# A DISTRIBUTED ALGORITHM FOR MINING ASSOCIATION RULES

Pham Nguyen Anh Huy[*], Ho Tu Bao[**]
[*] Department of Information Technology, Natural Sciences University of HoChiMinh city
227 Nguyen Van Cu Street, District 5, HoChiMinh city, VietNam
[**]Japan Advanced Institute of Science and Technnology
1-1 Asahidai, Nomi city, Ishikawa 923-1292, Japan

**Abstract.** The Mobile Agents (MA) technology has been applied in many fields such as in E-commerce, network management systems, information retrieval systems, decision support systems, etc. The MA is a powerful tool needed in many IT applications. The most significant feature of the MA technology is its ability of operating in distributed environments. In other words, MA shares computational functions of computers on the network so that computers can work together, and thus the computer's performance can be considerably improved. This paper proposes a distributed algorithm for mining association rules using the Apriori algorithm and the MA technology. The experimental evaluation of the algorithm shows the advantage of the approach.

**Keywords**: Distributed Algorithm, MA Technology, Aglets, Association Rules.

## 1. Introduction

The MA technology is currently an interesting topic in research on information systems. The benefits from MA technology are the distributive service control that reduces the load of the network, customizability, asynchronous and autonomous execution, robust and fault-tolerant, dynamical adaptability, etc. In particular, the ability of distributed computation of MA can also be used in applications in large databases requiring real-time computation such as prediction systems, data mining systems, etc.

This paper presents a distributed algorithm for mining association rules using MA technology. Section 2 provides a brief summary of the MA technology, in particular the concepts that will be used in the paper such as Aglets, and the MA technology of IBM. Section 3 starts with an illustration of distributed methods and Apriori algorithm [2], and presents a distributed Apriori algorithm for mining association rules. Section 4 presents experiments and evaluations. Section 5 gives conclusion and the future work.

## 2. Preliminaries

### *2.1. Apriori algorithm*

The basic problems of finding association rules relate to the following concepts:

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of literals, called *items*. Let D be a set of transactions, where each transaction T is a subset of itemsets, i.e., $T \subseteq I$. We say that a transaction T contains X, a set of items in I, if $X \subseteq T$.

An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \varnothing$. The rule $X \Rightarrow Y$ holds in the transaction set D with *confidence c* if c% of transactions in D that contain X and also contain Y. The rule $X \Rightarrow Y$ has *support s* in the transaction set D if s% of transactions in D that contain $X \cup Y$.

Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and conference greater than user-specified minimum support (Min_sup) and confidence (Min_conf). This problem can be decomposed into two phrases as follows:

***Phrase one***: Finding all intemsets whose support is greater than Min_sup. Itemsets with Min_sup are called frequent itemsets.
***Phrase two***: Using frequent intemsets to generate the rules.

The well-known solution to the phrase one is the Apriori algorithm for finding all frequent itemsets [2]. Figure 1 gives an overview of this algorithm in which will be used the notations given in Table 1.

| | |
|---|---|
| k-intemset | An intemset have k items |
| $L_k$ | Set of frequent k-itemsets (with Min_sup) |
| $C_k$ | Set of candidate k-itemsets (potentially frequent itemsets) |

Table 1: Notations

```
L1 = {frequent 1-intemsets};
  k = 2;
  while (Lk-1 ≠ ∅) do {
    Ck = {k-intemset generate from Lk-1 & Lk-1}
    forall transaction T ∈ D do
      increase (count of c∈Ck that are contained in T);
    Lk = All candidates in Ck with Min_sup;
    k = k +1;
  }
  Answer = ∪k Lk;
```

**Figure 1**: Apriori Algorithm

Secondly, in the prune step, it deletes all itemsets in $c \in C_k$ such that some (k-1) subset of c is not in $L_{k-1}$:

**forall** itemset $c \in C_k$ **do**
  **forall** (k-1) subsets s of c **do**
    **if** $(s \notin L_{k-1})$
      delete c from $C_k$

In our paper, the proposed algorithm is related to the phrase one. That is finding all frequent itemsets with a distributed algorithm using the MA technology.

### *2. 2. The MA technology*

#### *2.2.1 Overview*

We begin with a definition. A mobile agent is

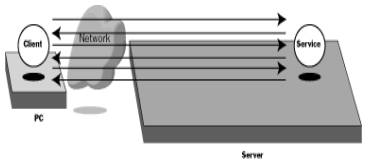


**Figure 2**: Old RPC-based client-server computing

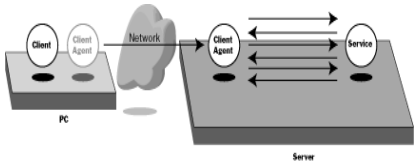


**Figure 3**: Mobile agent computing paradigm

a program that can migrate from machine to machine in the heterogeneous network. The program chooses time and place for its location. It can suspend its execution at an arbitrary point, and transport itself to another machine and resume the execution. In figure 2, a client communicates with a remote service via a network using the client-server paradigm. In figure 3, the client creates a mobile agent to do its bidding, then dispatches it to the remote service where it can interact locally with that service.

With the above definition, we must also consider the software environment in which mobile agents exit and provoke the mobile agent environment to operate. The mobile agent environment is a software system which is distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which MA can execute. The mobile agent system is illustrated in figure 4.

The mobile agent environment is built on the top of a host system. The faces stand for mobile agents traveling between mobile agent environments. The directional arrows indicate the communication between mobile agents (local and remote). The communication can also take place between a mobile agent and a host service.

We present the Aglets of IBM in section 2.2.2. This is a powerful MA environment to develop mobile applications on platforms: Windows and Unix.

#### *2.2.2 Aglets*

Aglets are Java objects or an MA moving from one host to another on the network. That is, an aglet that executes on one host can suddenly halt execution, dispatch to a remote host, and start executing again. When the aglet moves, it takes along its program code as well as the states of all objects it is carrying, **[3]**.

The MA environment of Aglets is Tahiti whose functions help Aglets work:

| Method | Behavior |
|---|---|
| dispose () | Dispose of the aglet. |
| dispatch(URL) | Dispatch the aglet to the destination specified in the URL. |
| deactivate(long duration) | Instruct the aglet to store itself into a persistent medium. |
| getAgletInfo() | Get information on the aglet |

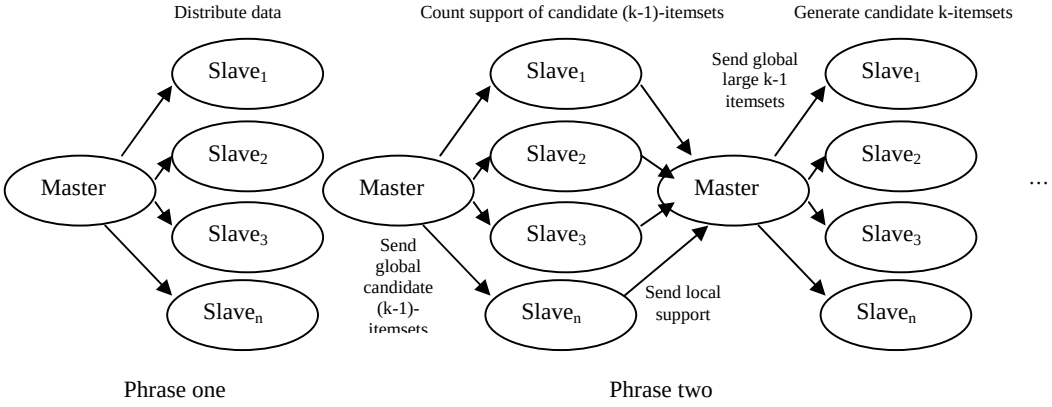


**Figure 5**: The framework of algorithm

By using the told definition and functions of Aglets, we can creat two more Aglets: counting supports and generating for the candidate k-itemsets in our algorithm. Operations of the algorithm will show in section 3.

**3. Working progress**

***3.1 Framework***

A framework of the algorithm is described in figure 5. It is decompressed into two phrases:

**Phrase one:**

*The master process*

Step 1: spawn n slave process.

Step 2: divide database into n partitions.

Step 3: distribute partitions to each slave process.

**Phrase two:**

*The master process:*

Step 1: send global candidate (k-1)-itemsets to each slave process to count supports.

Step 2: wait and receive local supports from each slave process, and then compute global supports for the global candidate (k-1)-itemsets.

Step 3: make the global large (k-1)-itemsets with minimum support.

Step 4: send the global large (k-1)-itemsets to each slave process to generate candidate k-itemsets.

Step 5: wait and receive local candidate k-itemsets from each slave process.

Step 6: unionize local candidate k-itemsets to the global candidate k-itemsets

Step 7: go on with the next loop, step 1.

*The slave process:*

Step 1: receive the global candidate (k-1)-itemsets from the master process.

Step 2: count local supports for the global candidate (k-1)-itemsets.

Step 3: send local supports to master process.

Step 4: receive the global large (k-1)-itemsets from master process.

Step 5: generate the global large k-itemsets

Step 6: send the global large k-itemsets to master process.

The master process distributes data to each slave process in phrase one. This phrase only initializes once when we start with the algorithm. In phrase two, the master process sends the global candidate (k-1)-itemsets to each slave process. The slave processes perform the same procedure to count local supports for global candidate (k-1)-itemsets as the serial algorithm; We call them counting support Aglets. After counting local supports of global candidate (k-1)-itemsets, slave processes send the results to the master. The master collects these local supports from each slave process, then calculates global support counts, and prunes them with a minimum support to make the global large (k-1)-itemsets. Next, the master continues to distribute global large (k-1)-itemsets to each slave process. Slaves processes perform the same procedure to generate into local candidate k-itemsets as the serial algorithm does. We call Apriori_gen Aglets. After generating local candidate k-itemsets, slave processes send the results to the master. The master collects these local candidate k-itemsets from each slave process, unionizes local candidate k-itemsets into global candidate k-itemsets and goes on with the next loop. The computation of global support counts and global candidate k-itemsets will be presented in section 3.2. With the above framework, we have a distributed method for mining association rules with two Aglets: counting_support and Apriori_gen.

***3.2 Global support and candidates***

The slave computes the number of local supports of candidate k-itemsets with its local database and then returns the results to the master. The global support of these candidate k-itemsets will compute as follows:

Let $X_k$ be a candidate k-itemset, the global support of $X_k$, G_Sup, is:

$$X_k.G\_Sup = \sum_{i=1}^{n} X_k.L\_Sup_i \qquad (1)$$

With $X_k.L\_Sup_i$ is local support of $X_k$ at slave-i.

Moreover, the global candidate k-itemsets, $GC_k$, is a set of candidates with k-items at the master. $GC_{i,k}$ is a set of local candidates with k-items at slave-i. The relation of $GC_k$ and $GC_{i,k}$ will be:

$$GC_k = \bigcup_{i=1}^{n} GC_{i,k} \qquad (2)$$

(1) and (2) are important steps in the master's algorithm. They will work after slave processes return all computed results to the master. Another notation, $GL_k$, global large k-itemset, is $GC_k$ whoseelements have the support larger than minimum support. We define itlike below:

$$GL_k = \{X \in GC_k / X.G\_Sup \geq Min\_sup\}$$

### *3.3 Algorithm*

#### *3.3.1 The master's algorithm*

In this section, the basic version of the algorithm is presented in figure 6, which adopts an algorithm for the master. Assume that phrase one has passed and the algorithm will do with the phrase two. The input and the output are:

Input: database, n: number of slaves, min support.
Output: frequent itemsets.

1. $GL_1$ = {frequent 1-itemsets};
2. k = 2;
3. **while** ($GL_{k-1} \neq \emptyset$) **do{**
4. **for each** i = 1 **to** n
5. sendGeneration ($GL_{k-1}$,Slave-i, From_p,To_q);
6. **wait_all**;
7. $GC_k = \cup GC_{i,k}$;
8. **for each** i = 1 **to** n **do**
9. sendCount($GC_k$,Slave-i);
10. **wait-all**;
11. **for each** $X_k \in GC_k$ **do**
12. $X_k.G\_Sup = \sum X_k.L\_Sup_{i,k}$;
13. $GL_k = \{X_k \in GC_k / X_k.G\_Sup \geq$ min support};
14. k = k +1;
15. **}**
16. Answer = $\cup GL_k$;

**Figure 6**: The master's algorithm

In the algorithm, the first line simply counts item occurrences to specify the global large 1-itemsets. In the step k, there are four phrases. First, the global large (k-1)-itemsets, $GL_{k-1}$, having transaction ID from From_p to To_q, found in the $(k-1)^{th}$ pass are sent to n slave processes to generate the local candidate k-itemsets, $GC_{i,k}$ using the Apriori-gen Aglets described in section 3.3.1. Next, the master waits for all results of salve processes to unionize $GC_{i,k}$ to make global candidate k-itemsets, $GC_k$. Continuously, the algorithm sends $GC_k$ to slave processes to count local supports, using counting_support Aglets described in section 3.3.3, and also waits for all results of slave processes to make global large k-itemsets, $GL_k$, with minimum support. Finally, the master goes on with the next loop. Apriori-gen Aglets and counting_support Aglets are MAs that dispatched to slave processes for computations. They improve efficient operations of the master.

#### *3.3.2 Candidate generation*

At slave-i, Apriori-gen Aglets takes two steps: first, in the join step, join $GL_{k-1}$ and $GL_{k-1}$ with the transaction ID of $GL_{k-1}$ from From_p to To_q:

**insert** into $GC_{i,k}$
**select** $p.i_1, p.i_2, \ldots, p.i_{k-1}, q.i_{k-1}$
**from** $GL_{k-1}$ p, $GL_{k-1}$ q
**where** $p.i_1=q.i_1, \ldots, p.i_{k-2}=q.i_{k-2}, p.i_{k-1}<q.i_{k-1}$, p.ID≥From_p, p.ID≤To_q, q.ID≥From_p, q.ID≤To_q.

Next, in the prune step, delete all itemsets in $c \in GC_{i,k}$ such that some (k-1) subset of c is not in $GL_{k-1}$:

**forall** itemset $c \in GC_{i,k}$ **do**
**forall** (k-1) subsets s of c **do**
**if** ($s \notin GL_{k-1}$)
delete c from $GC_{i,k}$

#### *3.3.3 Support counts*

At slave-i, the counting-support Aglets simply scans its local database to count local supports of candidate k-itemsets. As the global database is partitioned over n slave processes into local databases so that the space of local databases are not too large to scan, especially when we use many machines on the network.

## 4. Experiments and evaluations

In this section, we present experiments and evaluations of the experiments. The content of this section contains: designing data and configuration of experiments, next, results and final evaluations. We start now with the first section.

### *4.1 Design configuration and data*

The configuration of hardware and software used in experiments are shown in table 2.

| | |
|---|---|
| Software | Database: Oracle server<br>Language: Java<br>Mobile Agents Environment: Aglets<br>Protocol traffic: ATP<br>Platform: Windows |
| Hardware | 15 Machines<br>PC Pentium III – 300MHZ<br>RAM 128MB |

**Table 2**: The configuration of software and hardware

We used synthetic datasets of varying complexity. The synthetic data is told to simulate customers buying patterns in a retail environment. The characteristics of three datasets that we used in the experiments are shown in table 3:

| Name | \|D\| | \|T\| | Size (MB) |
|---|---|---|---|
| D100k.T30 | 100K | 30 | 3M |
| D100k.T100 | 100K | 100 | 10M |
| D320k.T150 | 320K | 150 | 48M |
| \|D\| Number of transactions<br>\|T\| Average amount of items for a transaction | | | |

**Table 3**: Data of experiments

The largest data set is D320.T150, about 48MB in size, with 320000 transactions and 150 items for each transaction.

### *4.2 Results*

In the experiments, we examine with scale up of number of slaves, size up of datasets and difference between minimum supports. In the case of one slave process, it is actually the Apriori algorithm. The results and the execution time are very encouraging in table 4, 5 and 6 as following:

| Name | 1 Salve | 5 Slaves | 10 Slaves | 15 Slaves |
|---|---|---|---|---|
| D100k.T30 | 80860 | 42558 | 22079 | 15838 |
| D100k.T100 | 155440 | 77720 | 41080 | 28062 |
| D320k.T150 | 329532 | 147673 | 76432 | 53318 |

**Table 4:** The execution time (Sec) with min_sup = 35%

| Name | 1 Salve | 5 Slaves | 10 Slaves | 15 Slaves |
|---|---|---|---|---|
| D100k.T30 | 4158 | 1980 | 1149 | 988 |
| D100k.T100 | 30005 | 15792 | 7978 | 5843 |
| D320k.T150 | 69011 | 28425 | 18349 | 12854 |

**Table 5:** The execution time (Sec) with min_sup = 40%

| Name | 1 Salve | 5 Slaves | 10 Slaves | 15 Slaves |
|---|---|---|---|---|
| D100k.T30 | 485 | 244 | 218 | 95 |
| D100k.T100 | 27012 | 13506 | 7047 | 5062 |
| D320k.T150 | 52322 | 20259 | 11979 | 9112 |

**Table 6:** The execution time (Sec) with min_sup = 50%

From above tables, correlations of execution time with different number of slave processes are showed by graphs in figure 7, 8 and 9.

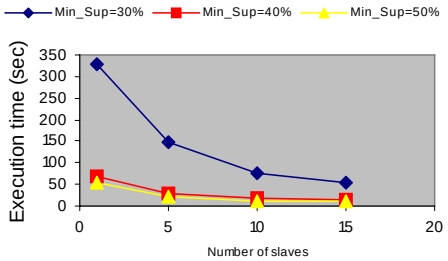


**Figure 7**: The correlation of D320K.T150

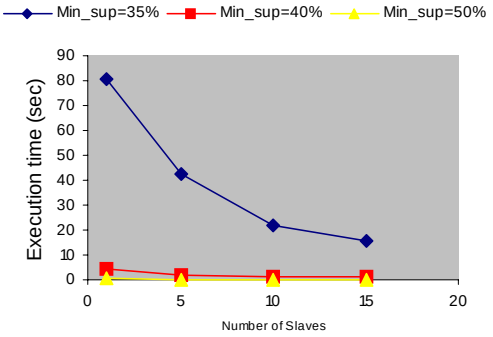


**Figure 8**: The correlation of D100K.T30

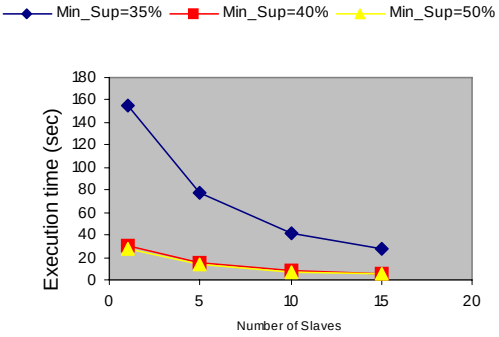


**Figure 9**: The correlation of D100K.T100

### *4.3 Evaluation*

In figure 7, 8 and 9, the chart indicates decrease with increase in the number of slave processes. In other words, the execution time is decreased when we increase the number of slave processes. For example, with data of table 4, 5 and 6, we have table 7 that shows rates of execution time of each one slave and remain number of slave processes for D100.T30 the number of slave processes is maintained. The figure 10 presents the graph of their correlation.

| Slaves | Min_Sup | | | |
|---|---|---|---|---|
| | 35% | 40% | 50% | Average |
| 15 | 5.1 | 4.2 | 5.1 | 4.8 |
| 10 | 3.6 | 3.6 | 2.2 | 3.13 |
| 5 | 1.9 | 2.1 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 |

**Table 7:** Rates of execution time between 5, 10 and 15 slaves with one slave for D100K.T30

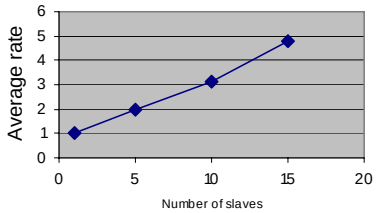


**Figure 10**: The rate of execution time

In the above graph, the rates of execution time between slave processes are an approximately linear equation. For example, the rate of execution time is 1:5 for one and fifteen slaves. This means that if we use fifteen slave processes, then the algorithm operates five times faster than one slave and so on. In the cases of D100K.T100, D320K.T150, they also have similar rates as in table 8, 9 and figure 10, 11.

| Slaves | Min_Sup | | | |
|---|---|---|---|---|
| | 35% | 40% | 50% | Average |
| 15 | 5.5 | 5.1 | 5.3 | 5.3 |
| 10 | 3.8 | 3.8 | 3.8 | 3.8 |
| 5 | 2 | 1.9 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 |

**Table 8:** Rates of execution time between 5, 10 and 15 slaves with one slave for D100K.T100

| Slaves | Min_Sup | | | |
|---|---|---|---|---|
| | 35% | 40% | 50% | Average |
| 15 | 6.2 | 5.4 | 5.7 | 5.7 |
| 10 | 4.3 | 3.8 | 4.4 | 4.1 |
| 5 | 2.2 | 2.4 | 2.6 | 2.4 |
| 1 | 1 | 1 | 1 | 1 |

**Table 9:** Rates of execution time between 5, 10 and 15 slaves with one slave for D320K.T150

## 5. Conclusion and future work

We have presented a distributed algorithm for mining association rules using the MA technology to improve efficient operations while finding frequent itemsets.

Our future work consists of discovering rules from these frequent itemsets that use the MA technology and of publishing them on the Internet.

## 6 References


[1] Michael Knapik and Jay Johnson, 1998, Developing Intelligent Agents for distributed systems, MCGraw-Hill.

[2] Rakesh Agrawal, Ramakrishnan Srikant, (1994), Fast Algorithms for Mining Association Rules, Proc. 20th Int. Conf. Very Large Data Bases, VLDB.

[3] Aglets homepage, http://www.trl.ibm.com/aglets/

[4] Shaw Green and al, 1997, Software Agents:A review, http://www/cs/tcd.ie/Brenda.Nangle/iag.html.

[5] David W.Cheung and al, 1996, A fast Distributed Algorithm for mining Association rules, International Conference on Parallel and Distributed Information Systems.

[6] Yuan Chen, 1998, An Efficient Parallel Algorithm for mining association Rules in Large Database, http://www.cc.gatech.edu/~yuanchen/.