DYNAMIC ITEMSET COUNTING IN PC CLUSTER BASED ASSOCIATION RULE MINING

Ferenc Kovács
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
kovacsf@aut.bme.hu

Renáta Iváncsy
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
renata.ivancsy@aut.bme.hu

István Vajk
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
vajk@aut.bme.hu

ABSTRACT

One of the most important problems in data mining is association rule mining. It requires very large computation and I/O traffic capacity. For that reason there are several parallel mining algorithms, which can take advantage of the performance of the cluster systems. These algorithms are optimized and developed on supercomputer platforms, but nowadays the capacity of PC keeps the possibility to build cluster systems cheaper. Usage of PC cluster systems raises some issues about the optimization of the distributed mining algorithms, especially the cost of the node to node communication and cost of the synchronization. The communication costs of currently used main distributed association rule mining algorithms depends on the number of nodes with $O(n^2)$ complexity. The node synchronization is also a very important issue. The current algorithms contain too many synchronization points and this can cause performance decrease, especially in PC cluster environment.

In this paper a new distributed association rule mining algorithm is introduced, which is based on dynamic itemset counting. The communication costs of this newly developed algorithm is $O(n)$ and the nodes can work asynchronously.

1 INTRODUCTION

The association rule mining (ARM) is very important task within the area of data mining [1]. Many algorithms were developed finding association rules, but the Apriori is the best-known [2]. The main disadvantage of the Apriori algorithm is its I/O costs and the Dynamic Itemset Counting (DIC) algorithm [3] tries to reduce these costs. The paper [4] introduces an algorithm for I/O cost cutting, but it has higher memory requirements than DIC.

Because of the complexity of the ARM task several parallel algorithms have been developed. The main part of the distributed algorithms is based on the Apriori algorithm. The count distribution (CD) based algorithms [5] generate the smallest network traffic, because they send only their own counters to the other nodes. The data distribution (DD) based algorithms [5] generate higher network traffic, because the nodes send not only their local counters, but their own database as well. There are some distributed algorithms, which are not based on Apriori, for instance [6] contributes such an algorithm.

The distributed algorithms were developed and evaluated in supercomputer environment, but the PC cluster systems have several differences compared to traditional cluster systems. The traditional supercomputer clusters contain uniform nodes, which mean each of the nodes has the same performance. But a PC cluster can contain different node types due to the short development cycle of PC. In this case the synchronization points can extremely decrease the performance of the algorithms.

The paper [7] is concerned with the behavior of HPA algorithm [8] in PC cluster environment. The node synchronization and possibility of different types of nodes can cause serious performance decrease in PC clusters. The PC cluster-based modification of CD and DD algorithms were discussed in [9]. The algorithm synchronization problem was examined in [10].

Both synchronization and network traffic issues of PC cluster-based algorithms are investigated in this paper. In the interest of the asynchronous behavior and the reduction of network traffic, a new PC cluster distributed algorithm is introduced, which is based on the CD and the DIC algorithms.

This paper is organized as follows: first of all the widely spread sequential ARM algorithms are described. Afterwards the basic distributed ARM algorithms are summarized. Then a novel algorithm is discussed with special regard to the asynchronous communication and to the benefit of the DIC. Then the I/O cost of the distributed algorithms is deduced and finally we outline some test result of the new algorithm.

2 ASSOCIATION RULE MINING

In this section the formal definition of the association rule is provided [1] and some basic sequential algorithm is described, which can be used for generating association rules.

2.1 ASSOCIATION RULE

First we elaborate on some basic concepts of association rules using the formalism presented in [1]. Let $I=\{i_1,i_2,...i_m\}$ be set of literals, called items. Let $D=\{t_1,t_2,...t_n\}$ be set of transactions, where each transaction t is a set of items such that $t \subseteq I$. The itemset X has support s in the transaction set D if s% of transactions contains X, here we denote s= support(X). An association rule is an implication in the form of X→Y, where $X, Y \subseteq I$, and $X \cap Y = \emptyset$. Each rule has two measures of value, support and confidence. The support of the rule X→Y is support($X \cup Y$). The confidence c of the rule X→Y in the transaction set D means c% of transactions in D that contain X also contain Y, which can be written in $c = \frac{S(X \cup Y)}{S(X)}$ form.

The problem of mining association rules is to find all the rules that satisfy a user specified minimum support and minimum confidence. If support(X) is larger than a user defined minimum support (denoted here min_sup) then the itemset X is called large itemset. The association rule mining can be decomposed into two subproblems:

- Finding all of the large itemsets
- Generating rules from these large itemsets

The second subproblem is much easier than the first one, that is the reason why the ARM algorithms are different from each other only in the method handling the first subproblem. The table 1 contains the notations that are used in detailed descriptions of the sequential algorithms.

k itemset	An itemset having k items
L	Set of the large itemset
L_i	Set of large i itemset
C_i	Set of candidate i itemset (potentially large itemset)
|A|	Number of elements in set A
active$^-$	The candidates are being counted and the current counting does not exceed the minimum support
active$^+$	The candidates are being counted and the current counting exceeds the minimum support

Table 1. Notations in the sequential algorithms

2.2 APRIORI ALGORITHM

The Apriori algorithm use the following theorem to reduce the search space: if an itemset is large then all of its subsets are large as well. This means it is possible to generate the potentially large i+1 itemset using large i itemset. Each subsets of candidate i+1 itemset must be large itemset. Hereby it is possible to find all large itemset using database scan repeatedly. During the i^{th} database scan it counts the occurrence of the i itemset and the end of the pass i, it generates the candidates, which contain i+1 item. The figure 1 shows the pseudo code of the Apriori algorithm. The main disadvantage of this algorithm is the multiple database scan. There are many solutions to reduce the multiple database scan, but lots of them have extremely high memory requirements. The DIC algorithm tries to reduce the I/O costs by increasing the memory usage to medium size.

```
L_1 ← {1 frequent itemsets}
C_2 ← GenerateCandidate(L_1)
i ← 2
while(L_{i-1} ≠ ∅)
{
 foreach(t ∈ D)
 {
  IncrementCounter(C_i, t)
 }
 L_i ← { c ∈ C_i | c.counter / |D| ≥ min_supp }
 i ← i + 1
 C_i ← GenerateCandidate(L_i)
}
L ← ⋃_i L_i
```

Figure 1. Apriori algorithm

2.3 DIC ALGORITHM

The DIC algorithm scans the database continuously and it can generate new candidates, which have more elements, earlier than the Apriori algorithm. It reads the database from checkpoint to checkpoint and it generates new candidates at the checkpoints. If it founds new large itemsets these can be the basis of the new candidates. But of course it is also necessary to maintain the old candidates, which were generated one pass before and they are not become large. The maintenance and the candidate generation are complex tasks; therefore the distance of the checkpoints is an important issue. The optimum was reached about 10,000 read transactions as described in [3]. Figure 2 shows the pseudo code of the DIC algorithm.

```
active⁻ ← {items}
active⁺ ← ∅
L ← ∅
while (active⁻ ≠ ∅ and active⁺ ≠ ∅)
{
 foreach(t ∈ D)
 {
  IncrementCounter(active⁻ ∪ active⁺, t)
  if (checkpoint) then
  {
   break
  }
 }
 active⁻ ← active⁻ ∪ GenerateCandidate (active⁺)
 foreach(c ∈ active⁺)
 {
  if (c was introduced at the same checkpoint one round before) then
  {
   L ← L ∪ c
   active⁺ ← active⁺ \ c
  }
 }
 foreach(c ∈ active⁺)
 {
  if (c was introduced at the same checkpoint one round before) then
  {
   active⁻ ← active⁻ \ c
  }
 }
}
```

Figure 2. The DIC algorithm

3 PARALLEL ALGORITHMS

Several parallel algorithms were developed due to the complexity of the ARM task. These algorithms try to benefit the power of the parallel computation. The sequential algorithms are the basis of distributed mining algorithms. In fact, the main part of the parallel algorithms is the parallelized versions of the Apriori algorithm, which try to distribute the counting task in different ways. The used notations of the parallel algorithms are shown in table 2.

N	Number of the nodes
C_j^i	The local candidate set on node i, which contains j item
D^i	Local dataset on node i

Table 2. Notation in the distributed algorithms

3.1 COUNT DISTRIBUTION ALGORITHM

The basic idea of this algorithm is that each of the nodes keep large itemsets and counters of candidates locally, which are related to the whole database. These counters are maintained in accordance with the local dataset and incoming counter values. They locally run the Apriori algorithm and after reading through the local dataset they broadcast own counters to the other nodes then on the basis of the global counter values, so each of the nodes can generate the new candidates. Therefore each of the nodes has the candidates for the whole database and the globally large itemsets.

$$
\begin{aligned}
&C_1^j \leftarrow \{items\}\\
&i \leftarrow 1\\
&do\\
&\{\\
&\ foreach\left(t \in D^j\right)\\
&\ \{\\
&\ IncrementCounter\left(C_i^j, t\right)\\
&\ \}\\
&\ Broadcast\left(C_i^j\right)\\
&\ for\left(k = 1\ to\ N, k \neq j\right)\\
&\ \{\\
&\ \text{Receive}\left(C_i^k\right)\\
&\ ComputeCounters\left(C_i^j, C_i^k\right)\\
&\ \}\\
&\ L_i \leftarrow \left\{ c \in C_i^j \left| \frac{c.counter}{|D|} \geq \min\_supp \right. \right\}\\
&\ i \leftarrow i+1\\
&\ C_i^j \leftarrow GenerateCandidate\left(L_{i-1}\right)\\
&\} while\left(L_{i-1} \neq \varnothing\right)\\
&L \leftarrow \bigcup_i L_i
\end{aligned}
$$

Figure 3. The Count Distribution algorithm

This algorithm keeps the possibility of low data transmission, but each of the nodes is synchronized after each of the database scan. The other disadvantage of previously mentioned algorithm is that if there are too many candidates then it does not take the opportunity that there could be enough memory in whole cluster to keep all the candidates in the memory. Figure 3 shows the pseudo code of the CD algorithm and figure 4 gives an overview of the CD algorithm.

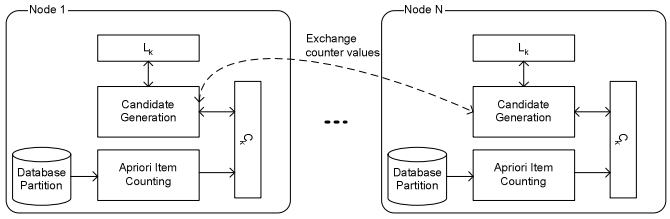

Figure 4. The overview of the count distribution algorithm

3.2 DATA DISTRIBUTION ALGORITHM

Data distribution algorithm gives a solution for a situation, when one of the nodes does not have enough memory for all of the candidates. In this case each of the nodes is responsible for only a part of the candidates. Each of the nodes counts the occurrence of its own candidates in the whole dataset. But all of these nodes have to broadcast their own local database to the other nodes. Figure 5 shows the pseudo code of the algorithm.

$$
\begin{aligned}
&C_1^j \leftarrow \{items\}\\
&i \leftarrow 1\\
&do\\
&\{\\
&\ foreach\left(t \in D^j\right)\\
&\ \{\\
&\ IncrementCounter\left(C_i^j, t\right)\\
&\ \}\\
&\ Broadcast\left(D^j\right)\\
&\ for\left(k = 1\ to\ N, k \neq j\right)\\
&\ \{\\
&\ \text{Receive}\left(D^k\right)\\
&\ foreach\left(t \in D^k\right)\\
&\ \{\\
&\ IncrementCounter\left(C_i^j, t\right)\\
&\ \}\\
&\ \}\\
&\ Broadcast\left(C_i^j\right)\\
&\ C_i \leftarrow C_i^j\\
&\ for\left(k = 1\ to\ N, k \neq j\right)\\
&\ \{\\
&\ \text{Receive}\left(C_i^k\right)\\
&\ C_i \leftarrow C_i \cup C_i^k\\
&\ \}\\
&\ L_i \leftarrow \left\{ c \in C_i \left| \frac{c.counter}{|D|} \geq \min\_supp \right. \right\}\\
&\ i \leftarrow i+1\\
&\ C_i^j \leftarrow GenerateCandidate\left(L_{i-1}\right)\\
&\} while\left(L_{i-1} \neq \varnothing\right)\\
&L \leftarrow \bigcup_i L_i
\end{aligned}
$$

Figure 5. Data Distribution algorithm

The disadvantage of the algorithm is that it generates very large network traffic, because it does not use any kind of optimization to reduce the network traffic. It can be noticed that the huge number of the candidates are decreasing during the later iteration step. Figure 6 gives an overview of the DD algorithm.

3.3 HPA ALGORITHM

The HPA algorithm is an improved version of the data distribution algorithm, it uses a hash function to determine the owner node of the candidate. With the help of this hash function it is possible to decide where to send each of the read itemsets. In this way the network traffic can be reduced.

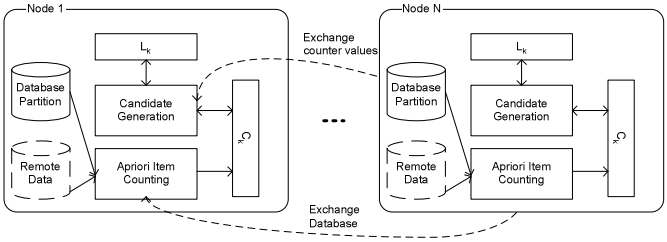

Figure 6. Overview of the Data Distribution algorithm

4 MESSAGE-BASED CONNECTION

The current distributed algorithms use synchronous communication model, and they have synchronized behavior. The recently developed algorithm uses asynchronous communication model. The benefit of the asynchronous communication is that each of the nodes can continue the data processing, while the actual communication is being done in the background. In this way time spent with communication and with data processing overlap.

The message-based connection can be used to separate data processing and communication in an asynchronous way. In this case each of the nodes has own message queue, where they can receive the incoming messages. Receiving of the messages is done in the background, in parallel with the main working process. The incoming messages are placed into a queue from where the owner of the queue can take it out, if it is needed. The process of the outgoing messages are similar, each of the nodes has own outgoing message queue where it can place its messages. There is a sender process, which takes out the outgoing messages and sends them to the appropriate node. Figure 7 shows an overview of the message based architecture. Steps of the message sending are the following:

1. During the communication a node creates a message and set the destination node and place it into an outgoing message queue
2. A background thread takes out the messages respectively and sends them to their destination node.

The processing of the incoming messages is similar:

1. A background thread receives the incoming messages and places them into their destination queue.
2. The main processing thread takes out the messages from the message queue when it needs them.

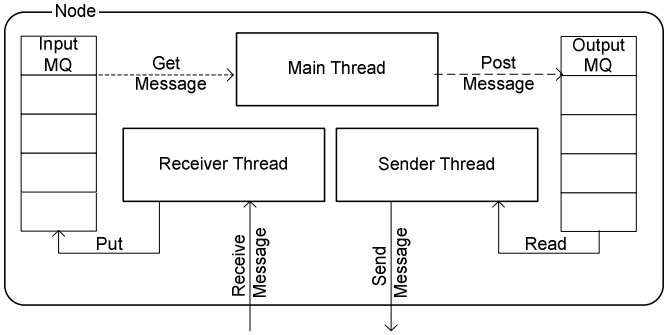

Figure 7. Message based architecture

5 DYNAMIC ITEMSET COUNTING IN THE DISTRIBUTED ALGORITHMS

The basis of the newly developed distributed algorithm is dynamic itemset counting, which keeps the possibility of asynchronous cooperation. In this way each of the nodes can work independently and they do not have to wait to each other. The asynchronous behavior can improve the efficiency of the system, especially in case of PC clusters where the nodes can have different performance. Naturally if there is a huge difference in the performance of the nodes, then the asynchronous behavior is not enough to reach the optimal performance. The improved algorithm uses the above introduced message based communication model. Table 3 contains the used notations.

C	Set of the active candidates
C_{new}	Set of the newly generated candidates
C_{delete}	Set of the erasable candidates
L_{new}	Set of the newly found large itemset
MQ	Incoming message queue

Table 3. Used notation in distributed DIC algorithm

5.1 USAGE OF THE DISTRIBUTOR COMPONENT

There are two well separable tasks during generation of the large itemsets:

1. Counting of the candidates in the local databases
2. Summarize of counters and generate new candidates

Because of the high communication costs in the PC clusters environment, broadcasting local counters does not give optimal solution [9]. Therefore, in case of reducing the network traffic second task is done by a distinguished node in the developed algorithm. As it is a complex task in a large database, the distributor node does not have any other task. According to the dynamics itemset counting the worker nodes send the changes of their counters as checkpoints and their status (under counting/ finished counting). The distributor component collects this information then generates new candidates. A candidate is removed from the list of candidates when all of the nodes finished its counting and its summarized support count does not exceed the given minimum support. Figure 8 shows the algorithm of the distributor component.

5.2 GENERATION OF THE NEW CANDIDATES

Considering the issue of generation the following question arises: when to start generating new candidates? It is a

significant point, because generating candidates needs a lot of computation. Currently the generation of the new candidates is done when each of the nodes sent their changes of their counters.

To increase the performance of the candidate generation the distributor node uses the following accelerations:

1. It stores the candidates in "trie" data structure in accordance with [3].
2. On summarizing of the counters it collects the itemsets, becoming large into a list. Hereby it lists potential sources of the new candidates.
3. An itemset can only be a candidate if all of its subsets are large. That is why the potential candidates are the new large itemset supplemented with the large items.
4. Of the potential candidates those become real candidates, whose all subsets are large.

```
C ← {items}
C_new ← C
C_delete ← ∅
Broadcast(Start)
do
{
  L_new ← ∅
  Broadcast(C_new)
  Broadcast(C_delete)
  for(k = 1 to N − 1)
  {
    Receive(C^k)
    L_new ← L_new ∪ ComputeCounters(C, C^k)
  }
  C_new ← GenerateCandidate(L_new, L)
  C_delete ← DeleteNonfrequentItemsets(C)
  C ← C ∪ C_new
  L ← L ∪ L_new
} while(C ≠ ∅)
Broadcast(Stop)
```

Figure 8.The algorithm of the distributor component

5.3 ITEMSET COUNTING

The task of the itemset counting nodes is really simple. They scan continuously their database, when they reach a checkpoint they send the changes in the counters to the distributor node. Then they read their incoming messages and process them by deleting old candidates and by inserting new ones. If there is not any incoming message for them they still can continue their work, if they have any candidates, which they have not finished counting. Therefore nodes do not have to wait for each other. Figure 9 shows the algorithm of the counter nodes.

6 I/O COSTS OF THE ALGORITHMS

The ARM algorithms are working with huge amount of data, therefore it is possible to estimate the running time by the I/O costs of algorithm. The I/O cost of algorithm contains two parts:

1. Reading database from the disk
2. Network communication

In this model every node has own disk, where it keeps the local database. The nodes are interconnected by a shared communication channel. There can be only one source and one drain on this channel at the same time. We suppose that the data is distributed uniformly among the nodes. Table 4 contains the notations used for modeling the running time.

B_D	Bandwidth of the disk
B_{NW}	Bandwidth of the network channel
$\|\|D\|\|$:	Size of the database in bytes
$\|\|C_j\|\|$:	Size of the set of the j itemset candidates in bytes
k	The number of items in the largest large itemset
m	Number of checkpoints

Table 4. Notation in the I/O cost modelling

```
WaitFor(Start)
do
{
  while(|MQ| ≠ 0)
  {
    msg ← GetMessage(MQ)
    if (msg.Id = NewCandidate) then
                C ← C ∪ msg.NewCandidate
    else if (msg.Id = DeleteCandidate) then
                C ← C \ msg.DeleteCandidate
    else if (msg.Id = Stop) then
                exit
  }
  if (C ≠ ∅)
  {
    foreach(t ∈ D^j, till the next checkpoint)
    {
      IncrementCounter(C, t)
    }
    SetStatusLocalCounters(C)
    PostMessage(C, DistributorNode)
  }
  else
  {
    WaitForNewMessage()
  }
} while(true)
```

Figure 9. Algorithm of the itemset counter node

6.1 EVALUATION OF THE COUNT DISTRIBUTION ALGORITHM

The I/O cost evaluation of CD algorithm is very simple due to the simplicity of the algorithm. The algorithm works like a sequential algorithm. First of all each of the nodes scans their own database, this means that it must read the whole database form the disk. Then the nodes broadcast own counters to each other using the network channel. These steps are repeated while the nodes find new candidates. According to this consideration the following deduction contains the I/O costs of the CD algorithm:

$$T_{read\ DB} = k\frac{\|D\|}{NB_D}$$

$$T_{NW\ pass\ j} = \frac{N(N-1)}{B_{NW}}\|C_j\|$$

$$T_{NW} = \sum_{j=1}^{k} T_{NW\ pass\ j} = \frac{N(N-1)}{B_{NW}}\sum_{j=1}^{k}\|C_j\|$$

$$T_{I/O} = T_{read\ DB} + T_{NW} = k\frac{\|D\|}{NB_D} + \frac{N(N-1)}{B_{NW}}\sum_{j=1}^{k}\|C_j\|$$

It is possible to see that the complexity of the I/O costs of the count distribution algorithm is $O(n^2)$. It is also very important to realize that the database is not sent through the network channel.

6.2 EVALUATION OF THE DATA DISTRIBUTION ALGORITHM

The modeling of the DD algorithm is not too difficult due to its simple behavior. This algorithm is also synchronous; therefore each of the nodes has to wait for the result of the other nodes. The algorithm works as follows: first each of the nodes reads own local database and increments own counter values, after all of the nodes broadcast own database to the other nodes, finally the each of the nodes broadcasts their own counter values. These steps are repeated while new candidates are generated. The consequence of these steps the algorithm has to read twice the local database: once in the counting and once in the broadcasting step. The following deduction contains the I/O costs of the DD algorithm accordance with these considerations.

$$T_{read\ DB} = k\frac{\|D\|}{NB_D}$$

$$T_{NW\ pass\ j} = \frac{N(N-1)}{B_{NW}}\frac{\|C_j\|}{N} + \frac{N(N-1)}{B_{NW}}\frac{\|D\|}{N} = \frac{(N-1)}{B_{NW}}\|C_j\| + \frac{(N-1)}{B_{NW}}\|D\|$$

$$T_{NW} = \sum_{j=1}^{k} T_{NW\ pass\ j} = \frac{(N-1)}{B_{NW}}\sum_{j=1}^{k}\|C_j\| + k\frac{(N-1)}{B_{NW}}\|D\|$$

$$T_{I/O} = 2\cdot T_{read\ DB} + T_{NW} = 2\cdot k\frac{\|D\|}{NB_D} + \frac{(N-1)}{B_{NW}}\sum_{j=1}^{k}\|C_j\| + k\frac{(N-1)}{B_{NW}}\|D\|$$

However the complexity of this algorithm is $O(n)$, but it also depends on the size of the database in $O(1)$ time. Hence it has worse performance than count distribution algorithm. It is also possible to see why the performance of this algorithm is so bad.

6.3 EVALUATION OF THE ASYNCHRONOUS DIC ALGORITHM

The evaluation of the asynchronous DIC algorithms can be done via the synchronous case. The worker nodes scan own databases, when they reach a checkpoint they send their counters to the distributor node. It is easy to prove that each frequent itemsets are sent through the network channel m+1 times, and a non frequent itemsets are sent through the network channel m+2 times, where m is the number of checkpoints. The following deduction contains the I/O costs estimation of the synchronous case.

$$T_{read\ DB} = l\frac{\|D\|}{(N-1)B_D},\ l \le k$$

$$T_{NW} = (m+2)\frac{(N-1)}{B_{NW}}\sum_{j=1}^{k}\|C_j\|$$

$$T_{I/O} = T_{read\ DB} + T_{NW} = l\frac{\|D\|}{(N-1)B_D} + (m+2)\frac{(N-1)}{B_{NW}}\sum_{j=1}^{k}\|C_j\|$$

During this deduction we did not take it into consideration that the I/O operations can overlap. If we look at the possibilities of the usage of asynchronous behavior and we can keep the system in that state, where the counting does not stop. This means there is no significant differences among the nodes. Then the I/O costs of the algorithm will be equal to the costs of the database scanning. But in this case it is possible that the database is scanned multiple times than in synchronous case.

$$T_{read\ DB} = l'\frac{\|D\|}{(N-1)B_D},\ l \le l' \le k$$

$$T_{I/O} = T_{read\ DB} = l'\frac{\|D\|}{(N-1)B_D}$$

It is possible to see,as well, that in the worst case, the I/O costs of the algorithm is $O(n)$, and database will not be send to the other nodes.

7 SIMULATION RESULTS

The asynchronous DIC-based algorithm has been implemented in ANSI C++ language (using STL), on MS Windows XP platform. The simulation took place in the PC laboratory of the department, where nodes have P4 2.2Ghz processor, 256 MByte RAM and the nodes have been interconnected by 100Mbit network. The introduced algorithm was tested on several synthetic databases [2]. The parameters of the synthetic databases were as follows:

Name	T	I	D	S
T15I8D500K	15	8	500	35.6
T15I8D750K	15	8	750	53.3
T15I8D1000K	15	8	1000	71.2

Table 5. Parameters of the synthetic databases

Where the meaning of the parameters are as follows:

T	Average transaction length
I	Average size of frequent itemsets
D	Number of transactions
S	Database size in Megabytes

Table 6. Meaning of the synthetic database parameters

Figure 10 shows the response time of the algorithm in different number of nodes. It can be realized that there is a hyperbolic effect in the function of response time. The reason of this effect can be found in the I/O costs evaluation: the reduction of the local database.

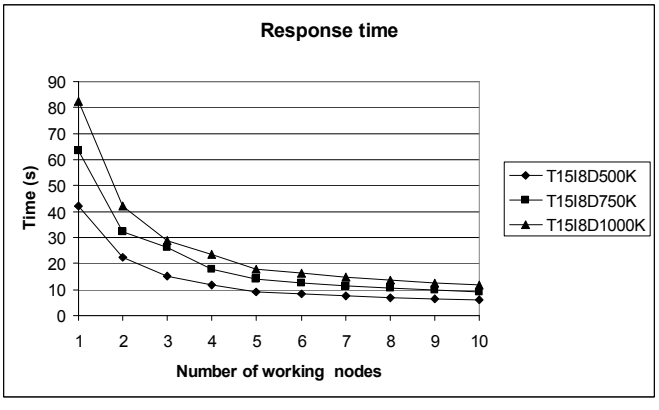

Figure 10. Response time in different number of working nodes

Figure 11 shows the relative speed up of the algorithm. The response times of the algorithm are normalized as follows: the response time of the algorithm run for several nodes is divided

by the one of the single working node case. The speed up ratio is intuitively acceptable because it exhibits almost linear tendency influenced by a subtle saturation effect.

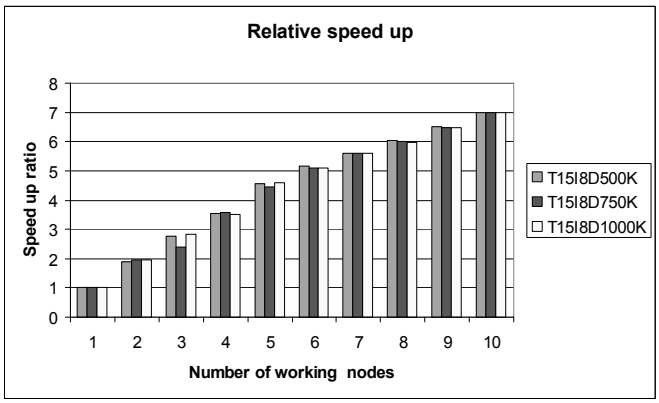

Figure 11. Relative speed up.

8 CONCLUSION

In this paper an algorithm has been introduced, which can take advantage of the PC clusters in field of ARM. The generated network traffic is moderate on PC cluster environment. Using a computation model the I/O costs of the basic ARM algorithms have been shown and the I/O cost of the novel algorithm has been also deduced. It has been also pointed out that in an optimal case I/O costs can be much lower but, taking the worst case into consideration, it has lower I/O costs than the referenced basic algorithms.

ACKNOWLEDGMENTS

This work has been supported by the fund of the Hungarian Academy of Sciences for control research and the Hungarian National Research Fund (grant number: T042741).

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami: *"Mining association rules between sets of items in large databases"*, proc. ACM-SIGMOD Conference, 1993

[2] R. Agrawal and R. Srikant: "*Fast algorithms for mining association rules",* proc. 20th Very Large Databases Conference, 1994

[3] S.Brin, R. Motawani, J.D. Ullman and S. Tsur: *"Dynamic Item set counting and implication rules for market basket data"*, proc. ACM-SIGMOD Conference, 1997

[4] J. Han J. Pei and Y. Yin: "Mining Freuent Patterns without Candidate Generation" proc. ACM SIGMOD International. Conference on Management of Data, 2000

[5] R. Agrwal and J.C. Schafer: *"Parallel mining of association rules"*, IEEE Trans. Knowledge and Data Engineering, vol. 8, no 6, 1996

[6] O. R. Zaïne, M. El-Hajj and P. Lu: "Fast Parallel Association Rule Mining Without Candidacy Generation", proc. IEEE International Conference on Data Mining , 2001

[7] M. Tamura and M. Kitsuregawa: *"Dynamic load balancing for parallel association rule mining on heterogeneous PC cluster system"*, proc. 25th Very Large Databases Conference, 1999

[8] T. Shintani and M. Kitsuregawa: *"Hash based parallel algorithms for mining association rules"*, proc. Parallel and Distributed Information Systems Conference, 1996

[9] T. Shimomura and S. Shibusawa: "Performance Evaluation of Distributed Algorithms for Mining Association Rules on Workstation Cluster", proc. IEEE Inernational Workshops on Parallel Processing (ICPP'00- Workshops), 2000

[10] J. Zhang, H. Shi and L. Zheng: "A Method and Algorithm of Distributed Mining Association Rules in Synchronism", proc. IEEE International Conference on Machine Learning and Cybernetics, 2002