, and users have different interests and requirements, to get the most interesting sequential patterns, usually a minimum support is pre-defined by users. By using the minimum support we can prune out those sequences which are of no interest, consequently making the mining process more efficient. Sequential patterns of higher support are useful and interesting.

1.3 Mining sequential patterns: Let DB be a set of customer transactions where each transaction T consists of customer ID, transaction time and a set of items involved in the transaction. All transactions from the same customer grouped together and sorted in increasing order are called a data sequence. A support value \( \{supp(s)\} \) for a sequence
gives its number of actual occurrences in DB. In other words the support of a sequence is defined as the fraction of total data sequences that contain \( s \). A data sequence contains a sequence \( s \) if \( s \) is a sub-sequence of the data sequence. In order to decide whether a sequence is frequent or not, a minimum support value \((\text{minSupp})\) is specified by the user, and the sequence is said to be frequent if the condition \( \text{supp}(s) > \text{minSupp} \) holds.

1.4 Cache management:- Earlier Acharya & Zdonik [1] proposed the technique of pre-fetching the data from the broadcast channel. The algorithm associates a PT value with every broadcast item. The PT value is the product of probability of the access of that item with the time that will elapse before that item appears on the broadcast again. When the cache is full it finds out the item with minimum PT value in the cache and replaces with the current item on the broadcast only if the latter has the higher PT value. The probability of access of a data item is calculated by using the overall access frequencies of the individual data items. Saygin and Ulusoy [6] also used the same technique like PT, here they associate with each item, p value which is basically the confidence of the rule from which the item is inferred. The p values are halved at every tick, but here they do not consider the temporal order of the requests.

In Barbara & Imielinski [2] server sends Invalidation report to the clients when a data changes, using which client can flush the data item from the cache. User mobility is considered in Khurana & Kahol [4] where there is a home agent who takes care about its home users. It maintains home location cache for each user and transfers is to the Mobile Switching Center (MSC) of the foreign cell if a user moves to some foreign cell.

2. MOBILE DATA BROADCASTING PARAMETERS

Our Problem can be Stated as Follows: Given a database of user requests over a certain period of time, what would be the optimum sequence to broadcast? So as to minimize access latency per request, where the dynamic changing nature of the environment is modeled by certain parameters?

The broadcast strategy has to make a trade-off between two parameters: First from the client side, we need to minimize the time elapsed between a request and when it is served. But, second from the server side there is a need to minimize the cost associated with each service.

Most of the time, it happens that cost is much higher than the benefit coming from the optimization of access latency. It has to somehow manage the trade-off between the cost and the popularity of a particular service. Thus during the mining of sequential patterns both parameters, support and cost should be considered to get the patterns which have support greater than some minimum support and cost less than some maximum cost. The main parameters of our system model are given in the Table 1. Our assumptions are:

- We optimize only access latency per request i.e. looking from the client's side.
- A little storage is present at the client end which acts as a cache.
The client has little processing power.

User requests follow some probability distribution.

**Table 1: Main Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minSupp</td>
<td>Minimum support value of the extracted rules</td>
</tr>
<tr>
<td>cacheSize</td>
<td>Maximum size of the cache at the client side</td>
</tr>
<tr>
<td>cacheDecay</td>
<td>Cache decay rate</td>
</tr>
<tr>
<td>maxWaitingTime</td>
<td>Maximum waiting time at client for a particular request</td>
</tr>
<tr>
<td>suppDecay</td>
<td>Sequential pattern's support decay rate</td>
</tr>
<tr>
<td>noOfClusters</td>
<td>Number of clusters for Hypergraph clustering</td>
</tr>
<tr>
<td>aW</td>
<td>Available bandwidth</td>
</tr>
<tr>
<td>bT</td>
<td>Time between two broadcasts</td>
</tr>
<tr>
<td>maxLatency</td>
<td>Upper threshold for access latency per request</td>
</tr>
<tr>
<td>itemSize</td>
<td>Size of a particular service</td>
</tr>
</tbody>
</table>

The mining process is done at the local servers of each cell and each cell has different broadcast sequences because of different user behavior.

Bandwidth of push channel is higher than that of pull channel. Server broadcasts data over push channel and client requests data from the pull channel if the request is not fulfilled through the push channel.

**3. SEQUENCE MINING AND BROADCASTING ALGORITHM**

The overall idea of the system is taken from Saygin & Ulusoy [6] where they have proposed this kind of system for data broadcasting in mobile environments. They do not propose an algorithm nor have they considered the true dynamic nature of the system where there are frequent updates. Also there is no consideration of various parameters which have direct impact on performance and cost. Here we have taken into account most of those parameters.

**3.1 Static Sequential Pattern Mining**

In our system we implement a data structure PLWAP tree (Pre-order Linked Web Access Pattern Tree) proposed by Lu & Ezeife [5] for mining sequential patterns. This is an N-ary tree which stores some minimal information about the items in the main memory. Like apriori based algorithms it avoids multiple scanning of database as there is no candidate generation phase. At every stage it finds the suffix trees and generates sequential patterns recursively using the prefix sequences.

We have done the following modifications in the original algorithm given by Lu & Ezeife [5].
In the original algorithm they prune the tree when a node is deleted, i.e. merge the nodes having same label if they are siblings. This approach loses the information about exact position of nodes and hence misses some patterns when some previously infrequent nodes are inserted back into the tree. We do not do this kind of pruning hence preserving the exact position of the nodes. When some nodes are inserted back into the tree in later updates all previously infrequent patterns can be retrieved. The following data structures are used in the algorithm other than the PLWAP tree

**Small Code Profile:** This is a list corresponding to every small item not present in the PLWAP tree. It stores position code of the small item and its support at that position if the item was present in the tree.

**Header Linkage Table:** This is a table which has entry for every frequent item present in the PLWAP tree. With the entry of the frequent items, it stores an event queue. An event queue is the list of references to nodes in the tree having the same label as that of the item. The references are inserted through preorder traversal of the tree. The overall steps of static mining of sequential patterns are given in Algorithm 1 and its sub parts are further explained

**Table 2: Initial Database DB**

<table>
<thead>
<tr>
<th>Cust.Id</th>
<th>Service Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Ffgnpq</td>
</tr>
<tr>
<td>1001</td>
<td>Ecgpq</td>
</tr>
<tr>
<td>1002</td>
<td>Ecinq</td>
</tr>
<tr>
<td>1003</td>
<td>Egopr</td>
</tr>
</tbody>
</table>

**Input:** original database Db, minSupp

**Output:** candidate list (C,F,S), frequent patterns FS, small code profile S-profile, header linkage table L, broadcast sequence B, Plwap tree T

**Begin:**

(1) Read DB and get candidate one itemset C

(2) For each item, i ∈ C

If item, support ≥ minSupp

Put item, in F

Else

Put item, in S

**Steps of Algorithm:**

a. Consider the initial data DB of Table 2. Read the database DB and get the support of all candidate items. Those having support greater than or equal to the minimum
support given by the user are called frequent one items and other small one items. Let C, F and S be the candidate one, frequent one and small one data items in DB with minimum support of 50%.

\[C = \{c:2, e:3, f:1, g:3, i:1, n:2, o:1, p:3, q:3, r:1\}\]

\[F = \{c:2, e:3, g:3, n:2, p:3, q:3\}\]

\[S = \{f:1, i:1, o:1, r:1\}\]

b. Insert the sequences into the tree where each node contains three pieces of information’s (label : count-position : code). The root is assigned an empty label and null position code with count 0. Let \(n\) be the node to be inserted next from a sequence \(S\), start from the root and look at its children, if there is a node with label \(r_i1\) increment its count by 1 else insert \(r_i1\) as a new child of the root. Now let \(r_i2\) be the next node in the same sequence \(S\) to be inserted, start from \(r_i1\) and follow the same procedure. Tree after insertion of first two sequences from the DB and then assignment of binary position codes is shown in figure 1 and 2 respectively.

![Figure 1: After Insertion of Two Sequences](image1)

![Figure 2: Assignment of Position Codes](image2)

3. Make preorder traversal of the tree and remove items which are small. Now make their small code profile \(S — profile\). This profile is useful for inserting the item back into the tree if it becomes frequent in the next update. During this, the preorder traversal header linkage table is generated for the frequent items.

\[S-profile(f) = \{1 : 1, 1 : 11\}\]

\[S-profile(i) = \{1 : 10110\}\]

\[S-profile(o) = \{1 : 10101\}\]

\[S-profile(r) = \{1 : 10111\}\]
4. INCREMENTAL SEQUENTIAL PATTERN MINING

Here we have followed the approach suggested by Ezeife & Chen [3], which is basically an extension of Lu & Ezeife [5]. Here the PLWAP tree is used for incremental mining considering all the possible cases and looking only into the changes in the database. The algorithm does not require re-mining of patterns when new data comes, the insertion and deletion of nodes in the tree is easy. We have introduced various parameters in the algorithm like decay of support of patterns and counts of nodes so that old patterns will die with time. Another parameter called \textit{maxLatency} is introduced which determines when to include incremental data. As long as the access latency per user request is less than the \textit{maxLatency} the server will broadcast the same sequence. When it exceeds the \textit{maxLatency} the algorithm includes incremental data and generates a new set of patterns and a new broadcast sequence. We have also taken into account the maximum bandwidth available and the size of each service (data item). These parameters will decide the time to fulfill a particular service. If the available bandwidth is high the data item will be sent in less time and vice versa.

Consider the incremental database of Table 4 with minimum support of 50% i.e. 2. Let C, F, S be the previous candidate, frequent and small items respectively; C', F', S' be that of the incremental database db. Let C', F', S' represent the updated candidate itemset, updated frequent itemset and updated small itemset of updated database DB'.

Table 4: Incremental Database db

<table>
<thead>
<tr>
<th>Cust.Id</th>
<th>Service Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>Fcgpr</td>
</tr>
<tr>
<td>2001</td>
<td>Cfq</td>
</tr>
<tr>
<td>2002</td>
<td>Gefpq</td>
</tr>
<tr>
<td>2003</td>
<td>Efcopq</td>
</tr>
</tbody>
</table>

When considering the incremental database db the set of items fall in the following six categories

1. \( F \rightarrow F' \), frequent in DB and still frequent in DB'
II. \( F \rightarrow S' \), frequent in DB and small in DB'  
III. \( S \rightarrow F' \), small in DB and frequent in DB'  
IV. \( S \rightarrow S' \), small in DB and still small in DB'  
V. \( p \rightarrow F' \), new and frequent in DB'  
VI. \( q \rightarrow S'' \), new and small in DB'

Among these categories \( F \rightarrow S' \) are the items which have to be deleted from the old tree and \( S \rightarrow F' \) are the items which have to be inserted into the old tree. The algorithm looks at the changes in the database (DB) and according to those changes updates the previously built tree without rescanning the previous database. It inserts those items that were not frequent earlier but now are into the old tree using their small code profile and deletes those items which were frequent previously and now small in the updated database. The profile entry of these deleted items is inserted into the small code profile.

4.1. Construct the initial PLWAP tree \( T \) using the original data DB and get \( C, F, S, \) Small-profile. Mine \( T \) and get initial set of patterns \( FS \).

4.2. Read db and get candidate one itemset \( C \). Combine \( C \) and \( C' \) to get updated candidate one itemset \( C \) of updated database DB'. \( C = C \cup C' \). The combined minimum support \( minSupp \) becomes 4.

\[
C = \{ c:2, e:3, f:1, g:3, i:1, n:2, o:1, p:3, q:3, r:1 \}
\]

\[
C_1 = \{ f:4, c:3, g:2, p:3, r:1, q:3, e:2, o:1 \}
\]

\[
C' = \{ f:5, c:5, g:5, p:6, r:2, q:6, e:5, o:2, i:2, n:2 \}
\]

4.3. Get other parameters

\( F' = \) items in \( C \) with support greater than or equal to \( minSupp \)

\( S' = \) items in \( C \) with support less then \( minSupp \)

Out of the above mentioned categories get the items to insert and items to delete using \( S \cap C \setminus F' \) and \( F \cap S' \) respectively

\( F' = \{ f:5, c:5, g:5, p:6, q:6, e:5 \} \)

\( S' = \{ r:2, o:2, i:2, n:2 \} \)

\( \text{items\_to\_delete} = F \cap S' = \{ n \} \)

\( \text{items\_to\_insert} = S \cap F' = \{ \} \)

4.4 Reduce the support of every node of the tree as well as that of old frequent sequences so that old items and hence the old sequences will die out with time if the same item is not frequently requested in future. In this way the broadcast sequence will change if the behavior of the users changes with time.

4.5 Reduce the support of the nodes in the Small-profile also. If the support of any profile entry becomes less than or equal to zero remove that entry.
4.6 The steps 4.4 and 4.5 are applied only when the performance of system goes below maxLatency. It happens when the service requests change with time. Here for the explanation of example we are omitting these steps.

4.7 Modify the old tree by inserting and deleting the items and make header linkage table. Insert the profiles of deleted items in Small-profile. Delete \( n \) from the tree and update small code profile. Insert / into the tree using its small code profile.

4.8 Mine only the updated part of the old tree by finding the common ancestor of all the modified branches and get rules \( FS_{up} \). The common ancestor is found by looking into the position codes of the nodes inserted and deleted from the tree and find the common prefix string. This is the position code of the node which is the common ancestor. Then climb up the tree to find out the level one ancestor of the matched node. This level one ancestor represents the branch which is modified. While mining updated branch set minimum support to 1. This is done to get the patterns of non-zero support. It may happen that these patterns get combined with the same patterns found in the incremental database. If that does not happen they will be automatically removed at the time of combining all the patterns because we keep only those patterns having support greater than the combined \( \text{minSupp} \). The common prefix match of position codes of nodes f: 1:1, f: 1:11, n:f :f If 1 and n:l:101f0f is 1. Hence the common ancestor is Root. Lower minimum support to f, mine updated part of the tree for those patterns which were previously not present.

\[
V = \{g:3, \text{gq:2, gp:2, gpq:2, q:3, e:3, eg:2, egp:2, egq:1, epq:1, eq:2, ep:2, ec:2, eq:2, p:3, pq:2, c:2, cq:2, f:1, fg:1, fgp:1, fgpq:1, fq:1, fcp:1, eqp:1, ecgpq:1, ecgp:1, ecq:1, cg:1, cgq:1, cgp:1, cgpq:1, cp:1, epq:1}\}
\]

4.9 Construct small PLWAP tree from db and mine it and get frequent sequences \( FS \).

\[
FS_i = \{g:2, \text{gp:2, q:3, f:4, fq:3, fp:3, fpq:2, fc:2, fcp:2, c:2, eq:2, ef:2, efq:2, efp:2, efpq:2, ep:2, eqp:2, p:3, pq:2, c:3, cq:2, cf:2, cp:2}\}
\]

4.10 Combine the three set of sequential patterns \( FS \cup FS_{up} \cup FS \) and keep those that have support greater than equal to \( \text{minSupp} \).

\[
FS' = FS \cup FS_{up} \cup FS = \{cq:4, gp:5, pq:4, eq:4, ep:4,fq:4, fp:4\}
\]

4.11 Insert frequent sequences from db into T and update header linkage table and Small-profile.

4.12 Again construct the hypergraph and do clustering. Then for each cluster construct planar graph, remove cycles then do weighted topological sort. The final broadcast sequence obtained is \( efpqcg \).
5. CATCH MANAGEMENT

At client side both pre-fetching and cache replacement exploits the same sequential patterns obtained by data mining. These patterns are broadcast by the server to the clients periodically.

5.1 Prefetching: The client fetches the data item currently available on the channel if that item belongs to the inferred itemset assuming it would be requested in the near future with high probability. The items are ranked on the basis of their pvalue which is the combined score of support and timestamp of that item, in inferred itemset.

The algorithm looks at the data item currently broadcast and if it is in the inferred itemset and not in the cache, it inserts the item into the cache if cache is not full. If cache is full and item's calculated pvalue is greater than the minimum pvalue of the cache resident data items, cache replacement algorithm is applied. With every broadcast tick the pvalues of the data items in the cache are reduced by some decay factor cacheDecay so that old values die with time and new values get cached. This decay factor determines the relative importance of old values and the new data items.

5.2 Cache Replacement: There are two approaches to replace data items in the cache. One is support based and another is timestamp based. In the former case the data item in the cache with minimum support is replaced each time i.e. items are prioritized based on the support of the pattern from which they are inferred and in the latter case the data item with higher timestamp value are replaced, i.e. the temporal order of the requests are considered. In Algorithm, we have adopted a combination of both the methods. At the time of insertion into the cache the pvalue of the data item is calculated from both its support and timestamp value. The higher the support, higher is the pvalue, and higher the timestamp lower is the pvalue. So each time it looks for data items which are in cache and not in inferred itemset, these are the items which are the prime candidates to be replaced every time. If no such item is present, replace the data item in cache with the minimum pvalue only if this minimum pvalue is less than the pvalue of the data item to be inserted.

With all these policies we introduce a parameter, maxWaitingTime, which is the maximum waiting time a client spends waiting on the broadcast channel for the arrival of a particular data item it has requested. If the waiting time exceeds this maxWaitingTime the client sends its request from the pull channel to the server causing a miss penalty. The lower this maxWaitingTime higher would be the number of penalties and vice versa. The amount of the penalty is determined by how much time the server takes to serve a request through the pull channel. Here we have considered a parameter missPenalty which takes care of penalties.
Input: cache C, item i, Inferred itemset I

begin:

\( X \cap C - I \neq \emptyset \) then
Replace i with \( x \{C - 1\} \) with lowest p-value;

end

else

Replace i with \( x \) C with lowest p-value only if \( i.pvalue < x.pvalue \);

end

Algorithm 2: ReplaceCache

CONCLUSION

Here we have tried to exploit data mining techniques for data broadcasting and cache management in a mobile environment. We have combined multiple techniques together with various tunable parameters to obtain an optimized broadcast sequence.

Our results show that with incremental mining the time required for mining reduces significantly, without much loss in performance. The system automatically adjusts to accommodate changes in the requester behavior. The system is modular and one can easily plug other algorithms after tuning them according to the requirements of the domain.

REFERENCES


Copyright © 2012 Published by IJESR. All rights reserved

[8] Liu, Yan S, Ren J. The design of frequent sequence tree in incremental mining of sequential patterns; Software Engineering and Service Science (ICSESS), IEEE 2nd International Conference on July 2011; 679-682.