Approche de parallélisation

de l'algorithme Apriori

K. Arour* — Y. Slimani** — A. Assas** — M. Jemni***

* Département Génie Informatique et Mathématiques,
Institut National des Sciences Appliquées et de Thechnologie, INSAT, Tunis, Tunisie
mohamed.arour@planet.tn.
** Département des Sciences Informatique. Faculté des Sciences de Tunis 1060, Tunis, Tunisie.
yahya.slimani@fst.rnu.tn ;assas_anis@yahoo.fr
*** Ecole Supérieure des Sciences et Techniques de Tunis, ESSTT, Tunis, Tunisie.
mohamed.jemni@fst.rnu.tn

RÉSUMÉ. Comme tous les autres domaines de l'informatique, les systèmes d'informations modernes ont bénéficié des avancées technologiques des dernières décennies. Ceci a permis d'avoir des systèmes d'information répartis et mobiles. D'un point de vue informationnel, ces systèmes renferment des données implicites qu'il faudra extraire et exploiter, en utilisant des techniques de datamining. Ces techniques utilisent des algorithmes d'extraction qui se caractérisent par leur degré de complexité (exponentielle), ce qui les rend inefficaces du point de vue performance. Un des moyens pour améliorer cette performance consiste à utiliser le parallélisme. Ainsi, nous allons, dans une première partie de cet article, faire une rétrospective sur les algorithmes séquentiels et parallèles de datamining. Puis, dans une deuxième partie, nous présenterons l'approche de parallélisation de l'algorithme Apriori d'Agrawal, qui est l'algorithme de base de tout processus de datamining.

ABSTRACT. Like all the other fields of data processing, the modern information systems have integrared the results of the advanced technologies of the last decades. This made it possible to have information systems that are distributed and mobile. These systems contain implicit data which it will be necessary to extract and exploit, by using datamining techniques. These techniques use algorithms of extraction which are characterized by their high degree of complexity (exponential), which makes their performance ineffective. One of the means to improve this performance consists in using parallelism. In this context, we propose a new parallel version of the Apriori algorithm of Agrawal, that is the main algorithm of each datamining technique.

MOTS-CLÉS : Fouille de données, Algorithme apriori, Itemsets fréquents, Algorithmes parallèles, Règles associatives

KEYWORDS : Datamining, Apriori algorithm, Parallel datamining algorithms, Association rules, Frequent itemsets

1. Introduction

Les systèmes d'information modernes se distinguent des systèmes d'information classiques par plusieurs aspects : *Localisation*, *Volume*, *Types d'informations*, *Accès* et *Exigences des utilisateurs*. Ainsi, les systèmes d'informations modernes posent des problématiques nouvelles pour gérer la masse d'informations qu'ils renferment. Or, il s'avère que cette masse d'informations, qui est en augmentation constante, renferme des données implicites qu'il faudra extraire et exploiter. Il s'agit donc d'utiliser des techniques d'analyse de données pour transformer ces données en connaissances, qui peuvent être exploitées par exemple, par une grande surface commerciale pour améliorer ses performances et répondre aux besoins de ses clients. Pour pallier aux insuffisances des techniques classiques d'analyse de données (basées essentiellement sur les statistiques), le datamining s'intègre dans un processus plus général d'extraction de connaissances (Knowledge Data Discovery -KDD). Il offre ainsi un certain nombre de techniques pour extraire des connaissances à partir d'ensembles de données volumineux. Parmi les techniques les plus populaires du datamining, nous pouvons citer la découverte de règles associatives [2]. En effet, l'information extraite (découverte) à partir d'ensembles volumineux de données, peut être exprimée sous forme d'un ensemble de règles associatives. Ces règles définissent, d'une part, des liens entre les données, et, d'autre part, peuvent servir à prédire le comportement d'autres données. L'extraction de ces algorithmes est une étape très couteuse, en ce sens que les algorithmes de datamining actuels se caractérisent par leur caractère hautement itératif et par le nombre important d'accès à une base de données. Ces facteurs font que les algorithmes de datamining ont une complexité exponentielle. L'analyse que nous avons faite sur les algorithmes de datamining existants à l'heure actuelle, nous a permis de relever qu'ils utilisaient tous un algorithme de base, à savoir l'algorithme Apriori d'Agrawal [1]. Une étude détaillée de cet algorithme a mis en évidence le lien qui existe entre cet algorithme et l'ordre de complexité des algorithmes de datamining. Pour cela, nous proposons dans cet article, une approche de parallélisation de l'algorithme Apriori. Le reste de l'article est structuré comme suit : dans la section suivante, nous allons faire un bref rappel sur les algorithmes séquentiels de datamining. Dans la section 3, nous ferons un survol des principales de parallélisation des algorithmes de datamining. La section 4 présente les détails de l'approche que nous avons défini et mise en oeuvre pour la parallélisation de l'algorithme Apriori. Il faut noter ici que cette approche constitue la première mise en oeuvre d'une version parallèle de l'algorithme Apriori. Dans la section 5, nous présenterons une série d'expérimentations sur les versions séquentielles et parallèles de l'algorithme Apriori. Enfin, la section 6 conclut cet article en présentant quelques orientations futures de recherche.

2. Algorithmes séquentiels de découverte de règles associatives

La découverte d'associations désigne la découverte de règles qui associent un ensemble d'attributs à un autre ensemble d'attributs. Une règle associative est de la forme $A \rightarrow B$, où A et B sont des conjonctions d'attributs de la base de données. A représente l'antécédent (ou la pré-

misse) de la règle et B la conclusion [2]. Par exemple, si on dispose d'une base de données contenant des informations sur la population d'un pays, la règle associative suivante peut être extraite : $(Age > 25) \wedge$ (Revenu> 5000) $\rightarrow$ (modèle_voiture=tourisme). Cette règle associe l'âge et le revenu d'un individu au type de la voiture qu'il peut posséder. L'interaction d'un processus de découverte d'associations avec un système guidé par l'utilisateur, devient intéressante dans le contexte du datamining. En effet, l'utilisateur peut intervenir pour définir un système de filtrage afin d'écarter les associations non évidentes ou ayant peu d'intérêt. A cet effet, les paramètres les plus couramment utilisés, dans un système de filtrage, sont le *support* (valeur qui permet de mesurer la fréquence de l'association) et la *confiance* (valeur permettant de mesurer la force de l'association, qu'on appelle parfois confidence) d'une règle [1]. Le problème de découverte des règles associatives se décompose en deux sous-problèmes [2] : (i) Trouver tous les ensembles d'articles possédant un support dépassant le seuil minimal *(minsup)* appelés *ensembles d'articles (ou itemsets) fréquents*. Ce sous problème constitue la partie fondamentale et coûteuse de la découverte des règles associatives. Plusieurs algorithmes ont été proposés dans la littérature pour trouver ce type d'ensemble [3, 2, 1] ; (ii) Utiliser les ensembles d'articles fréquents pour générer les règles désirées. Pour chaque ensemble d'articles fréquents L et pour chaque sous-ensemble non vide a de L, on génère une règle de la forme : $a \Rightarrow (L - a)$, si $support(L)/support(a) \geq minconf$.

Nous présentons de manière très succinte, dans ce qui suit, le principal algorithme de découverte des ensembles fréquents, à savoir l'algorithme Apriori d'Agrawal. Agrawal et al. [1] ont proposé des algorithmes pour l'extraction des ensembles (ou itemsets) fréquents. Les idées de base de ces algorithmes sont contenues dans l'algorithme Apriori [1], dont le principe de fonctionnement est définit comme suit : il effectue plusieurs passes sur une base de données. Dans la première passe, il calcule le support des (*1-ensemble d'articles*) et ils détermine ceux qui sont fréquents, et qui sont dénommés par *1-ensemble d'articles fréquents*. A partir de ce premier résultat, chaque passe ultérieure utilisera la liste des ensembles d'articles trouvés, dans la passe précédente, pour générer les nouveaux ensembles d'articles fréquents, appelés *ensembles d'articles candidats*. En même temps que cette phase de génération, le calcul des supports actuels pour ces ensembles d'articles candidats est réalisé. A la fin d'une passe, on détermine quels sont les candidats qui sont actuellement fréquents et qui seront utilisés dans la passe suivante. Ce traitement se répète jusqu'à ce que l'on ne puisse plus avoir de nouveaux ensembles d'articles.

2.1. Problèmes et complexité

La découverte de règles associatives, selon l'approche séquentielle, pose un certain nombre de problèmes, notamment quand il s'agit de phases itératives. Parmi ces problèmes, nous pouvons citer : (i) Le volume des données manipulées sur disques (E/S fréquentes) ; (ii) Le volume d'informations extraites qui a un effet sur la pertinence des connaissances extraites (fiabilité des règles) ; (iii) Les limites au niveau des tailles des mémoires centrales. Ces limites augmentent le volume d'opérations d'entrée/sortie entre la mémoire centrale et le disque.

La conjugaison de ces différents problèmes fait que les algorithmes séquentiels de découverte de règles associatives ont une complexité exponentielle en terme du nombre de transactions et

d'items différents contenus dans une base. En effet, considérons les hypothèses suivantes : (a) n : Nombre de transactions dans une base D ; (b) m : Nombre d'items différents dans cette base. Nous obtenons une complexité de l'ordre de $O(nm2^m)$. Etant donné cet ordre de complexité, il est tout à fait naturel de recourir à d'autres approches qui permettent de la réduire, autant que possible. Une de ces approches consisterait à utiliser le potentiel de calcul offert par les machines parallèles et distribuées, d'autant plus que l'on peut construire une machine parallèle assez puissante avec des grappes de PC.

3. Algorithmes parallèles de découverte de règles associatives

Etant donné la complexité des algorithmes séquentiels de découverte de règles associatives, beaucoup de travaux se sont orientés vers la parallélisation des algorithmes séquentiels existants, pour exploiter au mieux la puissance offerte par les machines parallèles et distribuées [5]. La majeure partie, voire la totalité de ces travaux, utilisent une parallélisation basée sur la distribution des données, qui peuvent être soit la base de transactions ou l'arbre des candidats. Or les méthodes de partitionnement proposées souffrent du problème de choix du critère de partitionnement. En effet, le choix doit s'effectuer de telle sorte que les charges des processeurs soient équilibrées, ce qui n'est pas le cas de toutes les méthodes proposées. Pour le calcul du support, plusieurs approches ont été proposées, tel que le partitionnement de l'ensemble des candidats [4]. Une autre solution consiste à partitionner la base de données entre les processeurs [4]. Chaque processeur utilise la partition, qui lui est associée, et génère un ensemble d'ensembles d'articles larges local. A la fin, une phase de communication s'effectue entre les processeurs pour élaborer l'ensemble large de l'itération en cours. Dans ce cadre, plusieurs algorithmes, basés sur *Apriori*, ont été définis, à savoir [3, 4, 5] : CD (Count Distribution), DD (Data Distribution), PCCD (Common Candidate Partitionned Database), CCPD (Partitionned Candidate Common Database), HD (Hybrid Distribution). Une étude comparative de ces algorithmes a été effectuée par Slimani et al. dans [5].

4. Nouvelle approche de parallélisation d'Apriori

4.1. Introduction

L'approche de parallélisation de l'algorithme Apriori, que nous proposons dans cet article, est une approche qui combine à la fois la parallélisation des données et la parallélisation des tâches. Nous essayons d'exploiter tous les niveaux de parallélisme des différentes phases de l'algorithme Apriori. Il s'agit de décomposer l'algorithme Apriori en tâches, puis de chercher les relations de dépendance entre ces tâches et par la suite la programmation en parallèle des tâches indépendantes. Afin d'obtenir une programmation parallèle qui soit efficace, nous tenons également compte des spécificités architecturales de la machine parallèle qui nous sert de plateforme d'exécution (machine distribuée IBM SP2 de 32 processeurs) ainsi que de l'environnement de

programmation utilisé (C et MPI). L'objectif visé à travers notre approche est de dégager le maximum de tâches pouvant s'effectuer en parallèle tout en minimisant les temps de communication entre les processeurs et d'équilibrer les affectations de tâches entre les processeurs. Dans une première étape, il s'agit de chercher, dans l'algorithme Apriori, les parties potontiellement parallélisables. Il est donc nécessaire de le décomposer en tâches, de rechercher les relations de dépendance entre ces tâches et la programmation parallèle des tâches indépedentantes. Une étude détaillée de l'algorithme Apriori, nous a permis de dégager plusieurs boucles, à partir desquelles nous pouvons extraire le maximum de parallélisme possible. Avant de définir notre nouvelle approche, nous allons commencer par préciser quelques définitions.

4.2. Construction du graphe de tâches

Pour l'algorithme Apriori, nous avons dégagé les tâches suivantes : (i) T1 : Lecture des transactions de la base de données ; (ii) T2-1 : Génération et tri des 1-itemsets fréquents ; (iii) T2-2 : Suppression des itemsets non fréquents de la base de données ; (iv) T3 : Construction de l'arbre des transactions ; (v) T4 : Construction de l'arbre des itemsets ; (vi) T5 : Génération des k-itemsets candidats à partir des (k-1) itemsets fréquents ; (vii) T6-1 : Calcul de la fréquence des k-itemsets fréquents ; (viii) T6-2 : Elagage et génération des k-itemsets fréquents,(ix) Test : Passage au niveau $(k + 1)$ suivant. Entre les tâches T1 et T2-1 nous avons une dépendance de flux. Par contre, les tâches T2-1 et T2-2 sont indépendantes.

4.3. Graphe de dépendance

Nous présentons dans la figure (1) le graphe de dépendance des différentes tâches de l'algorithme Apriori.

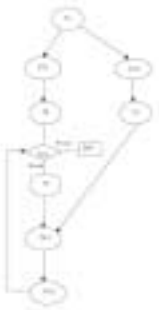

Figure 1. *Graphe de dépendance de tâches*

Figure 2. *Diagramme de GANTT*

Malgré qu'il existe une dépendance entre les tâches T5, T6-1 et T6-2, nous pouvons les paralléliser en utilisant le principe de pipeline (i.e. exécution des tâches en étages). En effet, dès qu'un nouveau (k+1)-itemset candidat est généré (tâche 5), la tâche T6-1 peut être exécutée afin de calculer la fréquence des items et par la suite l'élagage se fait via la tâche T6-2. Le

même processus se réitère pour chaque nouveau itemset candidat généré. Dans la figure (3), nous présentons la partie qui peut se faire en pipeline.

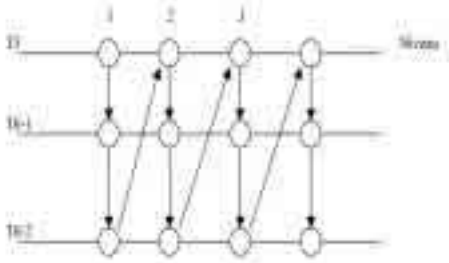

Figure 3. *Tâches en pipeline*

4.4. Tâches et parallélisme interne

Dans cette sous section, nous allons décrire la possibilité de parallélisation des différentes tâches, que nous avons défini dans la section précédente (parallélisation intra-tâche) :(i) **Tâche T1** : Cette tâche peut être effectuée en parallèle. Il s'agit de partitionner les données de la base de transactions entre les différents processeurs, afin de déterminer les itemsets et calculer leurs fréquences. Un calcul du support global des itemsets sera effectué à la fin ; (ii) **Tâche T2-1** : Cette tâche peut également être parallélisée. Il s'agit de partitionner les 1-itemsets candidats entre les différents processeurs. Au niveau de chaque processeur, nous procédons à une phase d'élagage qui consiste à éliminer les itemsets non fréquents et à une phase de tri. Par la suite, les processeurs échangent leurs résultats pour trier l'ensemble des 1-itemsets fréquents ; (iii) **Tâche T2-2** : Dans cette tâche, il s'agit de partitionner l'ensemble des transactions entre les différents processeurs. Elle peut donc être parallélisée, et chaque processeur s'occupe d'éliminer les itemsets non fréquents de sa base locale ; (iv) **Tâche T4** : Cette tâche consiste à partitionner les 1-itemsets fréquents entre les différents processeurs. Chaque processeur peut construire, localement, une branche d'un arbre de hachage. A la fin, une phase de communication est nécessaire pour construire la totalité de l'arbre ; (v) **Tâche T5** : Cette tâche est également parallélisable. Il s'agit d'attribuer à chaque processeur une copie partielle de la structure d'arbre des k-itemsets. Chaque processeur génère les nouveaux sous-ensembles (k+1)-itemsets candidats ; (vi) **Tâche T6** : Cette dernière tâche peut également être parallélisée. Les (k+1)-itemsets candidats sont partagés entre les différents processeurs, et chaque processeur effectue une phase d'élagage pour supprimer les itemsets non fréquents.

5. Expérimentations et évaluation

Dans le cadre de nos expérimentations, nous avons utilisé, à l'heure actuelle, trois bases de tailles différentes afin de comparer les performances des versions séquentielles et parallèles de l'algorithme Apriori. Les caractéristiques de ces trois bases sont présentées dans le tableau suivant :

Nom base	Nbre transactions	Nbre items	Taille moyenne transaction
Base10k	10000	883	19
Base50k	50000	884	10
Base100k	100000	976	19

La version séquentielle a été appliquée sur les trois bases citées ci dessus. En variant le support, nous avons obtenu des temps de calcul qui sont résumés par la figure suivante :

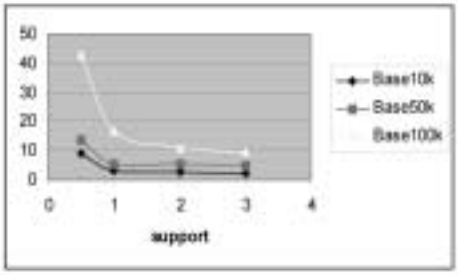

Figure 4. *Temps séquentiel/support sur les différentes bases*

En faisant l'analyse de ces valeurs, nous remarquons que le temps de calcul se réduit en augmentant le support. Ceci s'explique par le fait que l'augmentation du support diminue considérablement le nombre d'itemsets fréquents et par conséquent le nombre de candidats diminue. Ceci entraine donc une réduction du temps de calcul.
La première version parallèle de l'algorithme Apriori, que nous avons implémenté, comporte une version parallèle de la première tâche T1. Les figures suivantes donnent les temps de traitement, les temps de communication entre processeurs ainsi que le temps de calcul parallèle total.

L'augmentation du nombre de processeurs entraîne une diminution du temps de traitement, avec en contre partie une augmentation des temps de communications entre processeurs. Ceci peut s'expliquer par le fait que la communication entre les processeurs d'un même noeud (composé de 4 processeurs) est moins coûteuse que celle entre des processeurs de noeuds différents. C'est pourquoi qu'il y a toujours une augmentation de temps de calcul parallèle total, lors du passage d'un noeud à un autre alors qu'il y a une diminution lors de l'éxecution du programme au sein du même noeud.

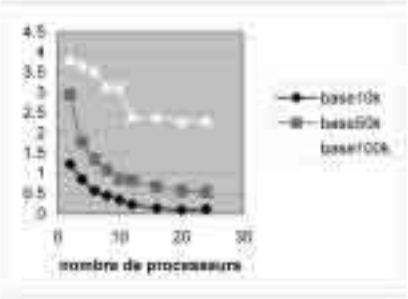

Figure 5. *Temps de traitement*

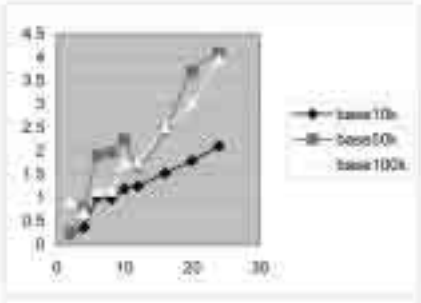

Figure 6. *Temps de communication*

Ainsi, nous pouvons conclure, que si les traitements à effectuer sont importants, le temps de traitement sera dominant par rapport au temps de communication. Les effets de la parallélisation seront donc bénéfiques. Par contre, si le volume de traitement est faible et que par ailleurs, nous augmentons le nombre de processeurs, le temps de calcul sera pénalisé par le temps de communications. Dans ce cas, l'exécution parallèle ne donne pas un gain significatif.

6. Conclusion

Les techniques de datamining permettent d'exploiter des données volumineuses, afin d'extraire les connaissances cachées qu'elles renferment. Pour extraire cette connaissance, les algorithmes associés à ces techniques réalisent plusieurs accès séquentiels aux ensembles de données (bases). Ces accès séquentiels et multiples ont des incidences négatives sur les temps de réponse. Pour améliorer ces temps, plusieurs approches (complémentaires dans certains cas) ont été utilisées. Parmi ces approches, le parallélisme semble être une voie prometteuse, quand il est bien exploité et rentable, pour améliorer les performances des algorithmes de datamining. Bien que ces approches aient permis d'améliorer les performances des algorithmes de découverte de règles associatives, elles restent néanmoins perfectibles. Dans ce contexte, nous avons proposé une approche de parallélisation hybride de l'algorithme Apriori. Cette approche est basée sur la distribution des données et le parallélisme au niveau calcul. Il faut noter que cette approche est la première tentative de parallélisation de l'algorithme Apriori qui essaye d'extraire le maximum de parallélisme contenu dans les tâches qui définissent la structure de l'algorithme. L'approche que nous utilisons est actuellement en cours d'expérimentation sur une machine IBM RS/6000 de 32 processeurs, en utilisant les environnements MPI et Open_MP. Les premiers résultats que nous avons obtenus, et qui sont contenus dans cet article, semblent très prometteurs, d'autant plus qu'ils ne concernent que la parallélisation de la tâche T1 de l'algorithme Apriori. Comme perspectives futures, nous allons poursuivre nos expérimentations sur la parallélisation de l'algorithme Apriori et nous envisageons de l'intégrer dans d'autres algorithmes parallèles existants, tels que CD, DD et IDD.

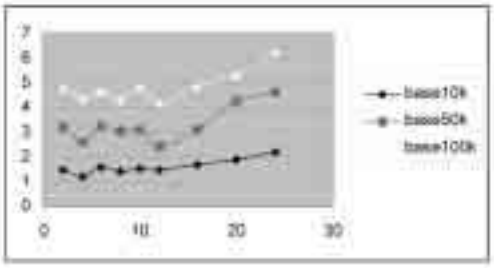

Figure 7. *Temps de calcul parallèle*

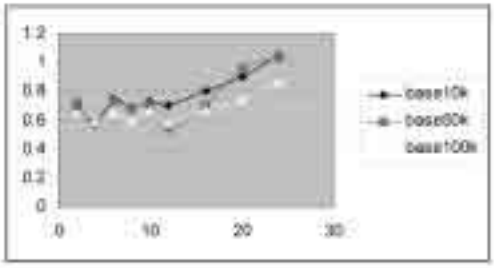

Figure 8. *Accélérations*

7. Bibliographie

[1] R. AGRAWAL, R. SKIRANT « Fast algorithms for Mining Association Rules » *Proceedings of the 20th Intl. Conference on Very Large Databases, Santiago, Chile*page 478-499, june,1994.

[2] R. AGRAWAL, T. IMIELINSKI, A. SWAMI« Mining Association Rules between sets of items in large Databases » *Proceedings of the ACM SIGMOD Intl. Conference on Management of Data. Washington. USA*, pp. 207-216, June, 1993.

[3] R. AGRAWAL, A. ARNING , T. BOLLINGER, M. MEHTA, J. SHAFER, R. STRIKANT« The Quest Data Mining System » *Proc. of the 2nd Intl. Conference on Knowledge Discovery in Databases and Data Mining, Portland, USA*, 1996.

[4] S. PATHASARATHY AND M.J. ZAKI AND W. LI « Memory placement techniques for Parallel association mining » *Technical report,University of Rochester. USA*, 1998.

[5] Y. SLIMANI AND KH. AROUR AND M. JEMNI « Découverte parallèle des règles associatives » *Chapitre du livre RENPAR'07, édition Hermes*, 2004.