Un algorithme parallèle pour l'extraction des règles d'association dans les grilles de calcul

Khadidja Belbachir

Laboratoire LAAR, Département d'informatique
Université des Sciences et de la Technologie d'Oran USTO-MB
ORAN

Belbachir_khadidja@yahoo.fr

Hafida Belbachir

Laboratoire LSSD, Département d'informatique
Université des Sciences et de la Technologie d'Oran USTO-MB
ORAN

h_belbach@yahoo.fr

RÉSUMÉ. Suite à l'expansion des supports physiques de stockage et les besoins incessant de sauvegarder de plus en plus des données, les algorithmes séquentiels de recherche des règles d'associations se sont avérés inefficace. Ainsi l'introduction des nouvelles versions parallèles est devenue impérative. Nous proposons dans cet article, une version parallèle d'un algorithme séquentiel Partition. Ce dernier est fondamentalement différent des autres algorithmes séquentiels, car il scanne la base de données deux fois seulement pour générer toutes les règles d'association significatives. Par conséquence, l'approche parallèle ne demande pas beaucoup de communication entre les sites. L'approche proposée a été implémentée pour une étude expérimentale. Les résultats obtenus, montre un grand gain en temps d'exécution par rapport à la version séquentielle.

ABSTRACT. Subsequently expansion of storage physical supports and needs ceaseless to save more data, the sequential algorithms of research association rules proved to be ineffective. Therefore introduction of new parallel versions became imperative. We propose in this paper, a parallel version of a sequential algorithm Partition. This version is fundamentally different from other sequential algorithms, because it scans data base twice only to generate significant association rules. By consequence, the parallel approach does not require much communication between sites. The proposed approach was implemented for an experimental study. The obtained results, shows a great saving of execution time compared to the sequential version.

MOTS-CLÉS : d'association, Data mining distribué, Partition, Algorithme parallèle.

KEYWORDS: Association rules; Distributed data mining; Partition; Parallel algorithms.

1. Introduction

Depuis quelques années, on assiste à une forte augmentation tant dans le nombre que dans le volume des informations mémorisées par des bases de données scientifiques, économiques, financières, administratives, médicales…etc. Ces méga bases de données sont peu exploitées, alors qu'elles cachent de connaissances décisives face au monde extérieur. Pour combler ce besoin, une nouvelle industrie est née le Data Mining. Le datamining distribué (DDM : Distributed Datamining) est le processus du datamining classique qui consiste à extraire une nouvelle connaissance à partir de sources de données partitionnées, distribuées sur différentes sites, chaque site applique un algorithme du Datamining sur les données locales .les résultats sont ensuite combinés avec le minimum d'interaction entre les sites de données. La plupart des algorithmes de DDM sont désignés sur le parallélisme potentiel qu'ils peuvent appliquer sur les données distribués fournies. Généralement le même algorithme fonctionne sur chaque site de données distribuées en même temps, produisant un modèle local par site, par la suite tous les modèles locaux sont regroupés pour produire le modèle final. Essentiellement, le succès des algorithmes DDM réside dans le transfert de données minimal. Parmi les algorithmes du Data Mining distribué on cite : la classification distribuée [2] [11] [14] [15] [21], le clustering [6] [7] [8] [17] distribué et les règles d'association distribuées.

Dans ce papier, on présente un nouvel algorithme parallèle qui diffère des autres algorithmes, on proposant une parallélisation d'un algorithme séquentiel appelé Partition [1]. Ce dernier nécessite que deux scannes de la base de données, ce qui permettra de réduire le nombre de communications ainsi que les étapes de synchronisation entre les sites dans la version parallèle de cet algorithme. Nous avons choisis une architecture centralisée pour minimiser le cout de communication, dont chaque site traite sa base de données indépendamment pendant les deux scannes et envoie les résultats au coordinateur qui se chargera de cumuler les résultats.

Le reste du papier est organisé comme suit, la section suivante est une introduction à l'extraction des règles associatives. Dans la section 3, on présente l'algorithme parallèle proposé en montrant ses différentes étapes. Les résultats des expérimentations de l'algorithme parallèle sont présentés dans la section 5 avec les commentaires. Nous conclurons avec des perspectives dans la section 6.

2. L'extraction des règles d'association

La technique d'extraction des règles d'association est une méthode d'apprentissage non supervisé, permettent de découvrir à partir d'un ensemble de transactions, un ensemble de règles qui expriment une possibilité d'association entre différents attributs. Prenons l'exemple d'un supermarché où les articles achetés par chaque consommateur sont enregistrés en même temps dans une base de données comme une transaction .A partir de cet exemple, on peut trouver une règle associative de la forme « 90% des

clients qui achètent du chocolat et du lait ont tendance à acheter du fromage ». Chocolat et lait constituent l'antécédent de cette règle, fromage est la conséquence et 90% est la confiance. Les règles associatives ont été utilisées avec succès dans des domaines comme l'aide à la planification, l'aide au diagnostic en recherche médical, l'amélioration des processus de télécommunication, l'organisation et l'accès aux sites Internet, l'analyse d'images et de données spatiales, statistiques et géographiques, etc.

2.1. Les algorithmes séquentiels

Les premiers algorithmes d'extraction des itemsets fréquents, AIS [19], Apriori [20] et SETM [10] qui ont été proposés en 1993. D'autres algorithmes permettant de réduire le temps d'extraction des itemsets fréquents sont ensuite apparus, ainsi que plusieurs optimisations et structures de données permettant d'améliorer l'efficacité de l'algorithme apriori, on peut citer :AprioriTid [20], AprioriHybrid [20], DHP (Direct Hashing and Pruning) [12], Partition [1], Sampling [9], DIC (Dynamic Itemsets Counting) [22].

2.2 Les algorithmes parallèles

Suite à l'expansion des supports physiques de stockage et les besoins incessant de sauvegarder de plus en plus des données, les algorithmes séquentiels de recherche des règles d'associations se sont avérés inefficace. Ainsi l'introduction des nouvelles versions parallèles est devenue impérative. Les algorithmes parallèles et distribués actuels sont basés sur l'algorithme séquentiel Apriori. Deux paradigmes de parallélisme sont employés, Le parallélisme de données et le parallélisme de taches.

Parmi les algorithmes du Parallélisme de données, on peut citer, CD (Count Distribution) [18], FDM (Fast Distributed Mining) [3], DMA (Distributed Mining of Association Rules)[4] et ODAM (Optimized Distributed Association Mining)[16]. Pareillement pour le parallélisme de tâches, de nombreux algorithmes ont été proposés, tel que, DD (Data Distribution)[18], IDD (Intelligent Data Distribution)[5] et HPA (Hash Partioned Apriori)[23]. Il existe d'autres algorithmes qui ne peuvent pas être classifiés dans les deux familles, comme HD (Hybrid Distribution)[5], CAD (Condidate Distribution) et ECLAt[13].

3. L'algorithme Partition parallèle

De nombreux algorithmes parallèles et distribués ont été proposés pour l'extraction distribuée des règles d'associations. L'objectif de ceux-ci est de réduire le coût de communication qui constitue un facteur important pour mesurer leurs performances. Cependant, la plupart de ces algorithmes présentent habituellement un nombre élevé de balayage des données et des étapes multiples des synchronisations et de communication qui dégrade leurs performances. Afin de résoudre ces inconvénients nous avons choisi l'algorithme séquentiel Partition [1], on s'intéresse à la parallélisation de cet algorithme. Ce dernier, ne nécessite que deux balayages de la base de données. Ce qui permet de réduire le nombre de communication dans la version parallèle de cet algorithme.

3.1. L'algorithme Partition

Proposé par A.Savasere, E.Omiecinski et S.Navathe en 1995, l'approche adaptée par cet algorithme est de diviser la base de données en partitions horizontales de même taille. Il est exécuté sur chaque sous-ensemble des transactions (Partition) de façon indépendante, produisant dans un 1er balayage un ensemble d'itemsets locaux fréquents pour chaque partition. L'ensemble des candidats globaux est formé comme étant l'union de tous les ensembles d'itemsets locaux fréquents. Pour obtenir le support global pour ces itemsets, un 2eme balayage de la base de données est nécessaire.

```
1)   P= partition_base de donnée(D)
2)   N= nombre_de_partitions
3)   for i =1 to n begin // Phase I
4)      lire-dans-partition (pi ∈ P )
5)      Li = gen-large-itemsets(pi)
6)   end
7)   for (i=2; L_i^j ≠ ∅ , j =1,2,…,n; i++) do
8)      C_i^G = ∪j=1,2,…,n L_i^j  // phase de fusion
9)   for i=1 to n begin // Phase II
10)     lire-dans-partition (pi ∈ P )
11)     Pour tous candidats c ∈ C_i^G gen-count ( c , pi)
12)  end
13)  L^G = { c ∈ C_i^G | c.count ≥ minSup }
```

Figure 1. ***L'algorithme Partition***

3.1.1. Phase I

Prend n itérations, pendant l'itération i, seule la partition pi est considérée. La fonction gen_large_itemsets illustrée dans la figure.2 prend la partition pi comme entrée et génère comme sortie les itemsets locaux fréquents de toutes les longueurs (L_2^i,L_3^i,..L_l^i).

```
Procedure gen-large-itemsets (p : partition_base_de_données)
1)   L_1^p = { 1-itemsets frequents avec leurs tidlists }
2)   for ( k=2 ; L_k^p ≠ ∅ ; k++ ) do begin
3)   for all itemsets l1 ∈ L_{k-1}^p do begin
4)   for all itemsets l2 ∈ L_{k-1}^p do begin
5)   if l1[1] = l2[1] ∧ l1[2] = l2[2] ∧ … ∧ l1[k-1] < l2[k-1] then
6)   c = l1 [1]. l1[2] …l1[k-1] . l2 [k-1]
7)   if c ne peut pas être élagué then
8)   c.tidlist= l1.tidlist ∩ l2.tidlist
9)   if | c.tidlist| / |p| ≥ minSup then
10)  L_k^p = L_k^p ∪ {c}
11)  end
12)  end
13)  end
14)  return ∪k L_k^p
```

Figure2. ***La fonction gen_large_itemsets***

3.1.2. La phase II

La procédure gen_final_count donnée dans la figure.4 met en place un compteur pour chaque itemset candidat global, compte son support dans la base de données et génère les itemsets globaux fréquents.

3.2. L'algorithme parallèle de Partition

Nous décrivons dans cette section l'approche parallèle de l'algorithme partition qui peut être efficacement appliquée pour l'extraction des règles d'association dans un environnement distribué.

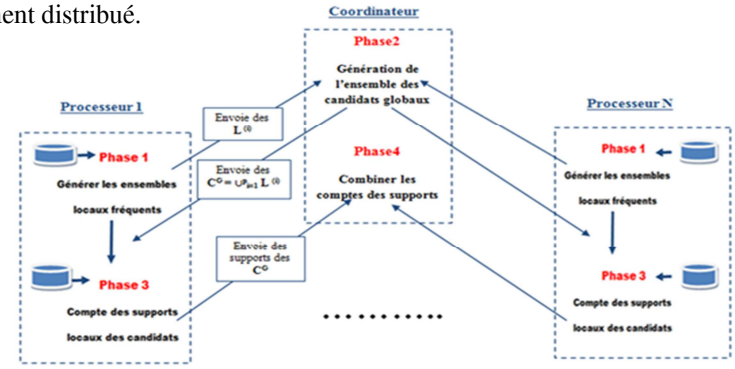


Figure 3. *Schéma de l'algorithme parallèle de Partition*

L'algorithme parallèle comme le montre la figure.3 s'exécute en quatre phases. Où la topologie adaptée est vue comme un ensemble de n sites (clients) gérés par un site appelé coordinateur (serveur).

3.2.1. L'algorithme

La base de données est partitionnée sur les N sites (clients) de manière équitable en divisant le nombre de transactions sur le nombre de sites, c.-à-d. chaque site disposera d'un ensemble de transaction (partition), où chacun va se charger d'identifier l'ensemble d'itemsets fréquent afin de découvrir les règles intéressantes. La figure.4 illustre le rôle du coordinateur et les différents sites dans chaque phase de l'algorithme.

Phase 1: chaque processeur dispose d'une partition, il applique la procédure gen_large_itemsets sur les données locales pour calculer les supports des itemsets locaux afin d'identifier l'ensemble des itemsets fréquents localement Li, ensuite il envoie le résultat au coordinateur. Comme la taille des données locales dans tous les nœuds de traitement est approximativement égale, cette phase prendra du temps approximativement égal dans tous les nœuds en réalisant un équilibrage de charge. En plus, seulement les données disponibles localement sont traitées. Par conséquent, il n'ya aucun coût de communication pour le transfert de données entre les nœuds.

Phase 2: Après la phase 1, le coordinateur va se charger d'identifier l'ensemble des candidats globaux CG, en faisons l'union de tous les ensembles d'itemsets frequent localement de chaque processeurs (traitement realisé dans la phase fusion de l'algorithme Partition). Le résultat est diffusé à tous les processeurs. Après la fin de cette phase, tous les nœuds auront exactement le même ensemble d'itemsets candidats.

Phase 3: correspond à la phase II de l'algorithme séquentiel, où chaque processeur doit déterminer le support pour tous les itemsets dans CG , ensuite les envoyer aux coordinateur.

Phase 4: après avoir reçue le résultat de la phase 3, le coordinateur va se charger d'identifier l'ensemble d'itemsets fréquents globaux en faisons la sommation des supports de chaque itemset.

Entrée : Base de données D, le nombre de sites P, le support minimal minsup
Sortie : l'ensemble des itemsets fréquents L^G.
Coordinateur : partitionne la base de données horizontalement en N partitions et les affecte aux différents sites.

Phase 1:// En parallèle chaque site génère les itemsets locaux fréquents
Pour chaque site I= 1 …… P **faire en parallèle**
$L^{(i)}$ = **gen-large-itemsets** (p_i) // trouver l'ensemble d'itemsets fréquents locaux ;
// Chaque site envoie les $L^{(i)}$ au coordinateur pour calculer l'ensemble de candidats globaux ;
Fin de parallèle
Phase 2 : //Le coordinateur calcule l'ensemble d'itemsets candidats globaux
Coordinateur: $C^G = \bigcup_{i=1..P} L^{(i)}$;
//Diffuser C^G à touts les processeurs ;
Phase 3 : //Compte des supports des candidats globaux
Pour chaque site I= 1 …… P **faire en parallèle**
Pour chaque c∈ C^G **faire**
$c.compter^{(i)}$ = gen-count (c , p_i) ;
// Envoie des résultats au coordinateur ;
Fin
Fin de parallèle
Phase 4 : //Le coordinateur calcul les supports globaux des candidats globaux
Coordinateur : **Pour chaque** c∈ C^G **faire**
c.compter = $\sum_{i=1}^{p} c.compter^{(i)}$ //combiner les comptes;
Return L^G= c∈ C^G | $\sum_{i=1}^{p} c.compter^{(i)} \geq \min\_sup$;
Fin

Figure 4. *L'algorithme parallèle de Partition*

4. Expérimentations

Pour étudier les performances de l'algorithme parallèle de partition, nous avons réalisé plusieurs expérimentations sur un simulateur (GridSim) en variant les paramètres de configuration de la topologie (nombre de sites), le seuil minimal et le nombre de transactions. La base de données utilisée contient 12 items et 50000 transactions.

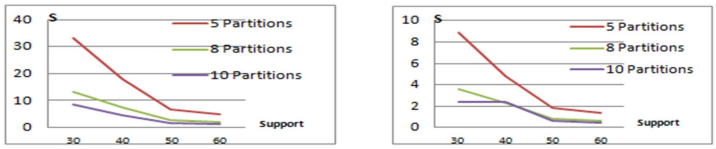
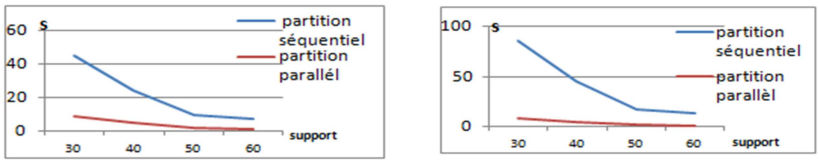


Figure 5. *L'algorithme Partition parallèle*
(a)sur 5000 transactions, (b) sur 2500 transactions

A partir des résultats présentés ci-dessus, nous constatons que plus le nombre de sites n'augmente, plus le temps d'exécution diminue. En remarquant un grand gain de temps surtout pour un nombre important de transactions, ce qui implique un cout minimal de communication entre les sites. Cette expérimentation montre aussi un accroissement modéré du temps d'exécution en augmentant la valeur du seuil minimal de support. Comme le montrent les graphes présentés dans ce qui suit, l'implémentation de notre algorithme parallèle a permis d'obtenir des résultats satisfaisants, puisque nous avons pu réduire le temps d'exécution en comparant avec la version séquentielle de l'algorithme Partition. Où on constate un gain important du temps d'exécution,

particulièrement pour un nombre élevé de partitions, une base de données large et une valeur minimale de support.



(a) (b)

Figure 6. *L'algorithme Partition séquentiel et parallèle*

(a)sur 5 partitions, 25000 transactions, (b) sur 10partitions, 50000 transactions

5. Conclusion et perspectives

Notre objectif était de concevoir un algorithme distribué pour l'extraction des itemsets fréquents prenant en compte ces critères. Après une étude détaillée de l'algorithme Partition qui ne nécessite que deux scans de la base de données, nous avons proposé notre contribution où on s'intéresse sur la réduction du temps d'exécution par la parallélisation de l'algorithme Partition dans un environnement distribué et parallèle. Ainsi, nous avons utilisé une approche centralisée afin de réduire le coût de communication. L'algorithme Partition parallèle présenté dans ce papier a été implémenté sur différentes dimensions, la taille de la base de données, la valeur du support et le nombre de sites (partitions). L'utilisation de super-coordinateur dans la communication a permet de réduire le nombre de messages échangés entre les sites, puisque l'envoi des messages se fait directement avec le super-coordinateur. En plus, notre algorithme parallèle a seulement deux phases qui impliquent la communication, ainsi que la taille des messages envoyés pendant les deux phases est petite, la raison est que cet algorithme échange seulement les comptes pour les ensembles localement fréquents, dont le nombre ne représente qu'une fraction du nombre de candidats. Ces couts minimaux de communication et de balayage des données ont permet de réduire le temps d'exécution et amplifier les performances de l'algorithme. Ce gain de temps est remarquable plus pour les bases de données volumineuses dont le nombre de sites est élevé et le support est plus petit. Comme perspectives, cet algorithme peut être amélioré dans les deux axes:

- Un seul scan de la base de données. Autrement dit, l'algorithme utilise dans la phase II les calculs des supports résolu dans la phase I sans obligation d'un deuxième scan.
- Un partitionnement intelligent de la base, en appliquant un clustering sur la base de données entière.

6. Bibliographie

[1] A. Savasere, E. Omiecinski, and S. Navathe. "An Efficient Algorithm for Mining Association Rules in Large Databases". In proc of 21th Int Conf on VLDB, Sept 1995.

[2] A. Srivastava, E. H. Han, V. Kumar, and V. Singh. "Parallel Formulations of Decisiontree Classification Algorithms". J. Data Mining and Knowledge Discovery, Vol3 N3, Sept 1999.

[3] David W. Cheung, and AL. “A Fast distributed Algorithm for Mining Association Rules”. In Proc of 4th int conf on Parallel and Distributed Information Systems, pp 31–42, Dec 1996.

[4] David W. Cheung, Vincent T. Ng, Ada W. Fu, and Y. Fu. “Efficient Mining of Association Rules in Distributed Databases”. Knowledge and Data Engineering, IEEE Trans, Vol. 8, pp 911-922, Dec 1996.

[5] E. H. Han , G. Karypis, and V. Kumar. “Scalable Parallel Data Mining for Association Rules”. Knowledge and Data Engineering, IEEE Transactions, Vol. 12, pp 337–352, 2000.

[6] E. L. Johnson and H. Kargupta. “Collective, hierarchical clustering from distributed, heterogenous data”. Lecture Notes in Comp Sc, 1759, pp 221-244, Springer-Verlag, 1999.

[7] F. Farnstrom, J. Lewis, and C. Elkan. “Scalability for Clustering Algorithms Revisisted”. In Proceedings Of ACM SIGKDD, 2:(2) :1-7, juillet 2000.

[8] G. Forman and B. Zhang. “Distributed data clustering can be efficient and exact”. In proceedings of SIGKDD Exploration, Vol 2, 2000.

[9] H. Toivonen. “Sampling Large Databases for Association Rules”. In proc of the 22th Int Conf on VLDB, 1996.

[10] J. Han and Y. Fu. “Discovery of Multiple-Level Association Rules from Large Database”. In proceedings of 21th Int Conf on Very Large Data Bases, September 1995.

[11] J. Shafer, R. Agrawal, and M. Mehta. “SPRINT: A scalable parallel classifier for data mining”. In Proceedings of VLDB '96 the 22th Int Conf on VLDB, pp 544–555, 1996.

[12] J. S. Park, M. S. Chen, and P. S. Yu. “An effective Hash Based Algorithm for Mining Association Rules” . In proceedings of SIGMOD '95 the int conf on Management of data, Vol 24 N 2, May 1995.

[13] M. J. ZAKI, S. Parthasarathy, and W. Li. “A Localized Algorithm for Parallel Association Mining”. In proceedings of the SPAA'97 ACM symposium on Parallel algorithms and architectures, 1997.

[14] M. J. Zaki, C.T. Ho, and R. Agrawal. “Parallel Classification for Data Mining on Shared-Memory Multiprocessors”. In Proceedings of ICDE ‘99 the 15th Int Conf on Data Engineering , page 198, IEEE Computer Society Washington, 1999.

[15] M. Mehta, R. Agrawal, and J. Rissanen. “SLIQ: A fast scalable classifier for data mining”. Lecture Notes in Computer Science, Vol 1057/1996, pp. 18-32, 1996.

[16] M. Z. Ashrafi, D. Taniar, and K. A. Smith . “ODAM : an Optimized Distributed Association Rules Mining Algorithm”. Distributed Systems Online, IEEE, Vol 5, 2004.

[17] Nagiza F. Samatova, G. Ostrouchov, A. Geist, and Anatoli V. Melenchko. “Rachet : An efficient covering based merging of clustering hierarchies from distributed datasets”. In Int. J. of Distributed and Parallel Databases, 11(2), pp 157-180, 2002.

[18] R. Agrawal and John C. Shafer. “Parallel Mining of Association Rules. Knowledge and Data Engineering”, IEEE Transactions, Vol. 8, pp 962 – 969, Dec 1996.

[19] R. Agrawal, T. Imielinski, and A. Swami. “Mining Association Rules between Sets of Items in Large Database”. In proc of int conf on Management of data, Vol22 N2, June, 1993.

[20] R. Agrawal, R. Srikant. “ Fast Algorithms for Mining Association Rules in Large Database”. In proceedings of VLDB, pp487-499, Sept 1994.

[21] R. Kufrin. “Decision trees on parallel processors”. In Proc of Parallel Processing for Artificial Intelligence, Vol. 20, pp 279-306, 1997.

[22] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. “Dynamic Itemset Counting and Implication Rules for Market Basket Data”. In proc of int conf on Management of data, Vol26 N2, June 1997.

[23] T. Shintani and K. Masaru. “Hash Based Parallel Algorithms for Mining Association Rules”. In proc of the 4th int conf on Parallel and Distributed Inf Sys, pp19–0, Dec 1996.