ORIENTAL JOURNAL OF
COMPUTER SCIENCE & TECHNOLOGY
An International Open Free Access, Peer Reviewed Research Journal
Published By: **Oriental Scientific Publishing Co., India.**
www.computerscijournal.org

ISSN: 0974-6471
September 2013,
Vol. 6, No. (3):
Pgs. 321-331

Parallel Mining Association Rules in Calculation Grids

KHADIDJA BELBACHIR and HAFIDA BELBACHIR

Computer Sciences Department, University of Sciences and Technologies of Oran USTO, Algeria.

(Received: November 22, 2012; Accepted: July 14, 2013)

ABSTRACT

As a result of the physical storage expansion and backup equipment and the increasing need to store more data, the sequential searching algorithms of association rules have been found ineffective. Thus the introduction of new parallel versions has become a necessity. We propose in this paper, a parallel version of a sequential algorithm Partition. This is fundamentally different from other sequential algorithms, as it scans the database only twice to generate all significant association rules. Consequently, the parallel approach does not require much communication between sites. The proposed approach was implemented for an experimental study. The results obtained show a large gain in execution time compared to the sequential version.

Key words: Association rules, Distributed data mining, Partition, Parallel algorithms.

INTRODUCTION

The tasks of data mining are expensive treatments, and the available data volumes increase with storage possibilities, imposing recourse to parallelism, from where the birth of Distributed Data Mining (DDM).

DDM is a traditional data mining process which consists of extracting a new knowledge from partitioned data sources, distributed on different sites, each site applies a Data mining algorithm on local data. The results are then combined with a minimum of interaction between data sites. The majority of DDM algorithms are designated on the potential parallelism which they can apply on the available distributed data. Generally, the same algorithm works on each site of distributed data at the same time, producing a local model for each site. Afterward, all local models are combined to produce the final model. Primarily, the success of DDM algorithms resides in a minimal transfer of data. Among the algorithms of distributed data mining are: distributed classification[1-7]distributed clustering[8-12] and distributed association rules.

Many parallel and distributed data mining algorithms were proposed in order to extract association rules. We distinguish two great families. The first is data Parallelism, such as CD (Count Distribution), FDM (Fast Distributed Mining), DMA (Distributed Mining of Association Rules) and ODAM (Optimized Distributed Association Mining). The second is Task Parallelisms, we can quote, DD (Data Distribution), IDD (Intelligent Data Distribution) and HPA (Hash Partioned Apriori).

There are other algorithms which cannot be classified in these two families, like HD (Hybrid Distribution), CAD (Candidate Distribution) and ECLAt.

In this paper, we present a new parallel algorithm which differs from other algorithms by proposing the parallelization of a sequential algorithm called Partition[13]. This latter requires two scans of database, which will make it possible to reduce the communications number thus the addition of a synchronization phase between sites in the parallel version of this algorithm. We chose a centralized architecture to minimize the communication cost, where each site treats independently its database during the two scans and sends results to the coordinator which will be in charge of cumulating the results.

The paper is organized as follows: the next section is an introduction to the extraction of associative rules. In section 3, we study the performance of best known parallel algorithms for extraction of association rules in distributed environments. Thereafter, we present the proposed parallel algorithm in section 4. The experiments and comparative results of the parallel algorithm are shown in section 5 with comments. We will conclude with future works in section 6.

Association rules extraction

The extraction techniques of association rules are not supervised learning methods; they allow the discovering from a set of transactions, a set of rules which expresses a possibility of association between different attributes. Taking the example of supermarket where the articles bought by each consumer are recorded in the database as transaction.

From this example, we can find associative rules of the form "90% of clients who buy chocolate and milk tend to buy cheese". Chocolate and milk constitute the antecedent of this rule, cheese is the consequence and 90% is confidence. The associative rules were used successfully in many fields like the planning aid, the diagnosis aid in medical research, the improvement of telecommunication processes, the organization and access to Internet sites, the analysis of images and spatial data, statistical and geographical applications, etc.

Definitions

Consider I = {I1, I2…,Im} a set of m items and D a set of transactions constituted by the items (II, IJ , IK of I).

An association rule is an implication of the form X=>Y where X and Y are included in I and X)" Y = Ø. X is called condition or antecedent and Y consequent. Two measures are defined for an associative rule: the support and confidence. The support is the report of a transactions number for which condition and conclusion appear at the same time on the total number of transactions and confidence is the report of transactions number for which a condition and a conclusion appear at same time on the transactions number for which at least one condition appears.

Association rules research

The association rules search consists of finding the rules whose support and confidence are respectively superior to the thresholds minimum of support MinSup and confidence MinConf fixed by the user. The discovery of association rules is done in two phases:

Research of the frequent itemsets

It consists of scanning iteratively the transactions set. For each iteration, a set of candidate itemsets is created. The supports of these itemsets are calculated and the non-frequent itemsets are removed (itemset which have a support inferior to MinSup). This phase is expensive in processing time because the number of frequent itemsets depends exponentially on the items number manipulated for the N items. Thus, we have 2N potentially frequent itemsets.

Research for the interesting rules in the itemstes

This phase use the frequent itemsets set to deduce the searched rules. Thus, if ABCD and AB are frequent itemsets, then we can generate a rule AB=>CD. To know if this rule is to be retained or rejected, we calculate the confidence of AB=>CD by the formula: conf = sup (ABCD)/sup (AB). If conf e" minconf, then the rule is retained.

Sequential algorithms

The extraction algorithms of frequent itemsets proceed in an iterative way. During the Kth iteration, we search the frequent K-itemsets, i.e. the itemsets of length K Which are frequent, in the following iteration, it (K+1)th, we will search the frequent (K+1)-itemsets and so on.

The first algorithms for extracting the frequent itemsets are AIS [14], Apriori [15] and SETM [16]. They were proposed in 1993. Other algorithms to reduce the time for extracting frequent itemsets then appeared, and several optimizations and data structures to improve the Apriori algorithm efficiency, include:

- AprioriTid by Agrawal and Srikant in 1994. [15]
- AprioriHybrid by Agrawal and Srikant in 1994. [15]
- DHP (Direct Hashing and Pruning) by Park and Al in 1995 [17]
- Partition by Savasere and Al in 1995. [13]
- Sampling by Toivonen in 1996. [18]
- DIC (Dynamic Itemsets Counting) by Bit and Al in 1997. [19]

To limit the candidates' number considered at each iteration, the Apriori algorithm (and its derivatives) is based on the following two properties:

Property 1

All subsets of frequent itemsets are frequent. This property makes it possible to limit the number of generated candidates during the Kth iteration by the conditional join of K-1 size frequent itemsets, discovered previously.

Property 2

All supersets of infrequent itemsets are Infrequent. This property helps removing a K-size candidate when at least one of its K-1 size subsets is not part of the previously discovered frequent itemsets.

Mining association rules in a distributed environment

As a result of the expansion of storage physical supports and the increasing needs to store more and more data, the sequential algorithms of researching association rules have been proved ineffective. Thus the introduction of new parallel versions became imperative. The current parallel and distributed algorithms are based on the sequential algorithm Apriori. Two paradigms of parallelism are used: data parallelism and task parallelism. The parallelization requires a new property in addition to the two properties previously seen in the sequential extraction of association rules.

Property 3

So that an itemset is globally frequent it is necessary that it is locally frequent on a site at least.

Data's Parallelism

In this type of paradigm, each node counts the same candidates' set. The sets are duplicated on all processors and the database is distributed through these processors. Each processor is responsible for calculating local supports of all candidates which are the supports in its database partition. All processors calculate then the global supports of candidates which are the total supports of candidates in the entire database by the exchange of local supports. Thereafter, the frequent itemsets are calculated by each processor independently.

CD (Count Distribution) algorithm proposed in [20] is based on the Apriori algorithm. It is realized in a distributed environment where each site treats its local portion of transactions' database, calculates local supports of the candidates and broadcasts them to other sites to calculate the global support.

Consequently each processor generates the same global set of frequent itemsets which is important at each step, so; the replication of calculation decreases performance. CD has a simple communication system to exchange local supports; it uses communication based on a general distribution (Broadcast all-to-all). On the contrary it suffers from a strong cost of communication due to general distribution for exaggerated set of generated candidates in each step.

To cover these limits, other parallelization of Apriori algorithm, FDM (Fast Distributed Mining) algorithm [22] proposed new techniques to deduce the candidates' number considered for computing the support. The first, local pruning which consists in deleting element X of candidates set on site if X is not locally frequent in its local database. Once that local pruning is realized, each site diffuses the supports of the remaining candidates to other sites. At the end of each iteration, each site adds these local supports to generate global frequent itemsets from total database, this technique is called global pruning. The FDM algorithm minimizes the generated candidates set which permit to reduce the number of messages transmitted between sites and execution time.

DMA (Distributed Mining Algorithm) algorithm[21] uses pruning technique at each site and retains only frequent itemsets. The local supports of frequent itemsets on each site are employed to decide if a frequent itemset is heavy (locally frequent in partition and globally frequent in the entire database). DMA generates reduced set of itemsets then it generates candidates from the heavy frequent itemsets. DMA introduces a new technique for optimizing communication in order to reduce messages size and to avoid their duplication. Instead diffusing local supports of all candidates as in CD, DMA associates each itemset a voting site which is responsible for collecting its local supports.

Another algorithm ODAM (Optimized Distributed Mining Association)[23] follows the CD and the FDM paradigm. The difference is that ODAM eliminates all global infrequent 1-itemsets from each transaction and inserts new transaction in memory. It starts then on the new transactions into memory and it generates the candidate itemsets supports of successive length. Then, the global frequent itemsets corresponding to this length by diffusing supports of candidate itemsets after each pass. ODAM proposes another improvement in communication schema reducing messages exchanges by sending support of candidate itemsets to a simple site called receiver. The receiver announces global frequent itemsets to distributed sites.

Tasks Parallelism

All candidates are partitioned and distributed through processors as well as the database. Each processor is responsible to maintain only global supports of candidates' subset. This approach demands two communication scans in each iteration. In the first one, each processor sends its database partition to all other processors. In the second pass, each processor diffuses frequent itemsets that it found to all other processors to calculate candidates of the next iteration.

These algorithms include DD (Distribution Data)[20]], which is an algorithm derived from Apriori. It differs from CD in the sense where processors do not share the job by database portion but by candidates. The itemsets supports are always calculated locally at each iteration. Then, between two iterations we broadcast the local partition of data. The major advantage of DD is that it is conceived for better exploiting all the memory. However, this algorithm implies a considerable quantity of communication between all processors to exchange their local partitions at each step. Thus, It suffers from a strong cost of communication due to this broadcasting.

In order to reduce the cost and charge in communication for this algorithm, a new version was proposed considering the processors as a logical ring: IDD (Intelligent Data Distribution) [26]. It is an improvement of the DD algorithm. Firstly, instead of a "round robin" partitioning of the candidate, IDD partitions candidates to processors based on the first item of candidates, i.e. the candidates with the same prefix item will be put in the same partition. Therefore, before the treatment of any transaction each processor must ensure that it contains the appropriate prefixes that are assigned to this processor. Otherwise, the transaction can be rejected.

The entire database is always communicated, but a transaction is not treated if it does not contain the appropriate itemsets. That reduces the redundant calculation in DD, and then each processor must check all subsets of each transaction. To realize a charge balancing in the candidates' distribution, it calculates initially for

each item the candidates number starting with this item and it assigns the itemsets to candidates' partitions so that the number of candidates in each partition is the same. It also adopts a ring architecture to improve performance and to reduce the cost of communication, i.e. it employs the asynchronous communication from point to point between neighbors in a ring instead of broadcasting.

Another algorithm HPA (Hash Partitioned Apriori)[24] is conceived to improve the IDD algorithm. Each processor generates k-itemsets candidates using (k-1)-itemsets frequent from the previous step. The hash function is then applied to the candidate itemsets to determine the ID of their host processor. Each candidate itemset is then inserted into a hash table of its host processor. In the next step, each processor calculates support of candidates k-itemsets from transactions stored on its local disk by eliminating those whose support is lower than MinSup. The hash function is then applied to each k-itemset to determine the ID of its host processor. Each processor is then responsible to increment the support value of itemsets generated locally and those sent by other processors to determine the frequent itemsets. As a result, the number of communications is reduced because all transactions in the database are not diffused.

Other types of parallelism

There are other algorithms that cannot be classified in the two paradigms. Eclat algorithm[25] was introduced by M.J.Zaki in 1997, it is based on database division into equivalence classes and workload distribution on all processors. It begins by a database scan to construct the frequent itemsets of size 2 (L2).

Then, it partitions L2 into equivalence classes which will be redistributed on processors with a balancing policy. It creates a vertical transformation of the database, it is then necessary to transmit respectively transactions lists to processors responsible of the corresponding equivalence class. In parallel, each processor generates independently frequent itemsets by intersecting transaction lists of each element in an equivalence class between them. The final task of algorithm consists in accumulating results of each processor. The most expensive task of the Eclat algorithm is generally the transmission of transaction lists with a significant size of each item to other processors. Moreover, this algorithm requires a considerable time to partition the items set into equivalence classes and to transform the local database into a vertical base.

Another proposed solution combines CD and IDD, HD (Hybrid Distribution)[26]. It partitions P processors in G groups of equal sizes, where each group is considered as super-processor. IDD algorithm is used within groups and CD algorithm between groups. The database is partitioned horizontally between super-processors G and candidates are divided between P/G processors in group. Additionally, HD adjusts the groups' number dynamically on each pass. The advantage of HD is that it reduces the cost of database communication and tries to keep the processor occupied.

Parallel partition algorithm

Many parallel and distributed algorithms were proposed for distributed mining of association rules. The objective of these is to reduce the communication cost which constitutes an important factor to measure their performance. However, the majority of these algorithms usually present a high number of data scanning and multiple phases of synchronization and communication which degrades their performances.

To resolve these disadvantages we chose the sequential algorithm Partition[13] and we focused on the parallelization of this algorithm. This latter requires only two database scans, which permit to reduce the communication number in the parallel version of this algorithm.

Partition Algorithm

Proposed by A.Savasere, E.Omiecinski and S. Navath in 1995, the approach adopted by this algorithm is to divide the database into horizontal partitions of the same size, where the size of each partition depends on the memory size so that the entire partition can reside in memory with intention that it is read only once in each phase, also depends on transactions average size, current

minimal support threshold and items number. It is executed on each subset of transactions (Partition) independently, producing in1st scanning a set of frequent local itemsets for each partition. The set of global candidates is formed by the union of all sets of frequent local itemsets. To obtain the total support for these itemsets, a 2nd scan of database is necessary.

The following pseudo code presents Partition algorithm phases:

```
P = partition_database (D)
n = partitions Number
// Phase I
      for i = 1 to n do begin
         read-in-partition (p_i ∈ P)
         Li = gen-large-itemsets(p_i)
      end
// Merged Phase
      for (i=2; L_i^P ? Ø , j=1,2,... n; i++) do begin
         C_i^G = ∪_{j=1,2,...,n} L_i^j
      end
// Phase II
      for i= 1 to n do begin
         read-in-partition (p_i ∈ P)
         For all candidates c ∈ C_i^G gen-count (c,p_i)
      end
L^G = { c ∈ C_i^G | c.count ? minSup }
Answer = L^G
```

Fig. 1: Pseudo code of partition algorithm phases

Phase I

Takes N iterations, during iteration i only the partition pi is considered. The procedure gen_large_itemsets presented below takes partition pi for entry and generates the frequent local itemsets of all lengths (L_2^i , L_3^i … L_l^i).(see figure2)

```
Procedure gen-large-itemsets (p: database partition)
L_1^P = {large 1-itemsets along their tidlists}
for ( k=2 ; L_k^P ? Ø ; k++ ) do begin
  for all itemsets l1 ∈ L_{k-1}^P do begin
    for all itemsets l2 ∈ L_{k-1}^P do begin
  if l_1[1]=l_2[1] ? l_1[2]=l_2[2] ? ... ? l_1[k-1]<l_2[k-1] then
      c = l_1[1].l_1[2] ... l_1[k-1] . l_2[k-1]
        if c cannot be pruned then
          c.tidlist= l_1 . tidlist n l_2.tidlist
          if | c.tidlist| / |p| ? MinSup then
            L_k^P = L_k^P ∪ {c}
  end
  end
 end
Return ∪_k L_k^P
```

Fig. 2: Pseudo code of phase 1

The transactions are in the form: <TID , I_j , I_k… I_n>, transaction elements are supposed to be sorted in lexicographical order, The items in the itemset are also conserved sorted depending on this order.

To determine at same time the frequent itemsets for each partition, and to count global supports during the counting phase, Partition algorithm employs new data representation TID-list, TID-list associates to each itemset X a list of all transactions' TID where this itemset appears. In other words the database is transformed into pairs <X, TID-list>. TID-lists are sorted so that their intersection can be calculated in a simple manner during joint's phase, which requires just one read of the two lists. TID-lists change in each pass and can be permuted in disk if there is not sufficient available memory to store them. Moreover, they are used for the elimination of all useless data, because TID-lists of not-frequent items can be removed easily. The example in figure 3 illustrates the initial database' presentation and the intermediate results; the minimum support is equivalent to 3 occurrences.

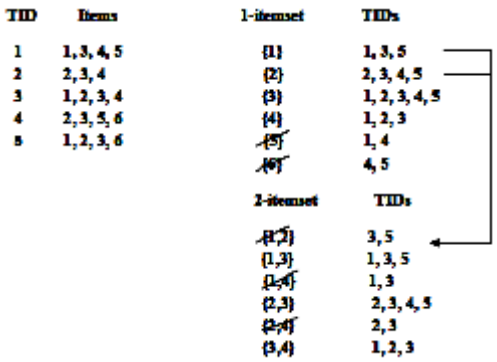

Fig. 3: Representation of TIDs and TID-Lists

After the generation of the frequent local itemsets, the fusion phase joins itemsets with the same length for all N partitions to generate a set of the global candidate itemsets. The global candidate itemsets' set of length j is calculated as: $C_j^G = \bigcup_{i=1,\ldots,n} L_j^i$.

Phase II

The gen_final_count procedure illustrated in figure.4 uses a counter for each global candidate itemset, calculates its support in the

database and generates the frequent global itemsets.

```
Procedure gen-final-counts (CG : global candidates set,
p : partition)
forall 1-itemsets do generate its tidlist
  for (k=2; C_k^G ? Ø;k++) do begin
          forall k-itemset c ∈ C_k^G do begin
   templist=c[1].tidlist ∩ c[2].tidlist ∩ ... ∩ c[k].tidlist
              c.count = c.count+ |templist|
  end
end
```

Fig. 4: Gen_final_count procedure

Parallel algorithm of Partition

We describe in this section the parallel approach of Partition algorithm which can be effectively applied for association rules' extraction in a distributed environment.

In our approach the adapted topology is viewed as a set of N sites (clients) managed by a site called the coordinator (server). The parallel algorithm as shown in figure.5 executes in four phases:

Fig. 5: Parallel algorithm of Partition

Phase 1

Each processor has a partition, it applies the gen_large_itemsets procedure on local data in order to identify a set of locally frequent itemsets Li, and then it sends the result to the coordinator. As local data size in all nodes is approximately equal, this phase will take approximately an equal time in all nodes by realizing a charge balancing. Moreover, only data available locally are treated. Consequently, there is no communication cost for the synchronization or data transfer between nodes.

Phase 2

After having received the result of phase 1, the coordinator identifies the global candidates' set CG, by the union of locally frequent itemsets of each processor (treatment realized in merged phase of Partition algorithm). The result is passed to all processors. In the end of this phase, all nodes will have exactly the same set of candidate itemsets.

Phase 3

corresponds to phase II of the sequential algorithm, where each processor must determine the support for all itemsets in CG, then to send them to the coordinator.

Phase 4

After having received result from phase 3, the coordinator will identify the set of global frequent itemsets by supports addition of each itemset.

```
Entry: Database D,the number of sites P,the minimal support
minsup
Exit: Set of frequent itemsets L^G.
Coordinator : partitions the data base horizontally to N
partitions and assigns them to the different sites.

Phase 1:// in parallel each site generates the frequent local
itemsets
For each site I= 1 ..... P do in parallel
L(i) = gen-large-itemsets(p_i) // Find the set of frequent local
itemsets;
// Each site send L(i) to coordinator to calculate the set of
global candidates ;
End of parallel

Phase 2: // the coordinator calculates the set of global
candidate itemsets
Coordinator : C^G = ∪_{i=1..p} L(i);
// Diffuse C^G to all processors;

Phase 3: // count the supports of global candidates
For each site I= 1 ..... P do in parallel
    For each  c ∈ C^G do
              c.compter(i) = gen-count (c, p_i);
              // Send the results to coordinator;
              End
End of parallel
Phase 4 : //the coordinator calculate the global supports of
global candidates
Coordinator : For each  c ∈ C^G do
    c.compter = Σ_{i=1}^{p} c.compter^(i)   //combine accounts;
    Return L^G = {c ∈ C^G | Σ_{i=1}^{p} c.compter^(i) ≥ min_sup
};
End
```

Fig. 6: Algorithm illustrated role of the coordinator and the different sites

The database is partitioned to N sites (clients) in an equitable way by dividing the number of transactions on the number of sites, i.e. each site will have a transaction set (partition), where each one is responsible to identify a set of frequent itemsets in order to discover the interesting rules. The following algorithm in figure6 shows the role of the coordinator and the different sites in each algorithm phase.

EXPERIMENTS

To study the performance of the parallel Partition algorithm, we realized several experiments by varying the topology configuration parameters (sites number), the minimal threshold and the number of transactions. The database used contains 12 items and 50000 transactions.

Sequential algorithm experiments

From test results, we can note that the more the value of minimal support decreases, the more the execution time increases. This is explained by an important number of candidates generated after each iteration. We observe also that the more the transaction number increases, the more the execution time increases. For the number of partitions, the execution time diminishes as the number of partitions grows bigger.

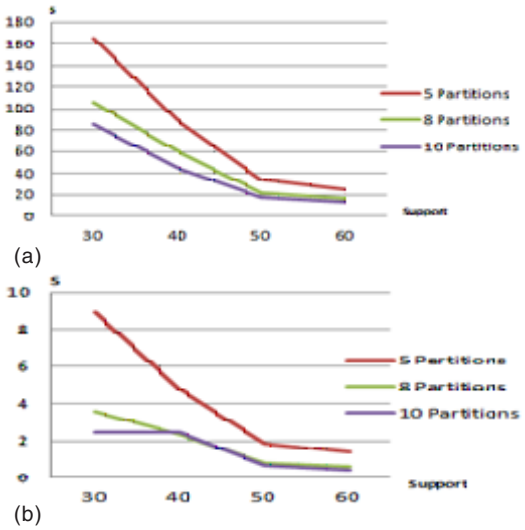

Fig. 7: Parallel algorithm of Partition on: (a) 50000 transactions (b) 25000 transactions

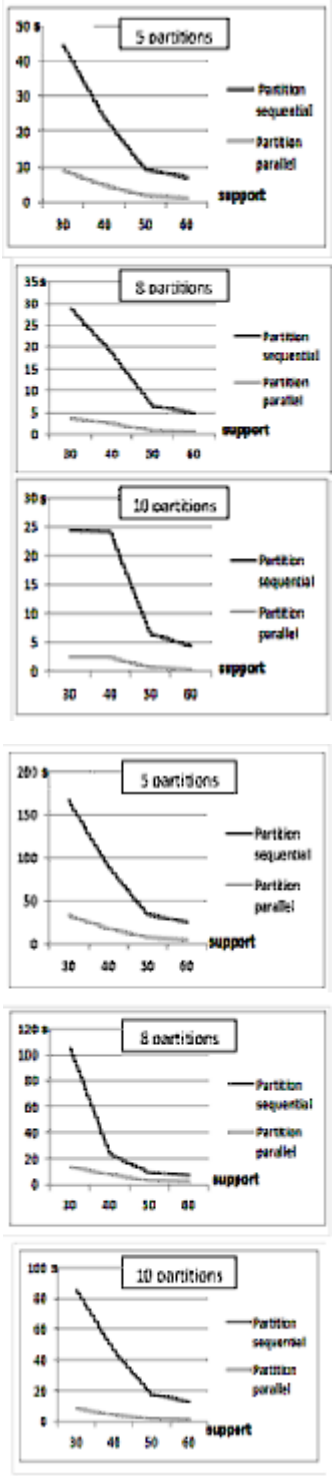

Fig. 8: Sequential and parallel Partition on: (a) 5,8,10 partitions respectively, 25000 transactions (b) 5,8,10 partitions respectively, 50000 transactions

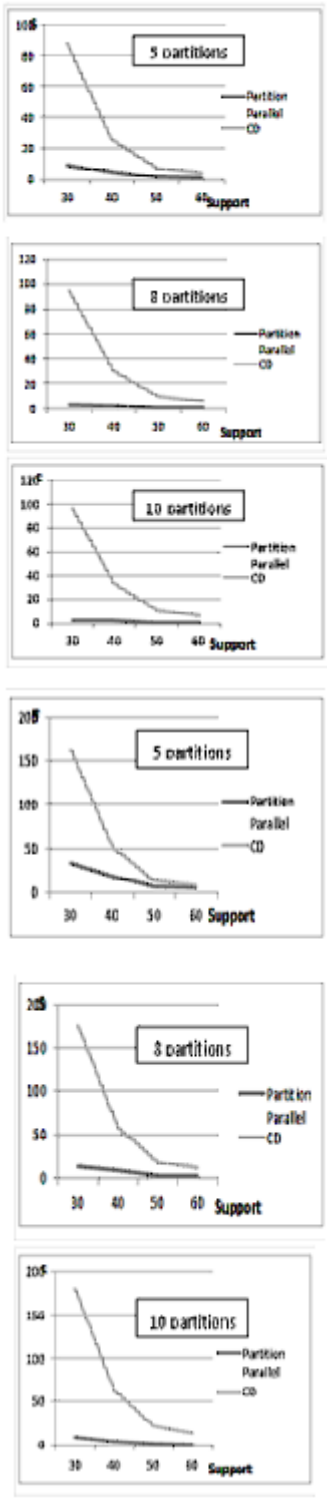

Fig. 9: CD and parallel Partition on: (a) 5,8,10 partitions respectively, 25000 transactions (b) 5, 8, 10 partitions respectively, 50000 transactions

Parallel algorithm experiments

From implementation of our parallel algorithm, we note a significant gain of execution time, particularly for a high number of partitions, a large database and a minimal value of support. (see figure7)

Comparative study with the sequential algorithm

We observe from the comparative results showed in figure 8, a remarkable reduction in execution time comparing with sequential version of Partition algorithm. Moreover, execution time decreases considerably by increasing the number of sites (partitions). We also notice a great time saving especially for an important number of transactions, this implies a minimal cost of communication between sites. This experimentation shows also a moderate increase in the execution time by increasing the minimal support value and the transaction number.

Comparative study with the Count Distribution algorithm

From the comparative results with the Count Distribution algorithm, we observe in figure 9, a satisfactory reduction of the execution time particularly for a significant number of nodes and transactions as well as a minimal value of support. This is due to the exchanges of important data (candidates) between nodes in CD algorithm during each pass, which is not the case for our parallel Partition algorithm.

CONCLUSION

Recently several parallel algorithms for the extraction of association rules appeared, after a study of some of these algorithms, we noticed that most of these algorithms usually require a high number of data scans and multi-phase of synchronizations and communication which degrade their performances. Our objective was to conceive a distributed algorithm for extracting frequent itemsets considering these criteria. After a detailed study of Partition algorithm which requires only two scans of the database, we proposed our contribution where we are interested in reducing the execution time by the parallelization of Partition algorithm in distributed and parallel environments. Thus, we used a centralized approach to reduce the communication cost.

The parallel Partition algorithm presented in this paper was implemented on different dimensions, database sizes, support values and the number of sites (partitions). The use of a super-coordinator in communication permitted to reduce the number of messages exchanged between sites, because a message sending is done directly via a super-coordinator. Moreover, our parallel algorithm has only two phases which implies communication and the size of the messages sent during the two phases is very small. This is because this algorithm exchanges only accounts for locally frequent itemsets, of which the number represents one fraction of the candidates' number. These minimal costs of communication and data scanning permitted to reduce execution time and to amplify the algorithm's performance. In comparison with the count distribution algorithm, we can note that the Parallel Partition algorithm represents a remarkable time-saving for large databases, where the number of sites is big and the support values are small. In perspective, this algorithm can be improved in two axes:

- A single scan of the database. In other words, the algorithm uses in phase II the calculations of supports solved in phase I without the obligation of a second scan.
- An intelligent partitioning of the database by applying a clustering algorithm on the entire database.

REFERENCES

1. Mehta, M., Agrawal, R. and Rissanen, J. : A fast scalable classifier for data mining. *Lecture Notes in Computer Science*, **1057**: 18-32 (1996).
2. Shafer, J., Agrawal, R. and Mehta, M.: A scalable parallel classifier for data mining. In Proceedings of VLDB'96 the 22th International Conference on Very Large Data Bases, 544–555 (1996).
3. Kufrin, R.: Decision trees on parallel processors. *In Proceedings of Parallel Processing for Artificial Intelligence* **3**(20): 279-306, (1997).
4. Joshi, M., Karypis, G. and Kumar, V.: A new scalable and efficient parallel classiûcation algorithm for mining large datasets. In Proceedings of IPPS the 11th International Parallel Processing Symposium, 7-7, IEEE Computer Society Press, (1998).
5. Satuluri, V.: A survey of parallel algorithms for classification. March (2007).
6. Zaki, M. J., Ho, C.T. and Agrawal. R.: Parallel Classification for Data Mining on Shared-Memory Multiprocessors. In Proceedings of ICDE '99 the 15th International Conference on Data Engineering , 198-198, IEEE Computer Society Washington, (1999).
7. Srivastava, A., Han, E. H., Kumar, V. and Singh. V.: Parallel Formulations of Decisiontree Classification Algorithms. *Journal of Data Mining and Knowledge Discovery*, **3**(3): (1999).
8. Farnstrom, F., Lewis, J. and Elkan, C.: Scalability for Clustering Algorithms Revisisted. In Proceedings Of ACM SIGKDD (SIGKDD Exploration : Special Interest Group on Knowledge Discovery in Data), 2(2): 1-7, juillet (2000).
9. Fasulo, D.: An analysis on recent work on clustering Algorithms, (1999).
10. Forman, G. and Zhang, B.: Distributed data clustering can be efficient and exact. In proceedings of SIGKDD Exploration, Vol2, (2000).
11. Johnson, E. L. and Kargupta, H.: Collective, hierarchical clustering from distributed, heterogenous data. Lecture Notes in Computer Science, 221-244, Springer-Verlag, (1999).
12. Nagiza F. Samatova, G. Ostrouchov, A. Geist, and Anatoli V. Melenchko, R.: An efficient covering based merging of clustering hierarchies from distributed datasets. *In International Journal of Distributed and Parallel Databases,* **11**(2): 157-180, (2002).
13. Savasere, A. Navathe, E.: An Efficient Algorithm for Mining Association Rules in Large Databases. In proceedings of VLDB '95 the 21th International Conference on Very Large Data Bases, September (1995).
14. Agrawal, R., Imielinski, T. and Swami, A.: Mining Association Rules between Sets of

Items in Large Database. In proceedings of SIGMOD '93 the international conference on Management of data, **22**(2): (1993).

15. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Database. In proceedings of VLDB'94 the 20th international conference on very large database, 487-499, (1994).
16. Han J. and Fu, Y.: Discovery of Multiple-Level Association Rules from Large Database. In proceedings of VLDB '95 the 21th International Conference on Very Large Data Bases, (1995).
17. Park, J. S., Chen, M. S. and Yu, P. S.: An effective Hash Based Algorithm for Mining Association Rules . In proceedings of SIGMOD '95 the international conference on Mang of data, Vol 24 No 2, May (1995).
18. Toivonen, H.: Sampling Large Databases for Association Rules. In proceedings of VLDB '96 the 22th International Conference on Very Large Data Bases, (1996).
19. Brin, S., Motwani, R., Ullman, J.D. and Tsur, S.: Dynamic Itemset Counting and Implication Rules for Market Basket Data. In proceedings of SIGMOD '97 the international conference on Management of data, **26**(2): (1997).
20. Agrawal, R. and John C. S.: Parallel Mining of Association Rules. Knowledge and Data Engineering, IEEE Transactions, **8**: 962 – 969 (1996).
21. Cheung, D. W., Ng, V. T., Fu, A. W. and Fu. Y.: Efficient Mining of Association Rules in Distributed Databases. Knowledge and Data Engineering, IEEE Transactions, **8**: 911 -922 (1996).
22. Cheung, D. W. and AL.: A Fast distributed Algorithm for Mining Association Rules. In Proceedings of the fourth international conference on Parallel and Distributed Information Systems, 31-42 (1996).
23. Ashrafi, M. Z. , Taniar, D. and Smith, K. A.: An Optimized Distributed Association Rules Mining Algorithm. Distributed Systems Online, IEEE, 5 (2004).
24. Shintani, T. and Masaru, K.: Hash Based Parallel Algorithms for Mining Association Rules. In proceedings of the fourth international conference on Parallel and Distributed Information Systems, 19-30 (1996).
25. ZAKI, M. J., Parthasarathy, S. and Li., W.: A Localized Algorithm for Parallel Association Mining. In proceedings of the SPAA '97 ACM symposium on Parallel algorithms and architectures, (1997).
26. Han, E. H., Karypis, G. and Kumar, V.: Scalable Parallel Data Mining for Association Rules. Knowledge and Data Engineering, IEEE Transactions, Vol 12, 337 – 352, (2000).