الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي
جامعة وهران للعلوم و التكنولوجيا محمد بوضياف

THÈSE

En vue de l'obtention du

Diplôme de Doctorat en Sciences

Présenté par : Khadidja El-Kobra BELBACHIR

Intitulé

DATA MINING DISTRIBUÉ SUR LES GRILLES DE DONNÉES : APPLICATION DE RÈGLES D'ASSOCIATION

Faculté : ***Mathématiques et Informatique***

Département : ***Informatique***

Spécialité : ***Informatique***

Option : ***Systèmes, Réseaux et Bases de données***

Devant le Jury Composé de :

Membres de Jury	*Grade*	*Qualité*	*Domiciliation*
Abdelkader BENYETTOU	*Pr*	***Président***	USTO-MB
Hafida BELBACHIR	*Pr*	***Encadrant***	USTO-MB
Mohamed SENOUCI	*Pr*	***Examinateur***	Univ. d'Oran 1
Ghalem BELALEM	*Pr*	***Examinateur***	Univ. d'Oran 1
Belkacem KOUNINEF	*Pr*	***Examinateur***	INTTIC d'Oran
Lynda ZAOUI	*Pr*	***Examinateur***	USTO-MB

Année Universitaire : 2016 / 2017

Remerciements

J’ai été très chanceuse et très honorée d’être membre de son équipe, motivée, grâce à ses conseils, d’attaquer des problématiques récentes. Elle a été la source du soutien scientifique et humain depuis le premier encadrement. Elle nous a beaucoup appris l’organisation, le sérieux et le bon sens de communication. Je ne pourrais jamais remercier ma directrice de thèse : Madame **Hafida BELBACHIR** pour m’avoir accueilli depuis tous ces années d’études et qui m’a apportée de sa compétence et de ses précieux conseils.

Je tiens à adresser mes remerciements les plus chaleureux aux membres du jury qui ont bien voulu s’intéresser à mon travail.

J'adresse toute ma reconnaissance à Monsieur le professeur **Abdelkader BENYETTOU**, qui m'a fait l'honneur de présider ce jury.

Je remercie également Messieurs : **Mohamed SENOUCI**, **Ghalem BELALEM, Belkacem KOUNINEF** et Madame **Lynda ZAOUI**, d'avoir acceptée d'examiner ce travail.

Je remercie toute personne a contribué et aider, du porche ou du loin, pour réaliser ce travail.

الملخص

شبكة البيانات تتوافق مع مفهوم قد ظهر سنة 2000 في المجتمع العلمي. توجد عدة تعريفات، ولكن كلها تخدم نفس الفكرة. في الواقع, الشبكة المعلوماتية عبارة عن مجموعة من الموارد المتنوعة الموزعة و المشتركة من قبل مؤسسات مختلفة، أكاديمية أو صناعية، لكل منها سياسات إدارية خاصة بها.

نظرا لكمية البيانات الهائلة المخزنة و المعالجة بواسطة برامج المراقبة والإدارة لموارد الشبكة، فمن المهم تطبيق خوارزميات التنقيب عن البيانات لتحليل مستودعات البيانات وذلك من اجل استخدام أفضل للموارد .

استخراج البيانات على الشبكة هو التحدي الجديد لمعالجة كميات كبيرة من البيانات من خلال طرق استكشاف واستخراج المعارف. الخوارزميات الموجودة للتنقيب عن البيانات، تحتاج الى العديد من الحسابات مع تكاليف الدراسة والحاجة لبعض منها للوصول إلى جميع البيانات، التي تتم معالجتها دون الأخذ بعين الاعتبار التوزيع الكلاسيكي للمشكلة. من أجل استغلال الموارد المتاحة على شكل شبكة الحوسبة من الضروري إيجاد أساليب جديدة مع الأخذ بعين الاعتبار خاصية المشاكل لاستخراج المعارف و خصائص شبكة الحسابات (شبكة البيانات).

هذا العمل يهدف لتوفير حلول لإعادة النظر في مهام استخراج البيانات على الشبكات. في المقام الأول نهدف الى تطوير خوارزميات جديدة مكيفة خصيصا لهذا النوع من الهندسة، و الأخذ بعين الاعتبار خاصيات التطبيق الموزعة. وفي المقام الثاني القيام بمجموعة من التجارب للتحقق من فعاليتها كطريقة استدلالية لاستخراج البيانات.

الكلمات المفتاحية: طرق استخراج البيانات، شبكة البيانات

Abstract

The grids correspond to a concept appeared in 2000 in the scientific community. Several definitions exist but converge all worms the same idea. Indeed, a computational grid is a whole of varied and distributed resources shared by various organizations, academic or industrial, having each one its own administration policy.
Being given enormous quantity of stored data treated by control applications and resources management of the grid, it is significant to apply the Data Mining algorithms for the data warehouses analysis in order to better exploit the resources.

Grid Data Mining is a new challenge consisting in treating large data sets by exploration methods and knowledge extraction. The Data Mining existing methods require many calculations with exponential treatment costs and the need for some of them for totality access of the data treated without being able to consider a traditional problem distribution. In order to as well as possible to exploit treatment available resources of calculation grids, It is thus necessary to find new methods taking important from some problems specificities in Data Mining and calculation grids specificities (Data Grids).

Our work aims at proposing solutions to revisit Data Mining tasks on the grids. It is initially a new algorithms planning problem, especially adapted to this architecture type, and fascinating specificities of distributed execution. And in second place, to make experiments to validate them as heuristic Data Mining.

Keywords: Data Mining methods, Data Grids.

Résumé

Les grilles correspondent à un concept apparu en 2000 dans la communauté scientifique. Plusieurs définitions existent mais convergent toutes vers une même idée. En effet, une grille Informatique est un ensemble de ressources variées et distribuées partagées par différentes organisations, académiques ou industrielles, ayant chacune sa propre politique d'administration.
Étant donnée l'énorme quantité de données stockées traitées par les applications de contrôle et de gestion des ressources de la grille, il est important d'appliquer les algorithmes de data Mining pour l'analyse des entrepôts de données afin de mieux exploiter les ressources.
Le Data Mining sur Grille, est un nouveau challenge consistant à traiter de grandes quantités de données par des méthodes d'exploration et d'extraction de connaissances. Les méthodes existantes de data Mining nécessitent de nombreux calculs avec des coûts exponentiels de traitement et la nécessité pour certaines d'entre elles d'accéder à la totalité des données traitées sans pouvoir envisager une distribution classique du problème. Afin d'exploiter au mieux les ressources de traitement disponibles de type grille de calcul. Il est donc nécessaire de trouver de nouvelles méthodes prenant en compte les spécificités des problèmes de data Mining et les spécificités du grille de calcul (grille de données).
Notre travail vise à proposer des solutions pour revisiter les tâches de data Mining sur les grilles. Il s'agit en premier lieu de concevoir de nouveaux algorithmes spécialement adaptés à ce type d'architecture, et prenant en compte les spécificités d'exécution distribuée. Et en seconde lieu, faire des expérimentations pour les valider comme heuristiques de data Mining.

Mots clés : techniques du data mining, grilles de données.

CONTRIBUTION DE L'AUTEUR

Publication Internationale

- K. Belbachir, H. Belbachir. « Parallel Mining Association Rules in Calculation Grids». *Oriental Journal of Computer Science & Technology. ISSN: 0974-6471, Vol. 6, No. (3): 321-331. September 2013.*

Communications Nationales ou Internationales

- K. Belbachir, H. Belbachir « The Parallelization of Algorithm Based on Partition Principle for Association Rules Discovery».International Conference on Telecommunication and ICT - ICTTelecom'15, 16-17 May 2015 Oran Algeria.
- K. Belbachir, H. Belbachir « An approach for Extraction the Spatial Association Rules». ICBIDM 2014: 16th International Conference on Business Intelligence and Data Mining. Paris, France October 30 - 31, 2014.
- K. Belbachir, H. Belbachir «Un algorithme parallèle pour l'extraction des règles d'association dans les grilles de calcul». Colloque Africain Sur La Recherche En Informatique Et En Mathématiques Appliquées. CARI'12 Algiers, Alegria 2012. 13-16 October, 2012.
- K. Belbachir, H. Belbachir « The Parallelization of Algorithm Based on Partition Principle for Association Rules Discovery». The 3rd International Conference on Multimedia Computing and Systems. ICMCS'12. Tangier, Morocco. May 10-12 2012.
- K. Belbachir, H. Belbachir «Une approche pour l'extraction de Règles d'association dans les bases de données Spatiales». Congres méditerranéen des télécommunications Casablanca Maroc. CMT 2010. March 18-20, 2010.
- K. Belbachir, H. Belbachir. «Data Mining spatial : Extraction de règles d'association». Image and Signal Processing and their Applications, Université de Mostaganem Algérie. ISPA 2009. October 19-21 2009
- K. Belbachir, H. Belbachir. «An approach for Extraction the Spatial Association Rules ». International Conference on Information & Communication Systems Jordanie Amman. ICICS 2009. November 1st 2009.
- K. Belbachir, H. Belbachir «Extraction de règles d'associations spatiales : méthode et application». Colloque International Méthodes et Outils d'Aide à la décision, MOAD 2007 Novembre 18-20, 2007 Bejaia, Algérie
- K. Belbachir, H. Belbachir. « Extraction de règles d'associations pour les applications spatiales». Colloque International sur l'Optimisation et les Systèmes d'Information, COSI 2007 Juin 11-13, 2007Oran, Algérie
- K. Belbachir, H. Belbachir. «Etudes de l'application du data Mining à l'analyse spatiale : Extraction de règles d'associations». The Eight International Symposium on Programming and Systems, ISPS 2007 May 7-9, 2007 Algiers, Algeria.

TABLE DES MATIÈRES

REMERCIEMENTS ..... **II**

الملخص ..... **III**

ABSTRACT ..... **IV**

RESUME ..... **V**

CONTRIBUTION DE L'AUTEUR ..... **VI**

TABLE DES MATIÈRES ..... **VII**

LISTE DES FIGURES ..... **XI**

LISTE DES TABLEAUX ..... **XIII**

INTRODUCTION GÉNÉRALE **1**

CHAPITRE I : ÉTAT DE L'ART SUR LE DATA MINING DISTRIBUÉ **5**

I.1. INTRODUCTION 5
I.2. DATA MINING 5
I.2.1. Les taches du Data Mining *6*
I.2.2. Les algorithmes du data Mining *8*
I.3. LES GRILLES INFORMATIQUES 8
I.3.1. Historique et définitions *9*
I.3.2. Les différents types de grille informatique *9*
I.3.2.1. Les grilles d'information: 9
I.3.2.2. Les grilles de stockage: 9
I.3.2.3. Les grilles de calcul: 10
I.3.3. Caractéristiques de la grille *10*
I.3.4. les topologies des grilles *10*
I.3.5. Architectures de grille *11*
I.3.4. Quelques projets de grilles informatiques *13*
I.3.4.1. Etats-Unis 13
I.3.4.2. Asie 14
I.3.4.3. Europe 14
I.4. DATA MINING DISTRIBUÉ 15
I.4.1. Les différents concepts de parallélisme *16*
I.4.1.1. Machines à mémoire distribuée(DMM) et systèmes à mémoire partagée(SMP) : 17
I.4.1.2. Parallélisme de tâche et parallélisme de données 17
I.4.1.3. Équilibrage de charge Statique et dynamique 17
I.4.1.4. Fragmentation Horizontal et Vertical des données 18
I.4.2. Les algorithmes du data mining distribué *18*
I.4.2. 1. Les règles d'association distribuées 19
I.4.2. 2. Le Clustering distribué 19
I.4.2.3. La classification distribuée 19
I.5. CONCLUSION 22

CHAPITRE II : LES RÈGLES D'ASSOCIATION DISTRIBUÉES **24**

II.1. INTRODUCTION 24
II.2. LES RÈGLES ASSOCIATIVES 24
II.2.1. Définitions *25*

II.2.2. La recherche des itemsets fréquents *26*
II.2.3. La génération des règles d'association *26*
II.3. LES ALGORITHMES D'EXTRACTION DE RÈGLES D'ASSOCIATION SÉQUENTIELS 26
II.3.1. L'algorithme Apriori *26*
II.3.2. DHP (Direct Hashing and Pruning) *28*
II.3.3. AprioriTid *28*
II.3.4. Partition *28*
II.3.5. Eclat/ MaxEclat/ clique / MaxClique *29*
II.3.6. Sampling *29*
II.3.7. DIC (Dynamic Itemsets Counting) *29*
II.3.8. FP-Growth (Frequent-Pattern Growth) *29*
II.4. LES ALGORITHMES D'EXTRACTION DE RÈGLES D'ASSOCIATION PARALLÈLES ET DISTRIBUÉS 30
II.4.1. Parallélisme de données *30*
II.4.1.1. Count Distribution (CD) 31
II.4.1.2. Fast Distributed Mining (FDM) 33
II.4.1.3. Distributed Mining of Association rules (DMA) 33
II.4.1.4. Optimized Distributed Association Mining (ODAM) 34
II.4.2. Parallélisme de taches *35*
II.4.2.1. Data Distribution (DD) 35
II.4.2.2. Candidate Distribution (CandD) 36
II.4.2.3. Intelligent Data Distribution (IDD) 37
II.4.2.4. Hash Partioned Apriori (HPA) 39
II.4.3. D'autres types de parallélisme *39*
II.4.3.1. Hybrid Distribution (HD) 39
II.4.3 2. ParEclat 40
II.5. DISCUSSION ET ÉTUDE COMPARATIVE 41
II.5.1. Parallélisme de données *41*
II.5.2. Parallélisme de taches *42*
II.6. CONCLUSION 44
CHAPITRE III : CLUSTERING DISTRIBUÉ 46
III.1. INTRODUCTION 46
III.2. LE CLUSTERING 46
III.2.1. le but de clustering *47*
III.2.2. Les étapes du processus de clustering *48*
III.3. CONCEPTS DE BASE 49
III.3.1. définitions et notations *49*
III.3.2. la similarité entre les objets ou individus *50*
III.3.2.1. la distance entre les données binaires 51
III.3.2.2. la distance entre les données numériques (continues ou discrètes) 52
III.3.2.3. la distance entre les données ordinales 53
III.3.2.4. la distance entre les données nominales 53
III.3.3. les différents types de cluster *54*
III.4. ÉVALUATION DE LA QUALITÉ DE CLUSTERING 54
III.4.1. les mesures de qualité internes *55*
III.4.2. les mesures de qualité externes *55*
III.5. PROBLEMES DU CLUSTERING 55
III.5.1. le choix du nombre de clusters *56*

III.5.2. paramétrage des algorithmes *56*
III.5.3. validité des clusters *57*
III.5.4. le choix du meilleur algorithme de clustering *58*
III.6. LES ALGORITHMES DE CLUSTERING 59
III.6.1. les algorithmes hiérarchiques *59*
III.6.2. Les algorithmes de partitionnement *60*
III.6.3. Les algorithmes basés sur la densité *60*
III.6.4. les algorithmes basés sur les graphes *60*
III.6.5. Les algorithmes basés sur les grilles *60*
III.6.6. Les algorithmes basés sur un modèle *60*
III.7. LES ALGORITHMES DE PARTITIONNEMENT EN CLUSTERING 61
III.7.1. l'algorithme K-means *61*
III.7.2. l'algorithme k-medoids *63*
III.8. LES ALGORITHMES DE PARTITIONNEMENT PARALLELES 64
III.8.1. Un algorithme de clustering sur Multiprocesseurs à Mémoires Distribuées *65*
III.8.1.1. L'algorithme k-means parallèle 65
III.8.2. Le Clustering K-Means Parallèle basé sur MapReduce *67*
III.8.2.1. L'algorithme Pkmeans basé sur MapReduce 67
III.8.3. Un Algorithme de Clustering Parallèle pour les données Catégoriques *70*
III.8.3.1. L'algorithme k-Modes parallèle 70
III.8.4. Algorithme de clustering k-means parallèle sur un réseau de stations de travail *72*
III.8.5. L'algorithme CLARANS parallèle *73*
III.9. CONCLUSION 74
CHAPITRE IV : APPROCHES PROPOSÉES 75
IV.1. INTRODUCTION 75
IV.2. L'ALGORITHME D'EXTRACTION DE REGLES D'ASSOCIATION PARALLELE 75
IV.2.I. l'algorithme Partition *76*
IV.2.1.1. Phase I 76
IV.2.1.2. Phase II 78
IV.2.2. l'algorithme Partition parallèle *78*
IV.2.2.1. la version parallèle de Partition 79
IV.2.2.2. Exemple de déroulement de l'algorithme Partition parallèle 82
IV.3. L'ALGORITHME DE CLUSTERING PARALLELE ET DISTRIBUE 85
IV.3.1. l'algorithme k-prototypes *86*
IV.3.1.1. les préliminaires mathématiques 86
IV.3.1.2. l'algorithme 87
IV.3.2. l'algorithme k-prototypes parallèle *89*
IV.3.2.1. Le modèle de parallélisme 90
IV.3.2.2. la version parallèle de k-prototypes 90
IV.4. CONCLUSION 93
CHAPITRE V : RÉSULTATS, ANALYSES ET DISCUSSIONS 94
V.1. INTRODUCTION 94
V.2. L'ENVIRONNEMENT DE TRAVAIL 94
V.2.1. JCreator *94*
V.2.2. Le simulateur GridSim *95*
V.2.3. Configuration matériel *95*

V.3. EXPERIENCES1 96
V.3.1 Comparaison entre Count Distribution et partition parallèle 96
V.3.2 Évaluation de l'algorithme Partition parallèle en fonction du taux de parallélisme 101
V.4. EXPERIENCES2 102
V.4.1. La base de données 103
V.4.2. Prétraitement de la base de données 103
V.4.3. Evaluation de l'impact du prétraitement (partitionnement intelligent) sur l'algorithme Partition parallèle 104
V.4.4. Évaluation de l'algorithme Partition parallèle après le prétraitement de la base en fonction du nombre d'attributs et de sites 108
V.5. CONCLUSION 112
CONCLUSION GÉNÉRALE ET PERSPECTIVES 113
BIBLIOGRAPHIE 115

LISTE DES FIGURES

Figure I-1: Data Mining dans le processus de découverte de la connaissance 6
Figure I-2 : Différentes topologies des grilles 11
Figure I-3 : L'architecture de la grille : couches et composants 12
Figure I-4: Architecture générale du data mining distribué 16
Figure I-5: Fragmentation horizontale *Figure I-6: Fragmentation verticale* 18
Figure II-1 : L'algorithme Apriori 27
Figure II-2: la procédure Apriori-Gen 27
Figure II-3 : Illustration de l'Algorithme Count Distribution 31
Figure II-4 : L'algorithme Count Distribution 32
Figure II-5 : Exemple de l'Algorithme Count Distribution 33
Figure II-6 : Illustration de l'Algorithme Data Distribution 36
Figure II-7 : Illustration de l'Algorithme Intelligent Data Distribution 38
Figure III-1 : différentes manières de clustering pour le même ensemble de points 47
Figure III-2 : Les étapes du processus de clustering 48
Figure III-3 : Exemple de données produisant des résultats différents suivant l'initialisation de k-means .. 57
Figure III-4 : Algorithme k-means 62
Figure III-5 : L'algorithme k-medoids 63
Figure III-6 : L'algorithme K-means parallèle 66
Figure III-7 : Map-function 68
Figure III-8 : Combine-function 69
Figure III-9 : Reduce-function 70
Figure III-10 : Algorithme de k-mode Parallèle 71
Figure III-11 : l'algorithme k-means distribué 72
Figure III-12 : L'algorithme CLARANS parallèle 73
Figure IV-1 : L'algorithme Partition 76
Figure IV-2 : La fonction gen_large_itemsets 77
Figure IV-3 : Représentation TIDs et TID-Liste 78
Figure IV-4 : La procédure gen_final_count 78
Figure IV-5 : La procedure count-candidat-final 79
Figure IV-6 : L'algorithme Partition parallèle 80
Figure IV-7: Schéma de l'algorithme Partition parallèle 81
Figure IV-8 : Génération des itemsets fréquents localement dans le Site1 82
Figure IV-9: Génération des itemsets fréquents localement dans le Site2 83
Figure IV-10 : Processus d'attribution initial 88
Figure IV-11 : processus de réattribution 89
Figure IV-12 : l'algorithme k-prototypes parallèle 91
Figure IV-13 : Schéma de l'algorithme k-prototypes parallèle 93
Figure V-1 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 30000 transactions) 97
Figure V-2 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 50000 transactions) 98
Figure V-3 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 70000 transactions) 99

Figure V-4 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 30000 transactions, minsup =30%) *100*
Figure V-5 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 50000 transactions, minsup =30%) *100*
Figure V-6 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 70000 transactions, minsup =30%) *101*
Figure V-7 : L'algorithme Partition parallèle sur 30000, 50000, 70000 transactions *102*
Figure V-8 : L'algorithme k-prototypes parallèle sur 1000, 5000, 10000 et 30000 transactions *103*
Figure V-9 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 1000 transactions *105*
Figure V-10 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 5000 transactions . *106*
Figure V-11 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 10000 transactions *107*
Figure V-12 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 30000 transactions *108*
Figure V-13 : L'algorithme Partition parallèle après prétraitement de données sur 1000 transactions ... *109*
Figure V-14 : L'algorithme Partition parallèle après prétraitement de données sur 5000 transactions ... *109*
Figure V-15 : L'algorithme Partition parallèle après prétraitement de données sur 10000 transactions *110*
Figure V-16 : L'algorithme Partition parallèle après prétraitement de données sur 10000 transactions *110*
Figure V-17 : Performances de l'algorithme Partition parallèle après prétraitement en fonction du nombre de sites *111*

LISTE DES TABLEAUX

Tableau III-1 : Tableau de contingence *51*
Tableau III-2 : Syntaxe et fonctionnalités des routines de MPI *66*
Tableau IV-1 : La base de données initiale *82*
Tableau IV-2 : La base de données dans le Site1 ..... *82*
Tableau IV-3: La base de données dans le Site2 ..... *82*
Tableau IV-4 : Génération des candidats globaux par le coordinateur *83*
Tableau IV-5 : Compte des fréquences des candidats globaux dans le Site1 ..... *84*
Tableau IV-6 : Compte des fréquences des candidats globaux dans le Site2 ..... *84*
Tableau IV-7 : Les itemsets fréquents *85*

INTRODUCTION GÉNÉRALE

Contexte générale

Ces dernières années les entreprises et les organisations ne cessent d'accumuler de grandes masses de données qui sont souvent sous exploitées, alors qu'elles peuvent renfermer des connaissances stratégiques que des experts peuvent ignorer.

Ces réservoirs de données représentent une importante mine d'informations que les entreprises doivent exploiter et explorer pour découvrir des informations pertinentes et utiles à des fins de prédiction et de prise de décisions. Par exemple, la plupart des grands magasins récoltent les informations sur leurs clients et étudient leurs comportements afin de prédire leurs besoins. Dans les établissements médicaux, les résultats d'analyses médicales sont également conservés pour être analysés. Ainsi, de nombreux processus industriels sont de plus en plus contrôlés par l'informatique.

Les entreprises sont alors de plus en plus confrontées à la nécessité de maîtriser cette information et le choix des outils permettant l'analyse et l'exploitation de ces données s'avère primordial. Répondant à ces besoins, l'extraction de connaissances à partir de données (Knowledge Discovery in Databases, KDD) est alors mise en place au début des années 1990. Le processus d'extraction de connaissances est historiquement issu des travaux de nombreuses disciplines : l'apprentissage automatique, l'analyse de données et la statistique.

Les mécanismes de data mining qui est le cœur du processus d'extraction des connaissances sont nombreux. Le problème de découverte de liens ou de la recherche de règles d'association est une méthode de Data Mining de type apprentissage non supervisé permettant d'extraire des relations intéressantes entre les données. Ces règles expriment une possibilité d'association entre différents attributs, et peuvent servir pour la prédiction ou pour la prise de décision.

D'autre part, la technologie moderne de l'information produit chaque année des ordinateurs plus puissants, qui nous permettent de collecter, sauvegarder, transférer des énormes quantités de données à prix très réduit. Vu que les algorithmes du data mining demandent une grande capacité de calcul, la difficulté de ces algorithmes augmente exponentiellement avec cette augmentation dans les bases de données. Ainsi, le développement des réseaux d'ordinateurs permettant le calcul distribué sur des réseaux

locaux comme à grande échelle tels que les grilles. Ces derniers ont encouragé le développement des algorithmes de Data Mining parallèles et distribué. Donc on peut dire que le data mining distribué se situe à la conjonction de deux évolutions majeures. D'une part, l'explosion de masses de données importantes qui est souvent réparties dont il faut savoir extraire des connaissances utiles, d'autre part le développement des réseaux d'ordinateurs permettant le calcul distribué.

Objectifs et motivations

Quand l'ensemble de données est volumineux la vitesse des taches du data mining est cruciale. De ce fait, la rapidité de calcul est une contrainte indispensable dans le développement étendu des algorithmes du data mining pour analyser les ensembles de données sur des machines parallèles avec des centaines ou des milliers de processeurs.

L'absence de la mémoire centrale partagée dans les environnements parallèles et distribués tel que les grilles impose la distribution de la base en partitions et le traitement de ces partitions en parallèle. Alors, les traitements parallèles d'un algorithme du data mining distribué doivent être les plus indépendants possible afin de minimiser les couts de communications et synchronisations qui dégradent l'efficacité de l'algorithme et augmente le temps d'exécution.

Dans le cadre du processus d'extraction des connaissances à partir des données, nous proposons dans cette thèse d'explorer une méthode adaptée à un environnement parallèle et distribué tel que les grilles pour l'extraction de règles d'association. Le but de notre travail est de minimiser le temps d'exécution en réduisant le cout de parallélisation et de communication.

Contributions

Les contributions effectuées durant cette thèse considèrent deux parties dans ce contexte de recherche. Le but est de proposer de nouvelles approches et d'améliorer les algorithmes existants pour l'extraction de connaissances dans un environnement parallèle ou distribué à un grand ou petit échelle.

Dans un premier temps, nous identifions nos contributions qui apportent d'avantage par rapport aux travaux existants en terme temps d'exécution. Généralement, les critères

primordiaux des algorithmes parallèles est de minimiser le nombre de balayage des données et les étapes de synchronisations et de communication.

Afin d'entretenir ces deux critères, nous proposons une version parallèle de l'algorithme séquentiel Partition, ce dernier ne demande que deux parcours de la base. Alors, Nous avons pensé à réformer l'algorithme Partition pour réduire le nombre de balayage de la base en un seul, ensuite concevoir une version parallèle pour l'algorithme réformé. Cette proposition permet de réduire considérablement les synchronisations entre les sites.

Dans un deuxième temps, nous avons pensé à une distribution intelligente de la base sur la grille en utilisant le clustering afin de minimiser l'espace de recherche dans chaque machine (toutes les combinaisons possibles pour les items existants dans la base de données) et en conséquence réduire le temps de traitement. A cet effet, nous proposons un algorithme de clustering parallèle pour un environnement distribué. Nous avons alors choisi l'algorithme séquentiel k-prototypes qui traite les données mixtes (numériques et catégoriques) et en se basant sur le modèle passage de messages, un algorithme de clustering parallèle et distribué est conçu.

Organisation de la thèse

Cette thèse est organisée en cinq chapitres :

Le chapitre I présente un état de l'art sur le data mining distribué. A ce titre, nous introduisons les taches et les algorithmes du data mining. Nous présentons également les grilles informatiques avec leurs caractéristiques, topologies et leurs architectures. Nous verrons par la suite, une synthèse des algorithmes du data mining distribué en abordant les différents concepts de parallélisme utilisé pour ce type d'algorithmes.

Le chapitre II est dédié à la présentation des règles d'associations distribuées. Nous détaillons les algorithmes pour la découverte des règles d'association dans un environnement parallèle et distribué, en étudiant et comparant les points forts et les points faibles de chaque algorithme. L'objectif recherché est d'introduire et de positionner nos contributions.

Le chapitre III est consacré au concept du clustering et plus précisément le clustering distribué. Nous présentons un état de l'art général des travaux qui traitent le problème de

classification ou de clustering. Nous insistons particulièrement sur les algorithmes de partitionnement et les algorithmes de partitionnement parallèle. Le but est d'envisager un algorithme de clustering parallèle pour un partitionnement intelligent de la base sur une grille.

Le chapitre IV décrit les deux approches proposées, l'algorithme Partition parallèle pour l'extraction de règles d'association et l'algorithme k-prototypes parallèle pour partitionner la base de données intelligemment afin d'accélérer le temps de traitement pour le problème de règles d'association. Nous évoquons les objectifs et les motivations qui nous ont poussés à choisir ces deux algorithmes. Concernant l'extraction de règles d'association, nous proposons l'algorithme Partition parallèle, c'est une amélioration de l'algorithme Partition pour un traitement parallèle en respectant les critères du traitement distribué et minimisant les couts de communication et de synchronisation. D'un autre côté, pour un bon partitionnement de la base tout en respectant les contraintes du support d'exécution, Autrement dit, pas de mémoire partagée et pas de synchronisation, nous présentons notre l'algorithme k-prototypes parallèle qui permet de traiter des données réelles non prétraitées (données numériques et catégoriques) sur une grille de données.

Le chapitre V détaille l'ensemble des travaux réalisés, une analyse et quelques discussions ont été abordées. Nous exposons l'application pour les deux approches proposées sur un jeu de données réelles. La première expérience étude les performances de l'algorithme Partition parallèle en comparant avec d'autre algorithme. Tandis que la deuxième expérience teste l'impact du partitionnement intelligent de l'algorithme k-protoypes parallèle sur l'efficacité de l'algorithme Partition parallèle.

Nous conclurons ensuite notre travail et présenterons quelques perspectives.

CHAPITRE I : ÉTAT DE L'ART SUR LE DATA MINING DISTRIBUÉ

I.1. Introduction

L'extraction de connaissances à partir de données (ECD) a été définie comme un processus consistant à identifier des informations et des relations valides, nouvelles, utiles et compréhensibles. Le processus ECD implique plusieurs étapes, le cœur du processus est l'étape Data Mining (Fouille de données) qui comprend l'application des algorithmes de découverte et d'analyse de grands volumes de données. Ces algorithmes sont des traitements coûteux, et les volumes de données disponibles augmentent avec les possibilités de stockage, imposant le recours au parallélisme.

Le calcul sur la grille (plus communément connu sous l'appellation "grid computing") répond à ce besoin, c'est un concept relativement récent que l'on peut interpréter comme une forme moderne du métacomputing introduit au début des années 80. En théorie, le calcul sur la grille permet d'exploiter des ressources de calcul, de stockage, de visualisation, etc. quasi-illimitées, distribuées sur différents sites géographiques interconnectés par un réseau Intranet ou un réseau à grande échelle comme Internet.

Une introduction générale du data mining sera présentée dans la section suivante, nous décrirons les différentes taches réalisées par le data mining et les algorithmes les plus populaires en présentant leur principe et les approches proposées. Ensuite, nous présenterons les différentes notions de la grille, les types, les topologies, l'architecture de la grille et quelques projets réalisés de la grille. Nous discuterons par la suite le data mining distribué en arborant les concepts de parallélisme qui peuvent être pris en considération pour l'application du data mining sur les grilles, ainsi quelques algorithmes distribués dans la littérature seront présentés. Enfin, nous conclurons cette section en montrant l'utilité du parallélisme surtout dans les grilles pour palier le problème du cout de calcul imposé par les algorithmes du data mining.

I.2. Data Mining

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 des connaissances décisives face au monde extérieur. Pour combler ce besoin, une nouvelle industrie est née, le Data Mining ou bien la fouille de données.

Le Data Mining est populairement connue sous le nom découverte de la connaissance dans les bases de données (Knowledge Discovery in Databases KDD), qui consiste à identifier à partir d'un important volume de données brutes, des connaissances originales auparavant inconnues. Bien que le Data Mining et la découverte de la connaissance (KDD) soient fréquemment traités comme synonymes, le Data Mining est réellement une partie du KDD. La

figure suivante montre le Data Mining comme une étape dans le processus de découverte de la connaissance.

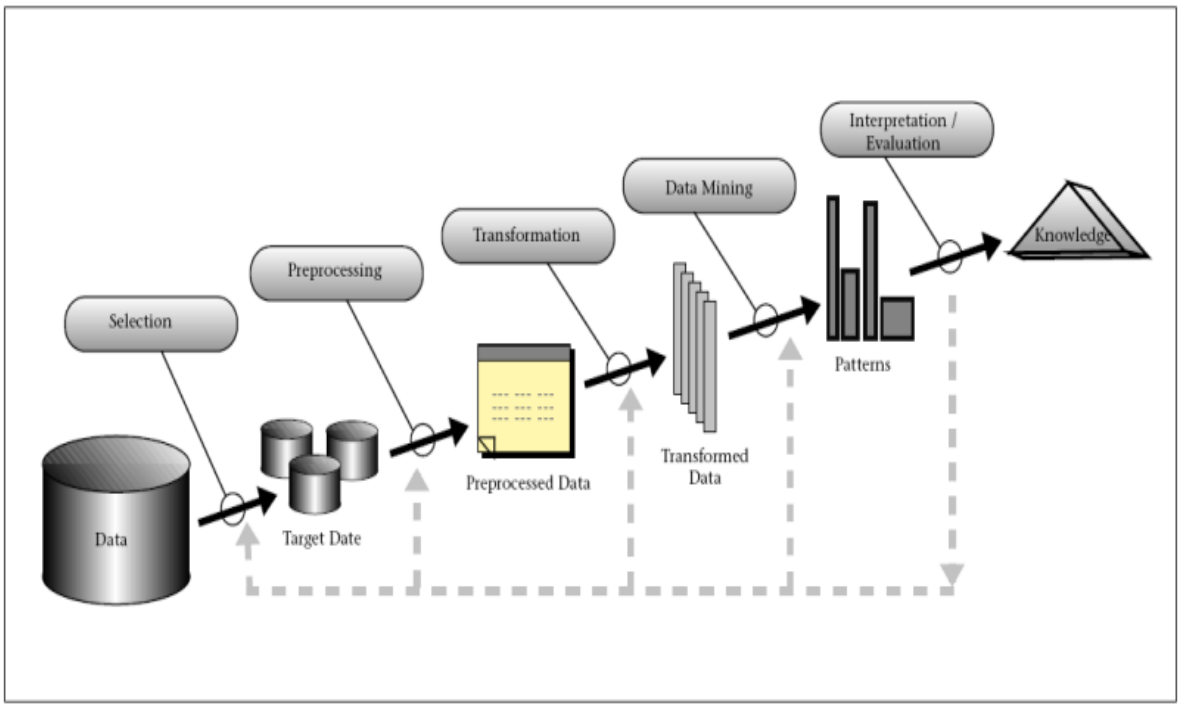

Figure I-1: Data Mining dans le processus de découverte de la connaissance

I.2.1. Les taches du Data Mining

Les modèles qui peuvent être découverts dépendent des tâches du data mining utilisé. Généralement, il y a deux types de tâches: tâches descriptives qui décrivent les propriétés générales des données existantes, et tâches prédictives qui essayent de faire des prévisions se basant sur les déductions dans les données disponibles. Les taches du data mining et les connaissances qu'ils découvrent sont [1]:

Caractérisation : La caractérisation de données est un résumé des caractéristiques générales des objets dans une classe cible, et produit ce qu'on appelle les règles caractéristiques. Les données appropriées à une classe spécifiée par l'utilisateur sont normalement recherchées par une question de base de données et parcourues par un module de synthétisation pour extraire le cœur des données à différent niveaux d'abstraction. Par exemple, on peut caractériser les clients du magazin *OurVideoStore* qui louent régulièrement plus de 30 films par année.

Discrimination : La discrimination de données produit ce qu'on appelle les règles discriminantes, c'est une comparaison des caractéristiques générales des objets entre deux classes appelées la classe cible et la classe contrastant. Par exemple, on peut comparer les caractéristiques générales des clients qui ont loué plus de 30 films au cours de la dernière année avec ceux dont le compte de location est inférieur à 5. Les techniques utilisées pour la discrimination de données sont très semblables aux techniques utilisées pour la

caractérisation de données avec l'exception que les résultats de discrimination incluent les mesures comparatives.

Analyse d'association : L'analyse d'association est la découverte de ce qu'on appelle les règles d'association. Elle étudie la fréquence des items qui se produisent dans la base de données transactionnelle, elle est basée sur un seuil appelé le support qui identifie les ensembles des items fréquents. Un autre seuil, confiance, qui est la probabilité conditionnelle qu'un item apparaît dans une transaction quand un autre item apparaît. Ce seuil est employé pour identifier les règles d'association. L'analyse d'association est généralement utilisée pour l'analyse de panier du marché. Par exemple, elle pourrait être utile pour le directeur d'OurVideoStore pour savoir quels films sont souvent loués ensemble ou s'il y a un rapport entre l'allocation de certains types de films et l'achat de popcorn. Les règles d'association découvertes sont de la forme : P→Q [s,c], où P et Q sont les conjonctions d'attribut, s (le support) est la probabilité que P et Q apparaitront ensemble dans une transaction et c (la confiance) est la probabilité conditionnel que Q apparaît dans une transaction quand P est présent. Par exemple, la règle d'association :

RentType(X, "jeu") ∧ Age(X, "13-19") → Buys(X, "pop") [s=2%, c=55%]

Indiquerait que 2% des transactions considérées sont des clients âgés entre 13 et 19 qui louent un jeu et achètent popcorn, et une certitude de 55% que les clients adolescents qui louent un jeu achètent également popcorn.

Classification : La classification est l'organisation des données dans des classes données. Ainsi connu sous le nom de *classification supervisée*, la classification utilise les étiquettes des classes données pour classer les objets. Les approches de classification utilisent un ensemble d'apprentissage où tous les objets sont déjà associés aux étiquettes de classes données. L'algorithme de classification apprend de l'ensemble d'apprentissage et établit un modèle. Par la suite, ce modèle est employé pour classifier de nouveaux objets. Par exemple, après avoir commencé une politique de crédit, les directeurs d'*OurVideoStore* pourraient analyser les comportements des clients vis-à-vis de leurs crédits, et étiquettent les clients qui ont reçu des crédits avec trois étiquettes possibles "sans risque", "risqué" et "très risqué". L'analyse de classification produirait un modèle qui peut être employé pour accepter ou rejeter les demandes de crédit dans le futur.

Prédiction : La prédiction a attiré une attention considérable donnant des implications potentielles des prévisions réussies dans le contexte d'affaires. Il y a deux types de prévisions: on peut prévoir quelques valeurs de données indisponibles, ou prévoir une étiquette de classe pour quelques données. Le dernier est attaché à la classification. Une fois qu'un modèle de classification est établi basé sur un ensemble d'apprentissage, l'étiquette de la classe d'un objet peut être prévue en se basant sur les valeurs d'attributs de l'objet et les valeurs d'attributs des classes. Cependant, la prévision est plus souvent visée pour prévoir les valeurs numériques manquantes. L'idée principale est d'employer un grand nombre de valeurs du passé pour estimer les probables futures valeurs.

Clustering : Semblable à la classification, le clustering est l'organisation des données dans les classes. Cependant, à la différence de la classification, dans le clustering les étiquettes de classe sont inconnues, c'est à l'algorithme de clustering de découvrir les classes. Le Clustering est également appelé *la classification non supervisée*, car la classification n'est pas basée sur les étiquettes des classes données. Il y a beaucoup d'approches de Clustering qui se basent sur le principe, maximisant la similarité entre les objets dans une même classe (similarité intra-classe) et réduisant au minimum la similarité entre les objets de différentes classes (similarité interclasse).

Analyse des annexes : Les annexes sont des données qui ne peuvent pas être groupées en classes données ou clusters. Elles sont connues comme exceptions ou surprises, il est souvent très important de les identifier. Tandis que les annexes peuvent être considérées comme bruit et être rejetées dans quelques applications, elles peuvent indiquer des connaissances importantes dans d'autres domaines, et peuvent être ainsi très signifiantes dans leurs analyses précieuses.

I.2.2. Les algorithmes du data Mining

Le Data Mining est un ensemble de techniques et de méthodes du domaine des statistiques, des mathématiques et de l'informatique permettant l'extraction, à partir d'un important volume de données brutes, de connaissances originales auparavant inconnues. Il s'agit de fouilles visant à découvrir de l'information cachée que les données renferment et que l'on découvre à la recherche d'associations, de tendances, de relations ou de régularités. Les algorithmes utilisés en Data Mining appartiennent à deux grandes familles :

Les techniques descriptives : Elles visent à mettre en évidence des informations présentes mais cachées par le volume de données. Elles sont utilisées pour réduire, résumer et synthétiser les données dont il n'y a pas de variable cible à prédire.

Les techniques prédictives : Elles visent à extrapoler de nouvelles informations à partir des informations présentes pour expliquer les données, elles sont utilisées aussi pour prédire des variables cibles.

De nombreux algorithmes appartenant à ces deux grandes familles ont été utilisés, parmi les plus connus on peut citer par exemple : Clustering[2][3][4][5][6][7], les arbres de décision[8][9][10][11][12], les règles d'association[13][14], les plus proches voisins[2][15]. Cependant d'autres méthodes existent, comme les réseaux de neurones et les algorithmes génétiques.

I.3. Les grilles informatiques

Les développements récents de technologies de l'information et de la communication permettent de créer aujourd'hui de véritables infrastructures pour le calcul et le stockage de données hétérogènes. Ces infrastructures appelées grilles informatiques.

I.3.1. Historique et définitions

En 1998, Ian Foster & Karl Kesselman ont posé le paradigme de la grille informatique dans leur ouvrage:« The Grid: Blueprint for a New Computing Infrastructure » [16]. L'accès aux services informatiques devrait être comparable en coût et facilité au branchement d'appareils sur le réseau électrique. Cet accès devrait être aisé et transparent quels que soient la puissance nécessaire, la complexité des équipements matériels et logiciels mis en œuvre, les fournisseurs du service [17].

De manière plus formelle, une grille informatique est une forme d'informatique distribuée, basée sur le partage dynamique des ressources entre des participants, des organisations et des entreprises dans le but de pouvoir les mutualiser. En effet, au-delà des besoins en capacité de traitement, l'approche grille permet la co-allocation et le partage dynamique de toutes les ressources de systèmes hétérogènes indépendamment des unités organisationnelles et faire ainsi exécuter des applications de calcul intensif ou des traitements de très gros volumes de données. Ces applications ne seraient pas possibles au sein d'un seul organisme ou d'une seule entreprise. Du point de vue de l'utilisateur, l'origine des ressources utilisées est totalement abstraite et transparente. Ce dernier ne se soucie de rien : ni de la puissance nécessaire pour ses applications, ni du type de machines utilisées, et encore moins de leur localisation physique. C'est à la plate-forme de la grille de gérer tous ces aspects.

Les grilles informatiques sont de nature dynamiques, les sites peuvent quitter ou rejoindre la grille à tout moment ; de même, au sein de chaque site, de nouvelles ressources peuvent être ajoutées, d'autres peuvent tomber en panne ou être déconnectées. Sachant qu'un site dans la grille est un ensemble de ressources informatiques localisées géographiquement dans une même organisation (campus universitaire, centre de calcul, entreprise ou chez un individu) et qui forment un domaine d'administration autonome, uniforme et coordonné [18].

I.3.2. Les différents types de grille informatique

Compte tenu des capacités de la grille informatique, cette dernière garantit des qualités de service non triviales, c'est-à-dire qu'elle se distingue des autres infrastructures dans son aptitude à répondre adéquatement à des exigences. Suivant les services rendus, les grilles informatiques sont classées dans trois types :

I.3.2.1. Les grilles d'information: C'est sans aucun doute la première incarnation évidente du concept de grille. Elle permet le partage de l'information à travers un réseau. On peut, bien évidemment, dire que ces principes sont repris dans l'Internet. Toutefois, on ne se sait pas toujours d'où vient l'information (sources fiables).

I.3.2.2. Les grilles de stockage: Ce principe de grille permet le partage de données externalisées entre plusieurs nœuds vers plusieurs nœuds. Les cas les plus représentatifs de ce concept sont les réseaux d'échange Peer-To-Peer (P2P). Ce système permet l'accès à des données (fichiers, flux...) via un réseau de sites ou serveurs qui contiennent et partagent un index. Les données sont référencées pour optimiser les recherches à travers un moteur de recherche, dont Les fichiers peuvent se trouver sur des nœuds différents du réseau. Il peut donc y avoir plusieurs copies d'un même fichier afin d'optimiser l'obtention de ce dernier. Les

données sont dites "inaltérables", car elles sont marquées et identifiés via une procédure de hachage.

I.3.2.3. Les grilles de calcul: Ces grilles permettent la récupération et l'utilisation de ressources CPU inexploitées. Le principe est simple et permet de regrouper des millions de postes pour un intérêt commun.

I.3.3. Caractéristiques de la grille

Une grille se compose de ressources informatiques, tout élément qui permet l'exécution d'une tâche ou le stockage d'une donnée numérique. La grille et ses ressources doivent répondre aux caractéristiques suivantes [19]:

- **Partagées :** les ressources sont mises à la disposition des différents consommateurs de la grille et éventuellement pour différents usages applicatifs.
- **Distribuées :** les ressources sont situées dans des lieux géographiques différents, et appartiennent à différentes organisations, chacune ayant ses propres politiques de gestion et de sécurités. Ainsi il est indispensable de respecter les politiques de chacune de ces organisations.
- **Hétérogènes :** les ressources sont de toute nature, c.-à-d. de nature hétérogène en termes de matériels et de logiciels. Par exemple, différant par le système d'exploitation ou le système de gestion des fichiers.
- **Coordonnées :** les ressources sont organisées, connectées et gérées en fonction de besoins (objectifs) et contraintes (environnements). Ces dispositions sont souvent assurées par un ou plusieurs ordonnanceurs.
- **Non-contrôlées (ou autonomes) :** les ressources ne sont pas contrôlées par une unité commune. Contrairement à un cluster, les ressources sont hors de la portée d'un moniteur de contrôle.
- **Passage à l'échelle (scalability) :** une grille pourra constituer de quelque dizaines de ressources à des millions voire des dizaines de millions.
- **Nature dynamique des ressources :** dans les grilles ce caractère dynamique est la règle et non pas l'exception. Cela pose des contraintes sur les applications telles que l'adaptation au changement dynamique du nombre de ressources, la tolérance aux pannes et aux délais d'attente.

I.3.4. les topologies des grilles

Les grilles informatiques sont classées dans trois types du point de vue topologie(Figure I-2), ces types sont classés par ordre croissant d'étendue géographique et de complexité. On a : Intragrilles (Intragrids), Extragrilles (Extragrids) et Intergrilles(Intergrids) [20].

- **Intragrille (en analogie avec Intranet) :** c'est la plus simple des grilles, elle est composée d'un ensemble relativement simple de ressources et de services qui appartiennent à une organisation unique. De ce fait, Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion performant et haut-débit, d'un domaine de sécurité unique et maitrisé par les administrateurs de l'organisation et d'un ensemble relativement statique et homogène de ressources.

- **Extragrille (en analogie avec Extranet) :** Une extragrille est un modèle étendu d'intragrille, en agrégeant plusieurs grilles de type intragrille. Par conséquence, les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion hétérogène haut débit (LAN/WAN), de plusieurs domaines de sécurité distincts, et d'un ensemble plus au moins dynamique de ressources.
- **Intergrille (en analogie avec Internet) :** Une grille de ce type est une agrégation de plusieurs grilles de multiples organisations, en une seule grille. La présence de plusieurs organisations dans chaque grille appartenant à l'Intergrille, génèrent les caractéristiques suivantes : la présence d'un réseau d'interconnexion très hétérogène, haut et bas débit (LAN/WAN), de plusieurs domaines de sécurité distincts et ayant parfois des politiques de sécurités différents et même contradictoires, et d'un ensemble très dynamique de ressources.

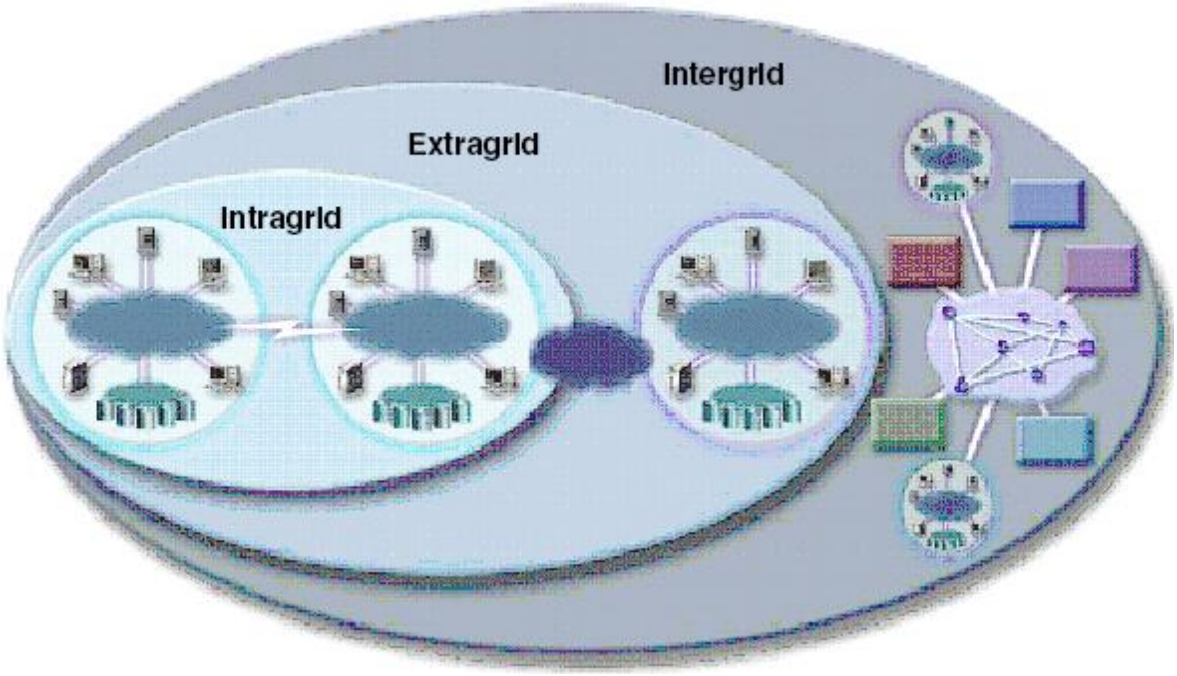

Figure I-2 : Différentes topologies des grilles

Une autre topologie, c'est la notion d'organisation virtuelle (Virtual Organisation) qui a été apparue avec la naissance de la notion de grille de calcul. Où les utilisateurs pourront être groupés, selon leurs différents intérêts, dans telles organisations dont chacune possède sa propre politique. Chaque organisation virtuelle mettra à la disposition de ces utilisateurs un ensemble de ressources sous la forme d'une grille.

I.3.5. Architecture de grille

Une grille informatique est un environnement de systèmes d'exploitation et d'architectures hétérogènes dont l'accès est fourni à l'utilisateur sous une forme unifiée. Pour ce fait, le besoin de développement de protocoles, services et outils est exigent pour répondre spécifiquement aux impératifs suivants :

- Un accès aux ressources à des parties prenantes de différentes natures, possédant donc des droits différents.
- La délégation et la mise en application de réglementations à l'échelle locale et globale.
- Un accès et un partage de ressources variées : ressources physiques, programmes, dossiers, données…
- Un accès et un partage dans des différents scénarios d'usage : utilisateur unique, multiutilisateur
- La mise en œuvre d'une réelle qualité de service (QoS), de la planification, de la co-allocation des ressources et du suivi de l'allocation des ressources entre les différentes parties prenantes.

Ces différents services et outils sont rangés dans des couches, chaque couche est construite sur les services offerts par la couche inférieur, en plus d'agir et de coopérer avec des composants au même niveau. La figure suivante montre l'ensemble des matériels et logiciels dans une architecture typique de grille. Elle est constitué de quatre couches : Fabric, Core middleware, user-level middleware, et la couche Application [21]:

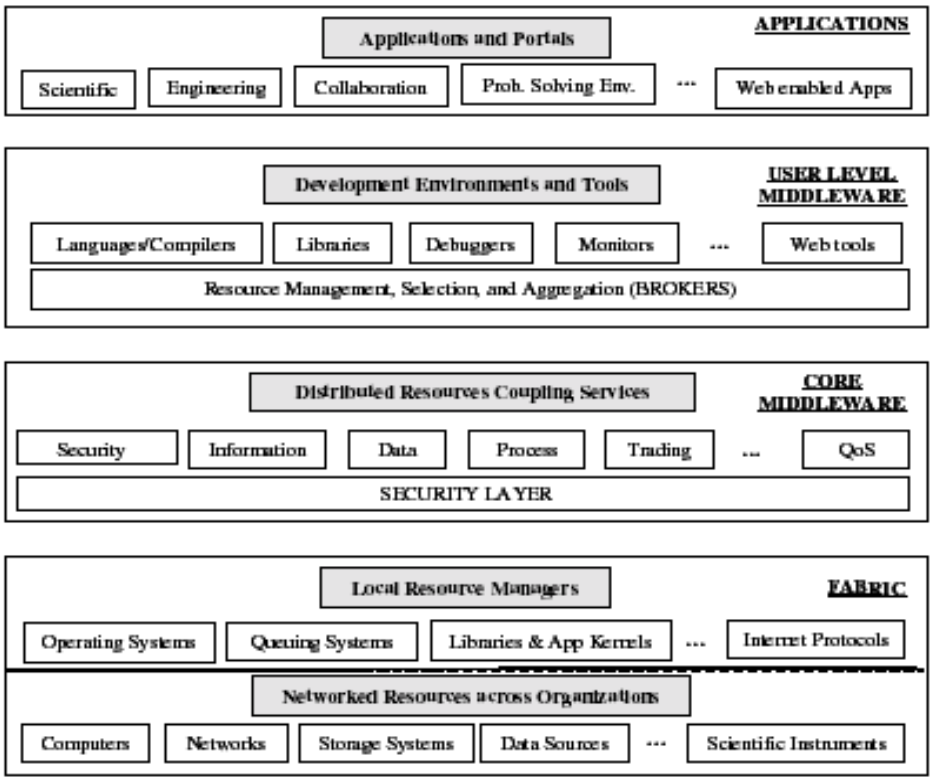

Figure I-3 : L'architecture de la grille : couches et composants

- **Couche Fabric :** elle set composée de ressources distribués tels que, ordinateurs, réseaux, et dispositifs de stockage. Les ressources informatiques représentent de multiple architecture comme clusters, supercalculateurs, serveurs et PCS ordinaires qui exécutent une variété de logiciels d'exploitation.

- **Couche Core middleware :** elle offre des services comme les processus de contrôle à distance, Co-allocation des ressources, l'accès au données, la découverte et l'enregistrement de l'information, la sécurité, et les aspects de la qualité de Service (QoS) comme la réservation de la ressource. Ces services abstraient la complexité et l'hétérogénéité de la couche Fabric en fournissant une méthode cohérente pour accéder aux ressources distribuées.
- **La couche User-level middleware :** elle utilise les interfaces fournies par la couche inferieure middleware pour fournir un niveau d'abstraction et de services plus élevé. Ceux-ci incluent les environnements de développement d'application, les outils de programmation et les courtiers de ressources pour la gestion des ressources et l'ordonnancement des taches des applications pour une exécution sur les ressources globales.
- **La couche Application :** elle est typiquement développée en utilisant les environnements de programmation, les interfaces et les services de courtage et d'ordonnancement fourni par la couche user-level middleware. Par exemple pour une application qui exige une puissance de calcul, un accès aux ensembles de données à distance. Cette couche offre des services d'applications, où les utilisateurs peuvent soumettre les taches et collecter les résultats sur des ressources à distance.

I.3.4. Quelques projets de grilles informatiques

Des dizaines, voire des centaines, de projets de grille sont en cours dans le monde, Les premières réalisations ont d'abord été limitées à l'Europe et aux pays industriels mais un nombre de plus en plus important de pays sont maintenant actifs dans le domaine, parmi lesquels la Chine a fait une entrée en puissance.

Nous présentons ci-dessous quelques exemples de projets dans les différentes régions du monde, par lesquels nous illustrerons le dynamisme que connaît actuellement le sujet. Certains projets sont orientés vers le développement de l'infrastructure et la technologie (essentiellement logicielle), d'autres abordent les problèmes de gestion, authentification, sécurité, etc., et d'autres encore sont orientés vers des domaines d'application particuliers [22].

I.3.4.1. Etats-Unis

Faisant usage de connexions aux réseaux à haute performance, *TeraGrid* intègre des ordinateurs puissants, des données et des outils logiciels, et des sites expérimentaux de haut niveau dans tout le pays. L'ensemble de ces ressources inclut plus de 100 Teraflops de capacité de calcul (un Teraflops représente 1000 milliards d'opérations par seconde) et plus de 15 Petaoctets de capacité de stockage (un Petaoctet égale un million de Gigaoctets). Au moyen de TeraGrid, les chercheurs peuvent accéder à plus de 100 bases de données de disciplines différentes. TeraGrid est coordonné par le "Grid Infrastructure Group" de l'université de Chicago et fonctionne en partenariat avec : Indiana University, Oak Ridge National Laboratory, National Center for Supercomputing Applications, Pittsburgh Supercomputing Center, Purdue University, San Diego Supercomputer Center, Texas

Advanced Computing Center, University of Chicago/Argonne National Laboratory, National Center for Atmospheric Research.

Open Science Grid (OSG) est une infrastructure américaine de calcul en grille pour les applications scientifiques, basée sur une collaboration ouverte entre chercheurs, développeurs logiciels et fournisseurs de ressources de calcul, de stockage de données et d'accès réseau. OSG est réalisé et géré par un consortium qui rassemble les ressources et les chercheurs d'une soixantaine d'universités et de laboratoires nationaux et internationaux.

I.3.4.2. Asie

Au moins cinq importants projets de grille ont été initiés en Chine ; il y a deux au niveau national: ChinaGrid et CNGrid. *ChinaGrid* (China Education and Research Grid) est un projet financé par le Ministère de l'Education, le puissant réseau chinois pour l'éducation et la recherche. Une vingtaine des plus grandes universités participent au projet, qui groupait déjà en 2005 plus de 6 Teraflops de puissance de calcul et plus de 60 Teraoctets de capacité de stockage.

CNGrid (China National Grid) est un projet du Ministère de la Science et de la Technologie pour développer l'informatique en grille pour la recherche scientifique, notamment dans les domaines de l'environnement, des techniques de fabrication, et des applications et services informatiques. Un des buts est de promouvoir le développement de l'industrie informatique. Le projet inclut au départ 8 nœuds.

A Taiwan, le projet national *KING* (Knowledge Innovation National Grid) a été initié, basé sur une amélioration des performances du réseau académique et de recherche. Parmi les applications développées, on note l'étude de l'asthme, l'écologie et la prévision des inondations, en collaboration avec des experts des domaines concernés.

Au Japon, un projet de grille a été lancé sur le réseau *Super-SINet* (Science Information Network), projet qui inclut au départ cinq des principales universités : Kyoto, Nagoya, Osaka, Tohoku, Tokyo. Super-SINet était initialement prévu pour les domaines de la physique des hautes énergies, la bio-informatique et l'astronomie, mais devrait se développer comme un projet de grille nationale pour diverses applications.

I.3.4.3. Europe

Des projets d'infrastructure de grille sont apparus dans nombreux pays d'Europe et la Commission européenne, notamment l'infrastructure EGEE. De nombreuses recherches dans des disciplines très variées sont déjà en cours grâce a cette infrastructure de grille européenne. Citant quelques exemples :

- **Recherche de médicaments**

  Du 11 juillet au 19 août 2005, le projet *WISDOM* a été développé sur l'infrastructure EGEE. Plus de 46 millions de molécules différentes ont été examinées afin de trouver ainsi de nouveaux médicaments contre le paludisme. Un seul ordinateur aurait eu besoin de 80 ans pour réaliser cette tâche. L'informatique en grille a mobilisé simultanément mille ordinateurs de quinze pays afin d'enrayer la deuxième plus grande cause de mortalité dans le monde. Il s'agit du projet biomédical à plus grande intensité de calculs jamais réalisé sur la grille européenne.

- **Analyse des séismes**
  La Terre constitue également un objet de recherche très attirant pour la grille de calcul/données. En utilisant l'infrastructure en grille d'EGEE, les chercheurs de l'Institut de Physique du Globe de Paris (IPGP) sont parvenus à analyser un séisme de grande amplitude durant les 30 heures. Ce tremblement de terre intervenu le 28 mars 2005 en Indonésie a coûté la vie à plus de mille personnes. L'analyse a permis de constater que ce séisme n'était pas une secousse secondaire tardive du tsunami intervenu quelques mois plus tôt, bien qu'il y soit toutefois lié par une relation complexe. En comprenant les paramètres exacts d'un séisme (quand, où et comment) les chercheurs s'approchent du pourquoi d'une secousse. Ce faisant, il sera peut-être possible à l'avenir de prévoir où frapperont les tremblements de terre et quels seront leurs impacts.
- **MAGIC**
  Le télescope MAGIC (Major Atmospheric Gamma Imaging Cherenkov) sur l'île canarienne de La Palma détecte les rayons gamma du cosmos. Son miroir présente un diamètre de 17 m et une superficie de 236 m^2. La grande quantité de rayons gamma captés nous donne des informations sur les événements intervenant dans l'espace. L'analyse des données reçues par le télescope MAGIC et la génération de simulations de gerbes atmosphériques de rayons cosmiques sont réalisées par l'intermédiaire de l'infrastructure en grille européenne.
- **Pico-Inside**
  Ceci est un autre exemple de projet soutenu par la Commission européenne, ne faisant pas usage du middleware développé par le projet EGEE mais bien d'un logiciel commercial de grille fourni par Fujitsu Siemens Europe. Le consortium Pico-Inside rassemble 13 organisations européennes avancées dans le domaine de la nanoscience. Ce projet explore les technologies à l'échelle atomique et pourra entraîner des calculs mathématiques au niveau d'une seule molécule. L'infrastructure du projet comprend le logiciel Fujitsu SynfiniWay qui fournit un service d'interface vers leurs serveurs de calcul et de données distribués. Ceci offre au consortium un accès banalisé aux différents serveurs ainsi que des outils de collaboration avancés, notamment pour la synchronisation des processus de calcul.

I.4. Data Mining distribué

Une étude effectuée en 2003 au sujet du stockage et des flux de données indique que les flux d'information provenant des canaux électroniques, téléphone, radio, TV, et l'Internet (World Wide Web contient environ 170 Terabyte d'informations), contiennent presque 18 exabytes de nouvelle information. D'autre part, chaque année, 800 megabyte d'informations à sauvegarder est produit par personne, dont Quatre-vingt-douze pour cent des nouvelles informations est stocké sur des médias magnétiques, principalement des disques durs.
D'un autre côté, la technologie moderne de l'information produit chaque année des ordinateurs plus puissants, qui nous permettent de collecter, sauvegarder, transférer des énormes quantités de données à prix très réduit. Le progrès de l'acquisition de données

numérique et la technologie de stockage a permis une énorme croissance des bases de données, cette augmentation dans les bases de données a également augmenté les difficultés de l'extraction d'informations valides et potentiellement utiles de ces bases de données, ce qui est la tâche principale du data mining. Quand l'ensemble de données est volumineux la vitesse des taches du data mining est crucial. Les techniques de découverte de connaissances parallèles adressent ce problème par l'utilisation des machines parallèles, la disponibilité progressive de telles machines demande un développement étendu des algorithmes du data mining, pour analyser les ensembles de données mesurés en Terabyte et petabyte sur des machines parallèles avec des centaines ou des milliers de processeurs. A partir de ces contraintes, on peut dire que le datamining distribué (DDM : Distributed Data Mining) se situe à la conjonction de deux évolutions majeures. D'une part, l'explosion de masses de données importantes et souvent réparties dont il faut savoir extraire une connaissance utile, d'autre part le développement sans précédent des réseaux d'ordinateurs permettant le calcul distribué sur des réseaux, locaux comme à grande échelle (les grilles).
Dans un autre terme, Le data mining distribué est le processus du datamining classique qui consiste à extraire une nouvelle connaissance à partir de sources de données partitionnées, distribuées sur différents processeurs, chaque processeur applique un algorithme du data mining sur les données locales .les résultats sont ensuite combinés avec le minimum d'interaction entre les sites de données. La figure I-4 illustre l'architecture générale du data mining distribué.

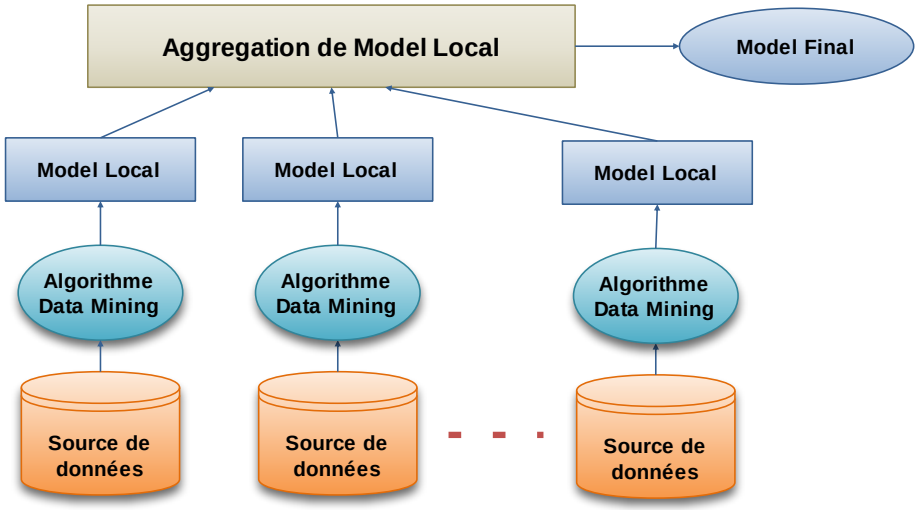

Figure I-4: Architecture générale du data mining distribué

I.4.1. Les différents concepts de parallélisme

La conception parallèle d'un espace s'étend sur un certain nombre de systèmes et de composants algorithmiques, y compris la plate-forme matérielle, le type de parallélisme exploité, la stratégie d'équilibrage de charge, la disposition des données et la procédure de recherche utilisé [23]:

I.4.1.1. Machines à mémoire distribuée(DMM) et systèmes à mémoire partagée(SMP) :

Les objectifs de l'optimisation des performances dépendent de l'architecture sous-jacente. La synchronisation dans DMMs est implicite par le passage de messages, de sorte que le but devient l'optimisation de la communication. Pour les systèmes à mémoire partagée, la synchronisation se fait via les verrous, et l'objectif est de minimiser ces points. La décomposition des données est très importante pour la mémoire distribuée, mais pas pour la mémoire partagée. Bien que les E/S parallèles sont libres dans DMMs, il peut être problématique pour les machines SMP, qui sérialisent généralement les E/S. Le principal défi pour l'obtention de bonnes performances sur DMM est de trouver une bonne décomposition de données entre les nœuds, et à minimiser la communication. Pour SMP, les objectifs sont d'atteindre une bonne localité des données, c.-à-d. optimiser les accès à l'antémémoire local, et d'éviter ou réduire les faux partages, c'est à dire, de minimiser l'effet de ping-pong, où plusieurs processeurs peuvent essayer de modifier les différentes variables qui résident sur la même ligne de mémoire. Aujourd'hui l'accès est non uniforme à la mémoire pour les machines hybrides ou hiérarchique (par exemple, le cluster de SMPs), les paramètres d'optimisation tirent de deux paradigmes à la fois DMM et SMP[24][25].

I.4.1.2. Parallélisme de tâche et parallélisme de données

Ce sont les deux principaux paradigmes pour exploiter le parallélisme des algorithmes. Le parallélisme de données correspond au cas où la base de données est partagée entre les processeurs. Chaque processeur travaille sur sa partition locale de la base de données, mais effectue le même calcul de l'évaluation des candidats et des modèles. Le parallélisme des tâches correspond au cas où les processeurs effectuent des calculs différents de manière indépendante, comme l'évaluation d'un ensemble disjoint de candidats, mais ils ont besoin d'accéder à la base de données entière. Les SMPs ont accès à la base des données entière, mais pour DMMs cela peut être fait via la réplication sélective ou la communication explicite des données locales. Le parallélisme hybride est également possible, il combine les deux à la fois le parallélisme de tâches et de données.

I.4.1.3. Équilibrage de charge Statique et dynamique

Dans l'équilibrage de charge statique les tâches sont d'abord partagées entre les processeurs en utilisant une fonction de coût heuristique, et il n'y a pas de données ultérieures ou des mouvements de calcul pour corriger les déséquilibres de charge qui résultent de la nature dynamique des algorithmes. L'équilibrage de charge dynamique cherche à résoudre ce problème en volant les tâches des processeurs lourdement chargés et réattribuant à ceux faiblement chargés. La circulation de calcul comporte également un mouvement de données, puisque le processeur responsable d'une tâche de calcul a ainsi besoin des données associées à cette tâche. L'équilibrage de charge dynamique expose donc des coûts supplémentaires pour les mouvements des tâches et de données, mais il est bénéfique si le déséquilibre de charge est important et si la charge change avec le temps. L'équilibrage de charge dynamique est particulièrement important dans les environnements multiutilisateurs avec des charges transitoires et dans les plates-formes hétérogènes, qui ont des processeurs et des vitesses de

réseau différents. Ces types d'environnements incluent des serveurs parallèles et méta-clusters hétérogènes. Avec quelques exceptions rares, la plupart des algorithmes du data mining parallèles existants utilisent uniquement une approche d'équilibrage de charge statique qui est inhérente à la répartition initiale de la base de données entre les nœuds disponibles.

I.4.1.4. Fragmentation Horizontal et Vertical des données

La base de données standard pour l'extraction de connaissances est une table relationnelle ayant N lignes, appelés également vecteurs de caractéristiques, transactions, ou enregistrements, et M colonnes, appelés également dimensions, caractéristiques, ou attributs. Le partitionnement des données peut être fait par colonne ou par ligne. De nombreux algorithmes de data mining utilisent un partitionnement horizontale (Figure I-5) de la base de données, où ils stockent comme unité, chaque transaction (tid), ainsi que les valeurs d'attribut pour cette transaction. D'autres algorithmes utilisent un partitionnement verticale (Figure I-6), où ils associent à chaque attribut une liste de tous les tids (appelé *tidlist*) contenant l'item et la valeur d'attribut correspondant à cette transaction. Certains algorithmes fonctionnent plus efficace en utilisant un format horizontal, tandis que d'autres sont plus efficaces en utilisant un format vertical.

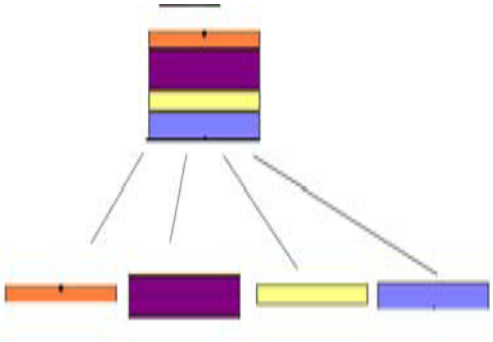

Figure I-5: Fragmentation horizontale

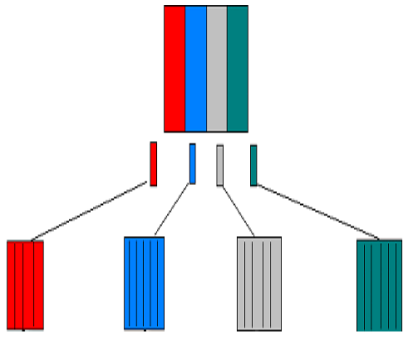

Figure I-6: Fragmentation verticale

I.4.2. Les algorithmes du data mining distribué

Les algorithmes parallèles et distribués pour la découverte de connaissances sont regroupés dans trois classes, les règles d'associations, la classification et le clustering. La plupart de ces algorithmes sont désignés sur le parallélisme potentiel qu'ils peuvent appliquer sur les données distribuées 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 du data mining distribué réside dans l'agrégation et le transfert de données minimal.

I.4.2. 1. Les règles d'association distribuées

L'algorithme Apriori est l'algorithme de base pour la majorité des algorithmes d'extraction de règles d'association parallèles. Apriori utilise un système de recherche complet, avec un format de données horizontale. Généralement les algorithmes d'extraction de règles d'association parallèles appartiennent à deux grandes familles :

- ***Parallélisme de données:*** CD (Count Distribution), FDM (Fast Distributed Mining), DMA (Distributed Mining of Association Rules) ODAM (Optimized Distributed Association Mining).
- ***Parallélisme de tâches :*** DD (Data Distribution), IDD (Intelligent Data Distribution), HPA (Hash Partioned Apriori).

Il existe d'autre algorithme comme HD (Hybrid Distribution), CAD (Candidate Distribution), ECLAT…etc. Ces algorithmes seront détaillés dans le chapitre suivant.

I.4.2. 2. Le Clustering distribué

L'algorithme K-means est une méthode de clustering populaire, la parallélisation de K-means a reçu beaucoup d'attention dans le passé. Différentes méthodes parallèles, en utilisant principalement des ordinateurs hyper cube sont apparus utilisant l'algorithme K-means. Il existe un autre paradigme de clustering parallèle, c'est les algorithmes hiérarchiques agglomératifs qui ont été proposés, en utilisant plusieurs métriques de distance inter-cluster et des architectures parallèles des ordinateurs. Dans le troisième chapitre nous discuterons ces algorithmes en détail.

I.4.2.3. La classification distribuée

Généralement, les techniques les plus connues dans la classification distribuée, les arbres de décision, réseaux de neurones et les algorithmes génétiques.

I.4.2.3.1. Les arbres de décision parallèles

La construction de l'arbre de décision se déroule en deux étapes, la phase de construction où l'arbre est construit à partir de l'ensemble d'apprentissage et une phase d'élagage où l'arbre est de taille réduite pour éviter un sur ajustement avec l'ensemble d'apprentissage. Cependant, la phase de construction prend beaucoup plus de temps de calcul, par conséquent tous les algorithmes de parallélisation concentrent sur la phase de construction de l'arbre. Parmi les algorithmes des arbres de décision parallèles on cite :

- ***SLIQ parallèle :*** L'ensemble de données est partitionnés horizontalement sur les processeurs où chaque processeur est responsable de créer ses propres listes d'attributs. Le problème se pose pour la liste de classe qui doit résider en mémoire et doit être disponible pour des accès et des mises à jour fréquents. L'algorithme doit conserver la liste de classes en mémoire. Ceci a engendré deux approches pour paralléliser SLIQ : Répliquer la liste des classes dans la mémoire de chaque processeur (Sliq/R), ou bien la distribuer à travers tous les processeurs (Sliq/D) [23].

- ***SPRINT parallèle :*** Les listes d'attributs sont distribuées équitablement sur N processeurs. Ceci permet à chaque processeur de travailler seulement sur 1/N de la totalité des données, et de générer ses propres listes d'attributs en parallèle avec les autres processeurs. Les listes d'attributs catégoriques sont équitablement partitionnées et n'exigent aucun autre traitement; celles des attributs numériques sont triés et repartitionnés [23].

- ***Parallel Decision Tree (PDT) :*** c'est un algorithme de parallélisme de données dans une mémoire distribuée, il divise les données d'apprentissage horizontalement dans des blocs de même taille, entre les processeurs. Il suit un paradigme maître-esclave, où le maître construit l'arbre, et trouve les meilleurs points de division. Les esclaves sont responsables de l'envoi de statistiques sur les fréquences de classes pour le maître. Pour les attributs catégoriques, chaque processeur regroupe les fréquences des classes locales, et les transmet au maître. Pour les attributs numériques, chaque processeur trie les valeurs locales, trouve la fréquence des classes pour les points de division, et les échanges avec tous les autres esclaves. Chaque esclave peut alors calculer le meilleur point de division local, qui est envoyé au maître, ce dernier choisit ensuite le meilleur point de division global [26].

- ***ScalParC :*** cet algorithme améliore SPRINT en utilisant une table de hachage distribuée. Ces nouvelles tables de hachage distribuées sont utilisées pour la division d'un nœud, en réduisant la complexité de la communication et la mémoire requises, afin de rendre l'algorithme scalable et sa performance ne se dégrade pas beaucoup avec l'accroissement de la taille de l'ensemble de données [27].

- ***Hybrid Tree Parallelism :*** Il utilise le parallélisme de tâches dynamique (qui existe en différents sous-arbres) combiné avec le parallélisme de données sur les systèmes à mémoire partagée. Initialement, tous les processeurs appartiennent à un groupe, le parallélisme de données est appliqué sur la racine. Une fois les nouveaux nœuds enfants se forment, les processeurs sont également divisés dans des groupes, afin qu'un groupe de nœuds enfants puisse être traitées en parallèle par un groupe de processeurs, Si les nœuds d'arbre associé à un groupe de processeurs deviennent pures (contiennent des exemples d'une classe unique), puis ces processeurs rejoignent un autre groupe actif [28].
- ***The Hybrid Tree Formulation (HTF) :*** est très similaire à l'algorithme précédent, HTF utilise des machines à mémoire distribuées, et donc la redistribution des données est nécessaire dans HTF lors de l'attribution d'un ensemble de nœuds à un groupe de processeurs, de sorte que le groupe de processeurs a tous les fichiers pertinents à un nœud assigné [29].

- ***Shared Tree, Shared Database :*** cette approche utilise la mise en œuvre de mémoire partagée sur la base SPRINT. Elle utilise le parallélisme dynamique de données basée sur les attributs. Les processeurs multiples coopèrent pour construire un arbre de décision partagé d'une façon en largeur. En utilisant un schéma d'ordonnancement dynamique, chaque processeur acquiert un attribut pour n'importe quel nœud d'arbre au niveau actuel,

et évalue les points de division, avant de traiter un autre attribut. Le processeur, qui évalue le dernier attribut d'un nœud de l'arbre, calcule aussi le meilleur point de division pour ce nœud. De même, les listes d'attributs sont partagées entre les fils utilisant le parallélisme des attributs [28].

I.4.2.3.2. Les Réseaux de Neurones Parallèles

Les réseaux de neurones sont des modèles inspirés de la biologie pour le calcul parallèle qui peut être utilisé dans la découverte de connaissances. Les réseaux de neurones supervisés sont utilisés pour la classification et non supervisé pour le clustering. Beaucoup de travaux sur l'implémentation parallèle des réseaux de neurones ont été faits dans le passé. Théoriquement, chaque neurone peut être exécuté en parallèle, mais dans la pratique le nombre de processeurs est généralement plus grand que le nombre de neurones. En outre, le degré d'interconnexion entre processeur est limité en comparaison avec l'interconnexion des neurones. D'où un sous-ensemble de neurones est généralement attribué à chacun des processeurs. Il ya plusieurs façons d'exploiter le parallélisme dans un réseau de neurones [30]:

- Parallélisme entre les sessions d'apprentissage: il est basé sur l'exécution simultanée de différentes sessions d'apprentissage.
- Parallélisme entre des exemples d'apprentissage : chaque processeur forme le même réseau sur un sous-ensemble de 1 / P exemples.
- Parallélisme de couches: chaque couche d'un réseau neuronal est attribuée à un autre processeur.
- Parallélisme de colonnes: les neurones qui appartiennent à une colonne sont exécutées sur des processeurs différents.
- Parallélisme de poids: la sommation des poids pour les connexions de chaque neurone est exécutée en parallèle.

Ces approches parallèles peuvent être combinées pour former différentes stratégies de parallélisation hybrides. Différentes combinaisons peuvent élever des questions différentes pour faire face à une implémentation efficace comme la topologie d'interconnexion, les stratégies d'ordonnancement, l'équilibrage de charge entre les processeurs, et le délai de communication. Les approches de parallélisme typiques utilisées pour l'implémentation des réseaux de neurones sur des architectures parallèles sont le parallélisme des tâches et le parallélisme SPMD (single process, multiple data).

I.4.2.3.3. Les algorithmes génétiques parallèles

Les algorithmes génétiques sont utilisés aujourd'hui pour plusieurs tâches du data mining telles que la classification, les règles d'association. Le Parallélisme peut être exploité en trois phases principales d'un algorithme génétique: initialisation de la population, calcul de forme (fitness), et exécution des opérations de mutation, sans modifier le comportement de l'algorithme par rapport à la version séquentielle. D'autre part, l'exécution parallèle des opérations de sélection et de croisement nécessite la définition de nouvelles stratégies qui

modifient le comportement (et les résultats) d'un algorithme génétique par rapport à la version séquentielle. L'approche la plus utilisée est appelée parallélisation globale. Elle est basée sur l'exécution en parallèle de la fonction fitness et les opérations de mutation pendant que les autres opérations sont exécutées séquentiellement. Cependant, il y a deux variantes possibles :

- Chaque processeur reçoit un sous-ensemble d'éléments et évalue leur fitness en utilisant l'ensemble des données entier D.

- Chaque processeur reçoit un sous-ensemble DIP de l'ensemble de données et évalue la fitness de chaque élément de la population sur son sous-ensemble local.

La parallélisation globale peut être efficace que si un grand ensemble de données doit être traité. Cette approche est simple et a le même comportement de sa version séquentielle, mais sa mise en œuvre n'a pas atteint de très bonnes performances sur des machines à mémoire distribuée, à cause du coût des communications élevé.
Deux stratégies de parallélisation différentes qui peuvent changer le comportement de l'algorithme génétique sont *island model*, où chaque processeur exécute l'algorithme génétique sur un sous-ensemble d'éléments et périodiquement les meilleurs éléments d'une sous-population sont migrés vers les autres processeurs, et *difusion model*, où la population est divisée en un grand nombre de sous-populations composées de quelques individus (où n>>P) qui évoluent en parallèle. Plusieurs sous-ensembles sont placés sur un processeur. Typiquement, les éléments sont arrangés dans une topologie régulière (par exemple, une grille). Chaque élément évolue en parallèle et exécute les opérations de sélection et de croisement avec les éléments voisins.
Une stratégie très simple, c'est l'exécution parallèle indépendante de P copies indépendantes d'un algorithme génétique sur P processeurs. Le résultat final est sélectionné comme le meilleur parmi les P résultats. Des paramètres et des populations initiales différents devraient être utilisés pour chaque copie. Dans cette approche, il n'ya pas de surcharge de communication. L'objectif principal ici n'est pas d'obtenir des performances plus élevées, mais une meilleure précision. Quelques exemples significatifs du data mining basés sur l'exécution parallèle des algorithmes génétiques sont GA-MINEUR, REGAL [31], et G-NET.

I.5. conclusion

Avec l'augmentation des besoins quantitatifs et qualitatifs des systèmes d'aides à la décision, les algorithmes du datamining se sont révélés indispensables et pertinents pour le décideur. Cependant, le nombre de données à analyser, dont le volume ne fait qu'augmenter et la durée de vie ne cesse de diminuer, ainsi que la complexité des algorithmes data mining rendent les traitements impossibles à réaliser sur une machine unique. Quoique, le calcul parallèle de ces algorithmes peut résoudre ce problème, particulièrement sur les grilles qui définissent une très grande puissance de calcul et une capacité de stockage illimitée, d'où la naissance du data mining distribué. Parmi les algorithmes les plus utilisés en data Mining ; les règles d'association, ces dernières permettent d'extraire des relations entre les données, qui

représentent des informations intéressantes et utiles et qui sont cachées dans ces grandes masses de données. Nous présenterons dans le chapitre suivant, les algorithmes les plus connus pour l'extraction de règles d'association dans les environnements distribués.

CHAPITRE II : LES RÈGLES D'ASSOCIATION DISTRIBUÉES

II.1. introduction

Les données stockées peuvent cacher un certain nombre de connaissances, de dépendances ou de corrélations, qui sont implicites et utiles, et qui n'attendent qu'à être explorées. Dans ce contexte, les règles d'association ont sollicité l'intérêt de plusieurs chercheurs vu leur utilité dans plusieurs secteurs tels que le management des entreprises, le marketing, la biologie, la robotique, etc. Afin de les extraire des données en entrée, organisées sous forme d'une base de transactions, deux étapes sont nécessaires à savoir, l'extraction des ensembles d'items (ou itemsets) fréquents et la génération des règles d'association à partir de ces ensembles d'items. La première étape est la plus couteuse, elle demande un cout exponentiel de calcul pour un grand volume de données. Par conséquence, les algorithmes distribués sont basés sur la parallélisation de la phase d'extraction des itemsets fréquents.
Nous présentons dans ce chapitre quelques définitions de base pour l'extraction de règles associatives. Ensuite, nous détaillons les différents algorithmes pour l'extraction distribuée de règles d'association. Ces algorithmes se divisent en deux grandes familles, parallélisme de données et parallélisme de tâche. Nous discutons ensuite leurs performances et limites.

II.2. les règles associatives

Une des principales applications de la recherche de règles d'association appartient au secteur de la distribution dans le cadre de « l'analyse du panier de la ménagère », c'est-à-dire la recherche d'association entre produit sur les tickets de caisse (on recherche des produits qui tendent à être achetés ensemble). À 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. Bien que cette application soit la plus répandue, la recherche de règles d'association peut s'appliquer à bien d'autres secteurs d'activités, banques, communications, télécommunications, secteur médical, l'analyse d'images et de données spatiales, etc.

Une règle d'association est une règle de la forme : $A \rightarrow B$, où A et B sont des conjonctions d'attributs de la base de données. *A* représente *l'antécédent* de la règle et *B* la *conclusion*. 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 :

$$(\text{Age}>25) \wedge (\text{Revenu}>5000) \rightarrow (\text{modèle_voiture=tourisme})$$

Cette règle associe l'âge et le revenu d'un individu au type de la voiture qu'il peut posséder.

II.2.1. Définitions

- Dans le problème de recherche de règles d'association, on suppose que la base de données est composée d'un ensemble d'instances et d'un ensemble d'attributs appelés *items*.

- On appelle *itemset* un ensemble d'items, le nombre d'items d'un itemset constitue sa longueur, un itemset contenant k items est appelé un k-itemset. Chaque instance d'une base de données est un itemset.

- La recherche de règles d'association consiste à produire des règles de dépendance, des relations entre les items d'une base de données.

- Soit $I = \{i_1, i_2, \ldots, i_m\}$ un ensemble de m items et soit D un ensemble d'instances constitué des items (i_i ,i_j , i_k de I).

- Une règle d'association est une implication de la forme X=>Y où X et Y sont inclus dans I et $X \cap Y = \emptyset$. X est appelé la condition ou l'antécédent et Y la conclusion ou la conséquence. Deux mesures sont définies pour une règle associative : le support et la confiance.

- La recherche de règles s'appuie sur des mesures statistiques qui permettent de générer les règles les plus pertinentes et de limiter l'espace de recherche :

    - *Le support :* le support d'un itemset est le pourcentage d'instances de D qui contiennent l'itemset. Le support permet de mesurer l'intérêt de l'itemset, c.à.d. sa fréquence dans le jeu de données. Le support d'une règle est le rapport de nombre d'instances pour les quelles à la fois la condition et la conclusion apparaissent sur le nombre total d'instances.

    - *La confiance :* la confiance d'une règle est le rapport de nombre d'instances pour les quelles à la fois la condition et la conclusion apparaissent sur le nombre d'instances pour les quelles au moins la condition apparaisse [32].

- De nombreuses autres mesures d'intérêt des règles existent, la conviction, le lift, le coefficient de corrélation, la JMeasure.

- La recherche de règles d'association consiste à trouver les règles de support et de confiance supérieurs à certains seuils fixés au départ.

- L'extraction de règles d'association est décomposée en deux sous-problèmes :
    - La recherche des ensembles fréquents d'items (les itemsets fréquents).
    - La génération des règles d'association à partir de ces itemsets fréquents.

II.2.2. La recherche des itemsets fréquents

Elle consiste à parcourir itérativement l'ensemble des itemsets. Durant chaque itération, un ensemble d'itemsets candidats est créé. Les supports de ces itemsets sont calculés et les itemsets non fréquents sont supprimés.

Cette étape est couteuse en termes de temps d'exécution car le nombre d'itemsets fréquents dépend exponentiellement du nombre d'items manipulés, pour n items on a 2^n itemsets potentiellement fréquents. De plus, le nombre d'itemsets fréquents est faible par rapport au nombre total d'itemsets. Il convient donc d'utiliser des approches plus complexes, on peut distinguer deux types d'algorithme [33]:

- Des algorithmes qui proposent une réduction de l'espace de recherche (par exemple l'algorithme de base Apriori).
- Des algorithmes qui proposent une réduction du nombre de balayages nécessaires du jeu de données (par exemple l'algorithme DIC).

II.2.3. La génération des règles d'association

Elle utilise l'ensemble d'itemsets fréquents pour déduire les règles recherchées. Pour chacun des itemsets fréquents obtenus, on va rechercher les règles d'association qui possèdent une confiance supérieure au seuil minimal de confiance fourni par l'utilisateur.

On regarde les différentes combinaisons condition-conclusion possibles pour chaque itemset. On garde la combinaison la plus pertinente celle qui répond au critère de seuil minimal de confiance.

A partir d'un k-itemset F_k fréquent, on cherche les règles de type : $F_i \rightarrow F_j$ de support et de confiance suffisants. F_i et F_j sont des itemsets de tailles respectives i et j, tels que : $i+j = k$, $F_i \subset F_k$, $F_j \subset F_k$ et $F_i \cap F_j = \emptyset$.

Par exemple, si ABCD et AB sont des itemsets fréquents, alors on peut générer la règle AB=>CD. Pour savoir si cette règle est à retenir ou à rejeter, on calcule la confiance de AB=>CD qui n'est autre que la relation : conf = sup (ABCD) / sup (AB). Si conf ≥ minconf, alors la règle est retenue [33].

II.3. Les algorithmes d'extraction de règles d'association séquentiels

L'algorithme le plus connu pour la recherche de règles d'association est l'algorithme Apriori [13]. La plupart des algorithmes existants se base sur Apriori, ce dernier procède de façon itérative. Lors de la Keme itération, on recherche les K-itemsets fréquents, c'est-à-dire les itemsets de longueur K qui sont fréquents. A l'itération suivante, la (K+1)ème, on recherchera les (K+1)-itemsets fréquents et ainsi de suite.

II.3.1. L'algorithme Apriori

Dans la première passe de cet algorithme (voir figure II-1 et figure II-2), on calcule le support des 1-itemsets puis on détermine ceux qui sont fréquents. A partir de cela, chaque passe ultérieur utilisera la liste des itemsets trouvés dans la passe précédente, pour générer les nouveaux itemsets fréquents, appelés itemsets candidats. En même temps que cette phase de génération, on calcule les supports actuels pour ces itemsets candidats. 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 puisse plus avoir de nouveaux itemsets.

L'idée de bases est que chaque super-ensemble d'un itemsets non fréquent est obligatoirement non fréquent. Par conséquent, Les itemsets candidats ayant k item, peuvent être générés en joignant les itemsets fréquents ayant (K-1) item, et en supprimant ceux qui ont un sous-ensemble non fréquent.

```
Algorithme Apriori
Debut
L1={Large 1-itemsets};
  Pour (k=2; Lk-1 ≠∅; k ++) faire
      /*Génération des Itemsets candidats*/
      Apriori-Gen(Lk-1)
      Ck={Ensemble des Itemsets candidats}
      /*calcul du support*/
       Pour toute transaction t∈D faire
         Pour tout k-sous-ensemble s de t faire
            si s ∈ Ck alors s.count++;FinSi
         Finpour
   /*Génération des Itemsets larges*/
   Lk={c∈ Ck / c.count ≥ minsup};
      Finpour
  Finpour
     Retourner ( Uk Lk = {Ensemble des Itemsets larges} )
Fin
```

Figure II-1 : L'algorithme Apriori

```
Fonction Apriori-Gen
/*Génération des Itemsets larges*/
  Pour chaque Itemsets L1 ∈ Lk-1 faire
     Pour chaque Itemsets L2 <> L1 de Lk-1
faire
        Si (préfixe (L1)= préfixe (L2)) alors
           Ck={ Ck U(c= L1 U L2)}}
        FinSi
     Finpour
  Finpour
```

Figure II-2: la procédure Apriori-Gen

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 et la confiance d'une règle :

- Le support permet de mesurer la fréquence de l'association :

Support=freq(condition et résultat) = d/m

Où d est le nombre de transactions où l'antécédent et la conclusion apparaissent et m le nombre total de transactions.

- La confiance permet de mesurer la force de l'association :
Confiance= freq(condition et résultat)/freq(condition) = d/c
Où c est le nombre de transactions où l'antécédent apparaît [2].
Donc à partir d'un ensemble de transactions D, l'intérêt est de découvrir les règles d'association qui satisfassent un niveau de support et de confiance minimum, notés, minsup et minconf, spécifiés par l'utilisateur.
Plusieurs 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 sont proposées dans les algorithmes suivants.

II.3.2. DHP (Direct Hashing and Pruning)

DHP [34] est une légère modification de Apriori qui utilise des tables de hachage pour diminuer les nombre d'itemsets candidats générés. Cet algorithme propose également une diminution de taille de la base de données, au fur et à mesure des itérations.
Lors de deux premières itérations (gestion des 1-itemsets et des 2-itemsets), on utilise une table de hachage sur les numéros des itemsets (c.à.d. associer un numéro à chacun des itemsets possible pour l'itération concernée). A chaque case de la table de hachage, on associe un compteur. A chaque fois qu'une instance contient l'un des itemsets associés (par la fonction de hachage) à une case de la table de hachage, on incrémente le compteur de cette case.
On garde les itemsets associés aux cases dont le compteur est suffisant. Par exemple, pour une case associée à 3 itemsets, on garde les itemsets si le compteur associé à la case possède une valeur supérieure à trois fois le seuil de support fixé.
A partir de la deuxième itération. On effectue un pruning des instances de la base. Pour chaque instance, si le nombre de bits de cette instance est supérieur à k, l'instance peut être utile à la k^{eme} itération, si non, on la supprime.

II.3.3. AprioriTid

Cet algorithme [35] utilise des listes d'itemsets associées aux identifiants des objets en relation avec les itemsets, afin de diminuer progressivement la taille du jeu de données. Il permet de diminuer progressivement la taille de la base, dans le but de la stocker en mémoire et de ne plus réaliser d'opérations d'entrée-sortie, après le premier balayage.
A partir de la deuxième itération, la procédure Apriopri-Gen est utilisée pour générer les itemsets candidats de l'itération suivante dans un ensemble $\bar{C}_k$. chaque élément de l'ensemble $\bar{C}_k$ est un couple (ID, c_k) dans lequel c_k est la liste des k-itemsets candidats contenus dans l'instance d'identifiant ID. le support d'un itemset candidat c_k correspond au nombre d'apparitions de c_k dans $\bar{C}_k$.

II.3.4. Partition

Cet algorithme [14]ne nécessite que deux balayage du jeu de données. Durant le premier balayage, la base est divisée en n partitions disjointes qui seront considérées une à une successivement. Pour chaque partition, l'ensemble des itemsets fréquents dans la partition

(les itemsets fréquents locaux) sont extraits. Les ensembles d'itemsets fréquents de chaque partition sont ensuite fusionnés pour obtenir un ensemble d'itemsets candidats à être fréquent sur la base entière. On a ainsi un sur-ensemble de l'ensemble des itemsets fréquents. Durant le second balayage les supports pour ces candidats sont calculés sur toute la base et les itemsets fréquents sont identifiés.
La taille de la partition est choisie de telle façon que chaque partition tienne en mémoire.

II.3.5. Eclat/ MaxEclat/ clique / MaxClique

Ces algorithmes [36]sont basés sur les classes d'équivalence. Chacun des quatre algorithmes implique les mêmes étapes. Premièrement, il calcule les items fréquents à partir de la base de données verticale tid-liste, ensuite calcule les 2-itemsets. Après génération des sous-treillis, ces derniers sont alors traités pour produire des itemsets fréquents maximaux.

II.3.6. Sampling

Cet algorithme [37] effectue la génération des itemsets sur un échantillon de la base. Leurs supports sont ensuite calculés sur la base entière. Au fur et à mesure du parcours de la base. On affine les résultats (la valeur des supports). On supprime des itemsets qui se révèlent non fréquents.
Cet algorithme suppose disposer d'un échantillon représentatif de la base.

II.3.7. DIC (Dynamic Itemsets Counting)

Cet algorithme [38] propose une diminution du nombre de balayages nécessaires du jeu de données. L'algorithme DIC est basé sur une méthode similaire à Apriori. Le jeu de données est décomposé en sous-ensembles de taille donnée, et durant chaque itération, un sous-contexte (un sous-ensemble d'instances) est lu. Ainsi, plusieurs ensembles d'itemsets candidats de tailles différentes sont traités simultanément durant chaque itération. Ce qui permet de diminuer le nombre total de balayages du contexte.
L'algorithme utilise un principe de fenêtrage de la base (on forme des blocs de M instances). On affecte un type aux itemsets : fréquent confirmé, fréquent potentiel, infréquent confirmé, infréquent potentiel.
Le parcours du premier bloc permet d'estimer les fréquences des 1-itemsets. Lorsqu'on parcourt tout ce premier bloc, on génère les 2-itemsets candidats à partir des estimations de fréquences réalisées pour les 1-itemsets. Le parcours du deuxième bloc permet d'améliorer les estimations de fréquences des 1-itemsets et de débuter l'estimation des fréquences des 2-itemsets et ainsi de suite.

II.3.8. FP-Growth (Frequent-Pattern Growth)

C'est un algorithme [39] complètement innovant par rapport aux autres algorithmes de recherche de règles associatives presque tous basés sur Apriori. L'algorithme utilise une structure de données compacte appelé *Frequent-Pattern tree* et qui apporte une solution au problème de la fouille de motifs fréquents dans une grande base de données transactionnelle.

En stockant l'ensemble des éléments fréquents de la base de transactions dans une structure compacte, on supprimer la nécessiter de devoir scanner de façon répétée la base de transactions. De plus, en triant les éléments dans la structure compacte, on accélère la recherche des motifs.
Un FP-tree est composé d'une racine nulle et d'un ensemble de nœuds préfixé par l'élément représenté. Un nœud est composé par : le nom de l'élément, le nombre d'occurrence de transaction où figure la portion de chemin jusqu'à ce nœud, un lien inter-nœud vers les autres occurrences du même élément figurant dans d'autres séquences de transactions. Une table d'en-tête pointe sur la première occurrence de chaque élément. L'avantage de cette représentation des données est qu'il suffit de suivre les liens inter-nœuds pour connaître toutes les associations fréquentes où figure l'élément fréquent.

Afin de limiter le nombre de candidats considérés à chaque itération, l'algorithme Apriori (et ses dérivés) se base sur les deux propriétés suivantes :
Propriété1 : Tous les sous-ensembles d'un itemset fréquent sont fréquents. Cette propriétés permet une limitation du nombre de candidats générés lors de la K^{eme} itération par une jointure conditionnelle des itemsets fréquents, de taille K-1, découvert précédemment.
Propriété2 : Tous les sur-ensembles d'un itemset infréquent sont non fréquents .Cette propriété permet la suppression d'un candidat de taille K lorsqu' au moins un de ses sous-ensembles de taille K-1 ne fait pas partie des itemsets fréquents découvert précédemment.

II.4. Les algorithmes d'extraction de règles d'association parallèles et distribués

Les techniques de découverte de règle d'association ont été graduellement adaptées aux systèmes parallèles afin d'améliorer le temps de réponse et tirer profit de la capacité de stockage étendue qu'offrent les systèmes distribués et parallèles tel que les grilles. La transition à un système de mémoire réparti exige la division de la base de données entre les processeurs. Différentes approches ont été proposées, elles se divisent en deux catégories, parallélisme de données et parallélisme de tâches. Ces parallélisations nécessitent une nouvelle propriété en plus des deux propriétés vues précédemment dans l'extraction séquentielle des règles d'association :

Propriété3 : Pour qu'un itemset soit globalement fréquent il faut qu'il soit localement fréquent sur au moins un site.

II.4.1. Parallélisme de données

Dans ce type de paradigme, chaque nœud dénombre le même ensemble de candidats. Les ensembles sont dupliqués sur tous les processeurs et la base de données est distribuée à travers ces processeurs. Chaque processeur est responsable du calcul des supports locaux de tous les candidats qui sont des supports dans sa propre partition de base de données. Tous les processeurs calculent ensuite les supports globaux des candidats qui sont les supports totaux des candidats dans la base de données entière par échange des supports locaux (la réduction globale).Par la suite, les itemsets fréquents sont calculés par chaque processeur

indépendamment. Parmi ces algorithmes, on peut citer Count Distribution, Fast Distributed Mining…etc

II.4.1.1. Count Distribution (CD)

Proposé par R.Agrawal et J.C Shafer en 1996, cet algorithme [40] est basé sur l'algorithme Apriori et sur le parallélisme de données, il est réalisé dans un environnement distribué où chaque site traite sa portion locale de la base de transactions, calcule les supports locaux des candidats et les diffuse aux autres sites pour calculer le support global, le principe de cet algorithme est présenté dans la figure suivante :

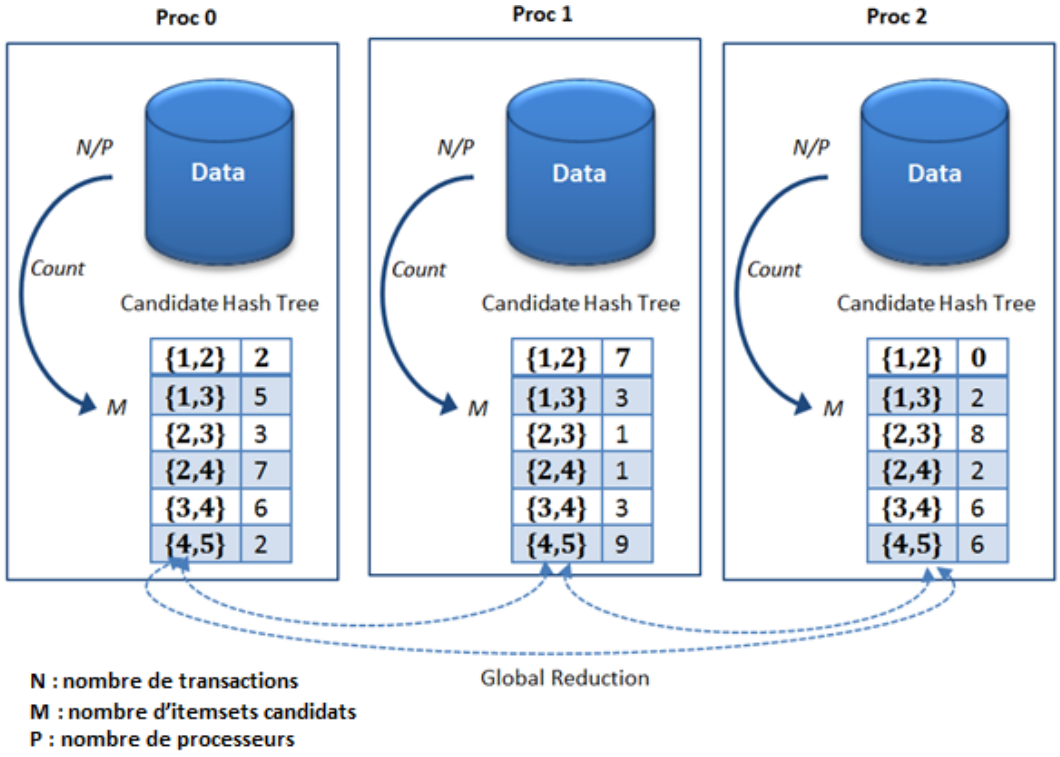

Figure II-3 : Illustration de l'Algorithme Count Distribution

L'algorithme CD est décrit comme suit :

Input: I, s, D_1, D_2… D_p
Output: L
Algorithm:
C_1=I;
for k=1; $C_k \neq \emptyset$; k++ **do** **begin**
//step one: counting to get the local counts

count (C_k, D_i); //local processor is I

//step two: exchanging the local counts with other processors to obtain the global counts in the whole database.

for all itemset $X \in C_k$ **do begin**

$X.count=\sum_{j=1}^{p}\{X_j.count\}$;

end

//step three: identifying the large itemsets and generating the candidates of size k+1

$L_k=\{c \in C_k \mid c.count \geq s \times |D_1 \cup D_2 \cup \ldots \cup D_p|\}$;

C_{k+1} =apriori_gen(L_k);

end

return $L=L_1 \cup L_2 \ldots \cup L_k$;

Figure II-4 : L'algorithme Count Distribution

Dans la première passe, chaque processeur P_i génère son ensemble d'itemsets candidats C_1^i à partir des items présents dans sa base de données locale D_i. Par conséquent, les candidats générés par différents processeurs ne peuvent pas être identiques et doivent être pris lors d'échange des supports locaux pour déterminer l'ensemble globale C_1.

La première passe est spéciale, pour les autres passes l'algorithme fonctionne comme suit :

1. Chaque processeurs P_i génère l'ensemble des candidats C_k en utilisant l'ensemble des itemsets fréquents (k-1) crées à la fin de l'étape (k-1).Tous les processeurs ont un ensemble identique d'itemsets fréquents L_{k-1} et génèrent le même ensemble de candidats C_k.
2. Chaque processeur P_i peut donc indépendamment obtenir le support local des candidats C_k à partir de sa base de données locale D_i.
3. L'algorithme effectue la somme pour obtenir le support global. Chaque processeur P_i diffuse les supports locaux de C_k aux autres processeurs pour développer le support global C_k. Les processeurs sont contraints de se synchroniser dans cette étape.
4. Chaque processeur P_i calcule maintenant les items fréquents L_k à partir de l'ensemble des candidats C_k en gardant seulement ceux ayant un support supérieur au support minimal.

L'exemple suivant montre le fonctionnement de l'algorithme *Count Distribution* sur trois processeurs, dont chaque processeur calcule les supports locaux de tous les itemsets et les envoie aux autres processeurs. Après la sommation des supports locaux dans chaque processeur, les itemsets non fréquents globalement tels que, ab, bc, cd , df, ade et bde sont supprimés :

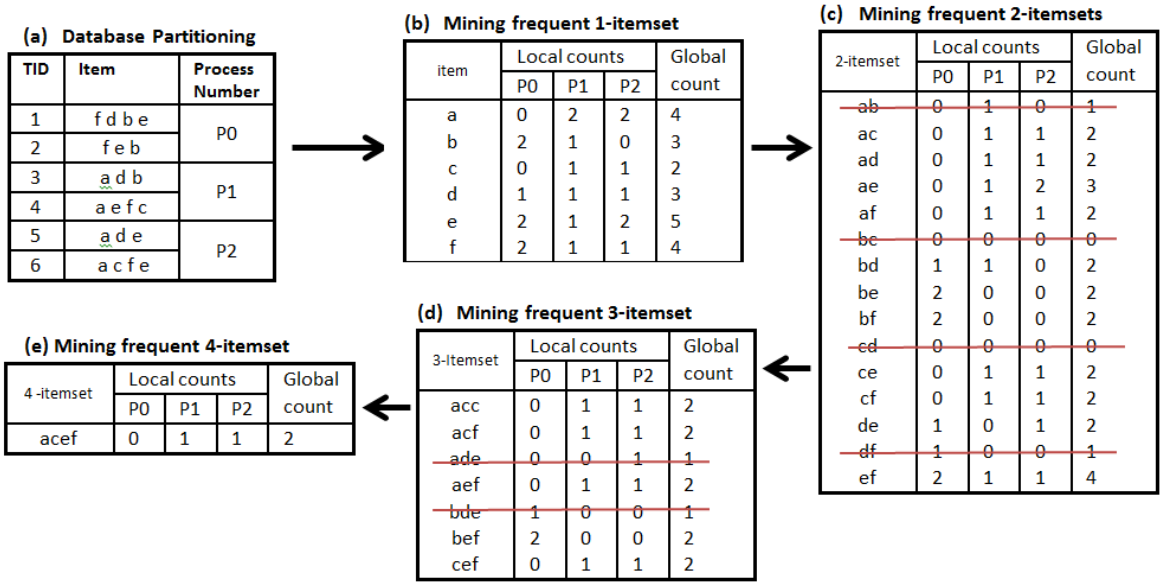

Figure II-5 : *Exemple de l'Algorithme Count Distribution*

II.4.1.2. Fast Distributed Mining (FDM)

Cet algorithme [41] est basé sur l'algorithme Apriori, il a été proposé par Daving Cheung pour l'extraction des règles d'association. Ce dernier propose ainsi de nouvelles techniques pour réduire le nombre des candidats considérés pour le calcul du support.

Après avoir généré l'ensemble global de candidats $CG_{(k)}$, il faut échanger les supports de cet ensemble entre les sites pour trouver les itemsets globalement fréquents. Toutefois, il est observé que certains ensembles de candidats dans $CG_{(k)}$ peuvent être élagués en utilisant des informations locales avant que l'échange des supports commence. L'élagage local consiste alors à supprimer l'élément X de l'ensemble de candidats sur le site S_i si X n'est pas localement fréquent par rapport à sa base de données locale.
Une fois que l'élagage local est effectué, chaque site diffuse les supports des candidats restant aux autres sites. A la fin de chaque itération chaque site somme ces supports locaux pour générer les itemsets globalement fréquents par rapport à la base de données globale, cette technique est appelée l'élagage global. L'algorithme FDM minimise ainsi l'ensemble des candidats générés en se basant sur ces deux techniques.

II.4.1.3. Distributed Mining of Association rules (DMA)

L'algorithme DMA [42] est une parallélisation de l'algorithme Apriori. Il utilise une technique d'élagage au niveau de chaque site et ne garde que les itemsets fréquents. Les supports locaux des itemsets fréquents sur chaque site sont employés pour décider si un itemset fréquent est lourd (localement fréquent dans une partition et globalement fréquent dans la base de données entière).
DMA introduit une nouvelle technique d'optimisation de la communication afin de réduire la taille des messages et éviter leur duplication. Au lieu de diffuser les supports locaux de tous

les candidats comme dans le cas de CD, DMA associe à chaque itemset un site de vote qui est le responsable de collecte de ses supports locaux.

Dans cet algorithme, pour chaque candidat X, son site de vote (polling site) est responsable d'envoyer les demandes (polling request) pour collecter les supports locaux, et déterminer si le candidat X est fréquent. Puisque il y a un seul site de vote pour chaque candidat X, le nombre de messages échangés pour calculer le support global de X est n.
Dans la k^{eme} itération, après la fin de la phase d'élagage local dans le site S^i, DMA utilise les procédures suivantes pour faire le vote :

- **Candidats envoyés à leurs sites de vote :** soit S^i le site référence (home site) de son ensemble de candidats, pour chaque site de vote (polling site) S^j, S^i trouve tous les candidats dans LL_k^i dont le site de vote est S^j et les stocke dans $LL_k^{i.j}$ (les candidats sont divisés dans des groupes selon leurs sites de vote), les supports locaux des candidats sont ainsi stockés dans $LL_k^{i.j}$. Ensuite, le site S^i envoie chaque $LL_k^{i.j}$ aux sites de vote correspondants.
- **Site de vote envoie les demandes de vote (polling request) :** soit S^i le site de vote, S^i reçois tous les $LL_k^{j.i}$ envoyés par les autre sites, pour chaque candidat X reçu, S^i recherche dans la liste les sites d'origines d'où X est envoyé. S^i alors envoi les demandes de vote aux autres sites qui ne figurent pas dans la liste pour collecter les supports locaux.
- **Site distant répond les demandes de vote :** soit S^i un site distant qui répond au demandes de vote reçus. Pour chaque demande de vote $LL_k^{p.i}$ envoyé par le site de vote S^p, S^i envoie les supports locaux des candidats dans $LL_k^{p.i}$ à S^p (ça nécessite pas un balayage de la partition D^i, car les supports sont calculés dans la phase d'élagage local).
- **Site de vote calcule le itemsets fréquents (lourds) :** soit S^i le site de vote qui calcule les itemsets lourds ou fréquents. S^i reçois les supports envoyés par les autres sites, calcule le support globale de leurs candidats dans LL_k^i et détermine les itemsets fréquents, éventuellement, S^i envoie les itemsets fréquents trouvés avec leurs supports globale à tous les sites.

II.4.1.4. Optimized Distributed Association Mining (ODAM)

ODAM [43] suit le paradigme de CD et de FDM. Il partitionne la base de données horizontalement sur les sites distribués.
ODAM calcule d'abord les supports des 1-itemsets sur chaque site de la même manière que l'algorithme séquentiel Apriori, il diffuse ensuite cet ensemble d'itemsets aux autres sites et identifie les 1-itemsets globalement fréquent. Par la suite, chaque site génère les 2-itemsets candidats à partir de cet ensemble et calcule leurs supports. En même temps, ODAM élimine tous les 1-itemsets non fréquents globalement de chaque transaction et insère la nouvelle transaction dans la mémoire .Après la génération des supports des 2-itemsets candidats à chaque site, ODAM génère les 2-itemsets globalement fréquents. Il recommence alors sur les nouvelles transactions en mémoire principale et il génère les supports des itemsets candidats

de longueur successive puis les itemsets fréquents globaux correspondant cette longueur en diffusant les supports des itemsets candidat après chaque passage.
ODAM propose une optimisation dans le schéma de communication en réduisant au minimum les échanges de messages en envoyant des comptes de support d'itemsets des candidats à un emplacement simple appelé le récepteur. Le récepteur annonce les itemsets globalement fréquent de nouveau aux emplacements distribués.

Pour calculer efficacement les supports des candidats générés, ODAM élimine tous les items non fréquents après la première itération et stocke les nouvelles transactions dans la mémoire. Cette technique permet non seulement de réduire la taille moyenne des transactions mais également réduit la taille de l'ensemble de données de manière significative, ainsi on peut accumuler plus de transactions dans la mémoire. Le nombre d'items dans l'ensemble de données pourrait être grand, mais seulement quelques-uns satisferont le seuil de support. En plus, le nombre des itemsets non fréquents augmente proportionnellement pour un seuil de support plus élevé.

II.4.2. Parallélisme de taches

Dans ce type de paradigme, l'ensemble de candidats est partitionné et distribué à travers les processeurs ainsi que la base de données. Chaque processeur est responsable de maintenir seulement les supports globaux d'un sous-ensemble de candidats. Cette approche exige deux passages de communication dans chaque itération. Dans le premier, chaque processeur envoie sa partition de la base de données à tous les autres processeurs. Dans le second passage, chaque processeur diffuse les itemsets fréquents qu'il a trouvés à tous les autres processeurs pour calculer les candidats de l'itération suivante. Plusieurs variantes du parallélisme de tache ont été proposées, elles différent dans la façon de diviser la base de données et les candidats :

II.4.2.1. Data Distribution (DD)

Il s'agit encore d'un algorithme [40] issu de l'algorithme Apriori et proposé par *R. Agrawal* et *J.C. Shafer* en 1996. Il diffère de Count Distribution dans le sens où les processeurs se partagent le travail non plus par portion de base de données mais par candidats. Les supports des itemsets sont toujours calculés de façon locale, à chaque itération. Puis, entre deux itérations on effectue une diffusion des partitions locales de données. La figure suivante montre le principe de l'algorithme DD.

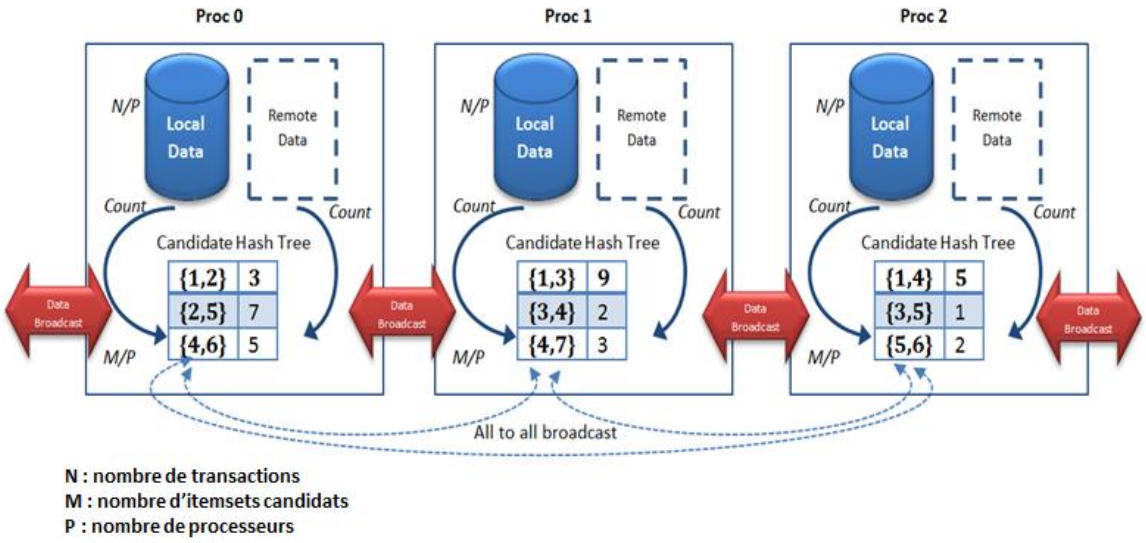

Figure II-6 : Illustration de l'Algorithme Data Distribution

Dans cet algorithme DD, chaque processeur est responsable de calculer les supports d'un ensemble de candidats qui est différent des ensembles de candidats assignés aux autres processeurs. De plus, chaque processeur envoie sa partition de base de données aux autres processors dans chaque itération. La première passe de Data Distribution est identique de celle présentée dans l'algorithme Count Distribution, les autres passes fonctionnent comme suit :

1. Le processeur P^i génère C_k (itemsets candidats) à partir de L_{k-1}(itemsets fréquents). Il maintient seulement 1/P des itemsets candidats, C_k^i est l'ensemble qu'il traitera. Les ensembles C_k^i sont tous disjoints et la jointure de tous les ensembles C_k^i donne l'ensemble C_k.
2. Le processeur P^i calcul les supports des itemsets dans son ensemble local de candidats C_k^i en parcourant la base de données locale et les partitions de données envoyées par les autres processeurs.
3. A la fin du parcours de la base de données, chaque processeur P^i détermine L_k^i en utilisant l'ensemble C_k^i. Encore, tous les ensembles L_k^i sont disjoints et l'union de tous les ensembles L_k^i donne L_k.
4. Les processeurs échangent L_k^i pour que chaque processeur ait l'ensemble complet L_k, afin de générer C_{k+1} pour l'itération suivante. Cette étape demande la synchronisation entre les processeurs. Après avoir obtenu l'ensemble L_k, chaque processeur peut indépendamment (car c'est identique) décider si c'est terminé ou continuer à l'itération suivante.

II.4.2.2. Candidate Distribution (CandD)

La limitation des algorithmes Count Distribution et Data Distribution est que chaque transaction doit être comparée avec tous les candidats. Ce qui demande un calcul dupliqué des candidats dans chaque processeur et échange des transactions de toute la base de données. En

plus, les deux algorithmes CD et DD exigent la synchronisation entre les processeurs à la fin de chaque itération pour échanger les supports et les itemsets fréquents. Si la charge de travail n'est pas parfaitement équilibrée, cela peut entrainer que tous les processeurs attendent celui qui termine le dernier dans chaque itération. Tous les processeurs doivent être consultés et toutes les informations doivent être récoltées avant de procéder à l'itération suivante.

L'algorithme Candidate Distribution [40] tente de travailler loin de ces dépendances requis dans les algorithmes CD et DD de façon à ce que chaque processeur travail indépendamment. A une certaine itération L, d'où L est déterminé, cet algorithme divise les itemsets fréquents L_{l-1} entre les processeurs de façon à ce que chaque processeur P^i peut générer uniquement les itemsets candidats C_m^i (m ≥ L), indépendamment de tous les processeurs ($C_m^i \cap C_m^j = \emptyset$, i≠j). Au même temps, les données sont repartitionnées afin qu'un processeur peut compter les supports des candidats dans C_m^i indépendamment des autres processeurs. Notons que selon la qualité du partitionnement des itemsets, les portions de base de données peuvent être dupliquées sur plusieurs processeurs. L'algorithme de partitionnement des itemsets prend en considération ces aspects par l'identification des segments de L_{l-1} qui sont probablement supportés par des transactions différentes de base de données. Le choix de l'itération de repartitionnement est fait en fonction de la différence entre le découplage de la dépendance des processeurs dés que possible et l'attente jusqu'à ce que les itemsets soient plus facilement et équitablement partitionnables. L'algorithme de partitionnement utilise la sémantique de la procédure de génération des candidats de l'algorithme Apriori.

Après la distribution des candidats, chaque processeur travaille indépendamment, en calculant seulement sa portion de l'ensemble de candidats globale sur la base de données locale. Aucun échange de supports ou de données n'est requiert. La seule dépendance entre les processeurs, est l'élagage de l'ensemble de candidats locaux durant l'étape de l'élagage des candidats générés. Cependant, cette information est envoyée sans synchronisation, et les processeurs ne doivent pas attendre l'information complète de l'élagage qui arrive de tous les autres processeurs. Pendant l'étape de l'élagage des candidats générés, le processeur supprime des candidats autant que possible en utilisant les informations arrivées, et commence le calcul des supports de l'ensemble de candidats. L'information de l'élagage arrivée en retard peut être utilisée dans l'itération suivante.

II.4.2.3. Intelligent Data Distribution (IDD)

Cet algorithme [44] a été proposé pour résoudre les problèmes de l'algorithme Data Distribution. Premièrement, Il adopte une architecture en anneau pour améliorer les performances et réduire les couts de communication, c'est-à-dire qu'il emploie la communication asynchrone de point en point entre les voisins dans l'anneau au lieu du broadcast.

Deuxièment, IDD résoud le problème du travail redondant dans DD par le partitionnement intelligent de l'ensemble de candidats. Afin d'éliminer le calcul redondant dû au partitionnement des itemsets, IDD trouve une manière rapide pour vérifier si les transactions données peuvent potentiellement contenir les candidats stockés dans chaque processeur. Ceci

n'est pas possible par le partitionnement aléatoire de l'ensemble de candidats global C_k. IDD partitionne les candidats à travers les processeurs en se basant sur le premier item des candidats, c'est-à-dire les candidats avec le même item préfix (premier item) seront mis dans la même partition. Donc, avant le traitement d'une transaction chaque processeur doit s'assurer qu'elle contient les préfixes appropriés qui sont assignés à ce processeur. Sinon, la transaction peut être rejetée.

La base de données entière est toujours communiquée, mais une transaction ne pourrait pas être traitée si elle ne contient pas les itemsets appropriés. Ce qui réduit le calcul redondant dans DD (Dans DD, chaque processeur doit vérifier tous les sous-ensembles de chaque transaction).Pour réaliser un équilibre de charge dans la distribution des candidats, il calcule d'abord pour chaque item le nombre de candidats commençant par cet item et il assigne les itemsets aux partitions des candidats de telle sorte que le nombre de candidats dans chaque partition soit le même.

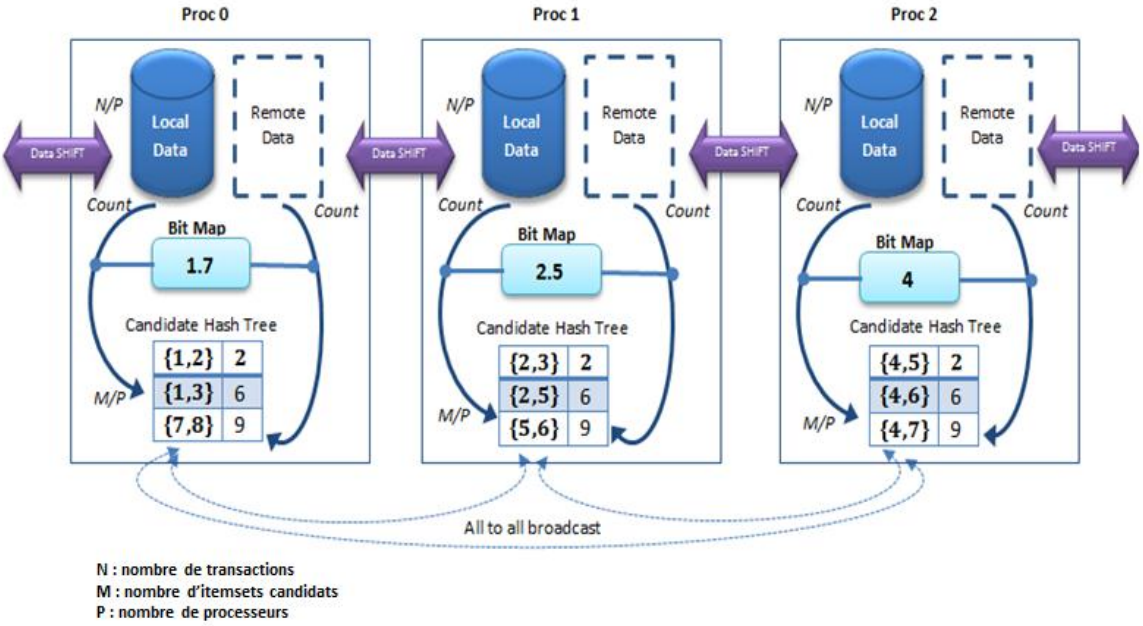

Figure II-7 : Illustration de l'Algorithme Intelligent Data Distribution

La figure II-7 montre clairement le principe de l'algorithme IDD. Dans cet exemple, Le processeur 0 a tous les candidats commençants par les items 1 et 7, le processeur 1 a tous les candidats commençants par les items 2 et 5, et le processeur 2 a tous les candidats commençants par 4. Chaque processeur garde les premiers items de leurs candidats dans bit-map. Dans l'algorithme Apriori, au niveau de la racine de hash-tree, chaque item dans une transaction est haché et vérifié dans hash-tree. Cependant, dans IDD, au niveau de la racine, chaque processeur filtre chaque item de la transaction par la vérification dans bit-map, pour voir si le processeur contient des candidats commençants par l'item figurant dans la transaction. Si le processeur ne contient pas les candidats commençants par cet item, le processeur peut sauter cet item. Par exemple, soit {1 2 3 4 5 6 7 8} une transaction, au niveau supérieur de hash-tree , le processeur 0 considère seulement les items commençants par 1 et

7 (1+ 2 3 4 5 6 7 8 et 7 + 8). si cette transaction est traitée dans le processeur 1, ce processeur considère seulement les items commençant par 2 et 5 (2 + 3 4 5 6 7 8 et 5 + 6 7 8). [45]

II.4.2.4. Hash Partioned Apriori (HPA)

L'algorithme HPA [46] est conçu dans le même esprit que l'algorithme IDD, il utilise la fonction de hachage pour partitionner les itemsets candidats entre les différents processeurs. Ce qui élimine le broadcast de tous les transactions et permet de réduire la charge de travail de manière significative.

Chaque processeur procède comme suit à l'itération k :

1. ***Génération des k-itemsets candidats :*** chaque processeur génère les k-itemsets candidats en utilisant les (k−1)-itemsets fréquents de l'iteration précédente. La fonction de hachage est ensuite appliquée aux itemsets candidats pour déterminer l'ID (identifiant) de leur processeur hôte. Si ID est son propre identificateur, alors l'itemset est inséré dans la table de hachage. Sinon, il est rejeté.
2. ***Parcourir les transactions de la base de données et calculer les supports:*** chaque processeur lit les transactions de sa base de données locale, et génère l'ensemble des k-itemsets à partir de ces transactions en éliminant ceux dont le support est inférieur à minsup. La fonction de hachage utilisée dans la phase 1 est alors appliquée à chaque k-itemset pour déterminer l'ID de son processeur hôte, et envoie le k-itemset à ce processeur. Pour les itemsets reçus par les autres processeurs et ceux générés localement dont ID est égale au propre identificateur du processeur, chercher dans la table de hachage et incrémenter la valeur de son support.
3. ***Détermination des itemsets fréquents :*** une fois que toutes les transactions sont traitées, chaque processeur détermine localement les itemsets fréquents à partir de ses itemsets candidats. La totalité des itemsets fréquents est obtenue par l'union des itemsets fréquents provenant de tous les processeurs.

II.4.3. D'autres types de parallélisme

Il existe d'autres algorithmes qui ne peuvent pas être classifiés dans les deux paradigmes. Ils ont des particularités distinctes. Ces algorithmes incluent :

II.4.3.1. Hybrid Distribution (HD)

L'algorithme Hybrid Distribution [44] combine Count Distribution et Intelligent Data Distribution. Il partitionne les P processeurs en G groupe de tailles égales, où chaque groupe est considéré comme un super-processeur contenant P/G processeurs. L'algorithme IDD est utilisé au sein des groupes et l'algorithme CD entre les groupes. La base de données est partitionnée horizontalement entre les super processeurs G, et les candidats sont repartis entre les P/G processeurs dans un groupe. En outre, la distribution hybride ajuste le nombre de groupes de manière dynamique à chaque passage.

II.4.3 2. ParEclat

Cet algorithme [47] repose sur le découpage de la base en classe d'équivalence, où chaque processeur calcule tous les itemsets fréquents d'une classe d'équivalence. Il découple les dépendances entre les processeurs, dont chacun peut procéder indépendamment, il n'a y pas une synchronisation coûteuse à la fin de chaque itération. En outre, Eclat utilise une disposition verticale de la base de données les Tid-listes. Notons que les tid-listes élaguent automatiquement les transactions non pertinentes, quand la taille de l'itemset augmente, la taille de tid-liste diminue, résultant par des intersections rapides. Ainsi la base de données locale est parcourue une fois seulement. L'algorithme *ParEclat* a quatre phases différentes :

- **La phase d'initialisation :** Chaque processeur calcule le support local de chaque 2-itemset de sa base de données locale. Ceci est suivi par des échanges des supports locaux entre tous les processeurs pour construire des supports globaux. À la fin de la phase initiale, tous les processeurs ont les supports globaux du 2-itemsets fréquent L_2.

- **La phase de transformation :** cette phase est constituée de deux sous-phases. Premièrement, il commence par partitionner L_2 en classes d'équivalence qui seront redistribuées sur les processeurs avec une politique d'équilibrage (assigner toutes les classes d'équivalences par charge décroissantes au processeur qui a la charge la plus petite au moment de l'assignation). En second lieu, il fait une transformation verticale de la base de données, et la repartitionne de sorte que chaque processeur aura sur son disque local les tid-listes de tous les 2-itemsets de sa classe d'équivalence.

- **La phase asynchrone :** À la fin de la phase de transformation, la base de données a été redistribuée de sorte que les tid-listes de tous les 2-itemsets dans les classes d'équivalence locales résident sur le disque local. Chaque processeur calcule indépendamment tous les itemsets fréquents sans synchronisation avec les autres processeurs en utilisant la procédure *compute_frequent*. Il lit les tid- listes pour les 2-itemsets dans chaque classe d'équivalence directement avant de passer à la prochaine classe, cette étape implique un balayage de la base de données locale une fois seulement. Dans chaque classe d'équivalence, il prend toutes les paires de 2-itemsets, et fait l'intersection de leurs tid-listes correspondantes. Si la cardinalité de tid-liste résultante excède le support minimum, le nouveau itemset (3-itemsets fréquent résultant) est inséré à L_3 dans les classes d'équivalence basées sur des préfixes communs de longueur 2. Tous les pairs de 3-itemsets dans une classe d'équivalence sont intersectés pour déterminer L_4. Ce processus est répété jusqu'à ce qu'il n'y ait plus de k-itemsets fréquents à trouver.

- **La phase de réduction finale :** La dernière tâche de l'algorithme accumule et la réunit les résultats de chaque processeur.

II.5. Discussion et étude comparative

Après une étude détaillée des algorithmes d'extraction de règles d'association dans un environnement parallèle et distribué. Nous observons que la majorité de ces algorithmes est basée sur l'algorithme Apriori.

II.5.1. Parallélisme de données

Dans le parallélisme de données avec lequel chaque site calcule le même ensemble de candidats qui sont dupliqués sur tous les processeurs et la base de données est distribuée à travers ces processeurs. On a l'algorithme CD (count distribution) basé sur l'algorithme Apriori, il est réalisé dans un environnement distribué où chaque site calcule les supports locaux des candidats sur sa base de données locale, et les diffuse aux autres sites pour calculer le support global. En conséquence chaque processeur génère le même ensemble global d'itemsets fréquents qui est important à chaque étape. Par ce fait, la réplication du calcul dégrade les performances. CD a un simple système de communication pour échanger les supports locaux, il utilise une communication basée sur une diffusion générale (Broadcast all-to-all). Il souffre par contre d'un fort coût de communication dû à la diffusion générale de l'ensemble exorbitant des candidats générés à chaque étape.

Pour pallier ces limites une autre parallélisation de l'algorithme Apriori est proposée, l'algorithme FDM (Fast Distributed Mining). Il propose ainsi de nouvelles techniques pour réduire le nombre des candidats considérés pour le calcul du support. La première, l'élagage local qui consiste à supprimer l'élément X de l'ensemble de candidats sur le site si X n'est pas localement fréquent par rapport à sa base de données locale. Après l'élagage local, chaque site diffuse les supports des candidats restant aux autres sites. Chaque site somme ces supports locaux pour générer les itemsets globalement fréquents, cette technique est appelée l'élagage global. L'algorithme FDM minimise ainsi l'ensemble des candidats générés permettant de réduire le nombre de messages transmis entre les sites et le temps d'exécution.

L'algorithme DMA (Distributed Mining Algorithm) améliore les performances de l'algorithme FDM. Il utilise une technique d'élagage au niveau de chaque site et ne garde que les itemsets fréquents. Les supports locaux des itemsets fréquents sur chaque site sont employés pour décider si un itemset fréquent est lourd (localement fréquents dans une partition et globalement fréquents dans la base de donnée entière). DMA génère un ensemble d'itemsets réduit puis il génère les candidats à partir des itemsets fréquents lourds. DMA introduit une nouvelle technique pour optimiser les communications et réduire la taille des messages échangés. Pour éviter la diffusion dupliquée des supports locaux de tous les candidats comme dans le cas de CD et de FDM, DMA associe à chaque itemset un site de vote qui est le responsable pour collecter les supports locaux de ses candidats.

Un autre algorithme ODAM (Optimized Distributed Association Mining) suit le paradigme de CD et de FDM. La spécificité d'ODAM est qu'il élimine tous les 1-itemsets globaux non fréquents de chaque transaction et insère la nouvelle transaction dans la mémoire, ce qui permet de réduire considérablement la taille de la basse de données traitée. Pour les

itérations suivantes, il travaille sur les nouvelles transactions et il calcul les supports des itemsets candidats ensuite diffuse les supports locaux pour générer les itemsets fréquents globaux. ODAM propose une autre amélioration pour minimiser les communications, les supports locaux sont envoyés à un emplacement simple appelé le récepteur. Ce dernier annonce les itemsets globalement fréquent aux emplacements distribués.

II.5.2. Parallélisme de taches

Dans le parallélisme de taches où les candidats et la base de données sont partitionnés et distribués à travers les processeurs. Chaque processeur, envoie sa partition aux autres processeurs et calcule les supports globaux de son ensemble de candidats, ce qui nécessite deux passages de communication. La première proposition l'algorithme DD (Data Distribution), c'est une autre parallélisation de l'algorithme Apriori. Dans DD, les processeurs partagent le travail par candidats et pas par portion de base de données comme dans le cas de CD. Chaque processeur envoie sa base de données locale à tous les autres processeurs, en même temps il calcule les supports de sa portion de l'ensemble des itemsets candidats sur sa base de données locale et sur les partitions de base données envoyées par les autres processeurs dans chaque itération. L'avantage majeur de DD est qu'il est conçu pour mieux exploiter la mémoire globale disponible du système. D'un autre coté, cet algorithme implique une quantité non négligeable de communication entre tous les processeurs pour échanger leurs partitions locales à chaque étape. Il souffre ainsi d'un fort cout de communication dû à cette diffusion.

Afin de réduire le cout et la charge en communication et éviter les synchronisations requises de l'algorithme DD, l'algorithme CandD (Candidate Distribution) a été ainsi proposé, A partir de la première itération jusqu'à L^{ieme} itération, ou l est déterminé. CandD utilise l'algorithme CD ou DD. A l'itération L, l'algorithme repartitionne la base de données et les itemsets fréquents de taille L-1 entre les processeurs d'une façon que chaque processeur peut générer son propre ensemble de candidats et calcule les itemsets fréquents dans les itérations suivantes indépendamment des autres processeurs, où les portions de base de données peuvent être dupliquées sur plusieurs processeurs. Les seuls échanges demandés dans CandD c'est pendant l'étape de l'élagage des candidats générés, sachant que ces échanges ne sont pas synchronisés, c.-à-d. que le processeur ne doit pas attendre l'information de tous les processeurs, l'information reçue en retard peut être utilisée dans l'itération suivante. Le repartitionnement de la base de données et des itemsets candidats dans l'algorithme CandD permet d'éliminer les dépendances entres les processeurs, et de minimiser largement les échanges entres les processeurs requis dans DD, c.-à-d. les échanges des supports des candidats et de la base de données dans chaque itération.

Une nouvelle version considérant les processeurs comme étant sur un anneau logique, c'est l'algorithme IDD (Intelligent Data Distribution). C'est une amélioration de l'algorithme DD, pour réduire les couts de communication, il emploie la communication asynchrone de point en point entre les voisins dans l'anneau au lieu du broadcast. IDD propose ainsi un partitionnement intelligent des candidats basant sur le premier item des candidats au lieu

d'un partitionnement round robin du candidat. D'une sorte qu'une partition contienne les candidats avec le même préfix. De ce fait le processeur ne traite que les transactions contenants les préfixes assignés a ce processeurs, ce qui permet de réduire le calcul redondant dans DD.

Un autre algorithme HPA (Hash Partitioned Apriori) est conçu pour améliorer l'algorithme IDD. Il applique une fonction de hachage sur les itemsets candidats pour déterminer l'ID de leur processeur hôte, chaque itemset candidat est ensuite inséré dans la table de hachage de son processeur hôte. A l'étape suivante, chaque processeur calcul le support des k-itemsets candidats sur sa base de données local en éliminant ceux dont le support est inférieur à minsup. Chaque processeur applique la fonction de hachage à chaque k-itemset pour déterminer l'ID de son processeur hôte et envoie les supports des candidats aux processeurs hôtes correspondants. Par la suite, chacun des processeurs est responsable d'incrémenter la valeur du support pour les itemsets générés localement et ceux envoyés par les autres processeurs. Les itemsets fréquents sont envoyés alors au coordinateur pour déterminer les l'ensemble de tous les itemsets fréquents globales. Par-ce-fait, le nombre de communications est réduit puisque toutes les transactions de la base de données ne sont plus diffusées. Ainsi les itemsets fréquents sont envoyés uniquement au coordinateur et pas à tous les processeurs.

Un autre algorithme qui diffère des autres algorithmes a été introduit, L'algorithme ParEclat. C'est une parallélisation de l'algorithme séquentiel Eclat. Il parcourt la base de données pour produire les itemsets fréquents de taille 2 (L_2). Ensuite, partitionne L_2 en classes d'équivalence qui seront redistribuées sur les processeurs avec une politique d'équilibrage. Il fait une transformation verticale de la base, et transmettre respectivement les listes de transactions aux processeurs chargés de la classe d'équivalence correspondante. Parallèlement, chaque processeur génère indépendamment les itemsets fréquents par l'intersection des Tid-listes des éléments de chaque classe d'équivalence. La dernière tâche de l'algorithme consiste en l'accumulation et la réunion des résultats de chaque processeur. La tâche la plus couteuse de l'algorithme ParEclat est en général la transmission des listes de transactions de taille importante de chaque item aux autres processeurs. De plus, Cet algorithme nécessite un temps considérable pour partitionner l'ensemble des items en classe d'équivalence et de transformer la base locale en base verticale. Par contre, il ne demande aucune synchronisation ou échange d'informations entre les processeurs. Aussi, la base de données est scannée une fois seulement pour produire les itemsets fréquents de taille 2.

Une autre solution proposée combine CD et IDD, L'algorithme HD (Hybrid Distribution). Il partitionne les P processeurs en G groupe de tailles égaux, chacun des groupes est considéré comme super-processeur. L'algorithme IDD est utilisé au sein des groupes où les candidats sont repartis entre les P/G processeurs dans un groupe. L'algorithme CD est appliqué entre les groupes où la base de données est partitionnée horizontalement entre les Groupes G. A chaque passage, le nombre de groupes est ajusté de manière dynamique. Les avantages de HD

est qu'il réduit les coûts de communication de la base de données et qu'il essaie de garder le processeur occupé.

Il existe d'autres travaux qui ne sont pas présentés dans ce chapitre. On a l'algorithme PDM (Parallel Data Mining) [48] qui est basé sur l'algorithme séquentiel DHP, et suit le même paradigme de CD. PDM utilise une technique appelé clue-and-poll pour résoudre l'incertitude en raison de la connaissance partielle collecté dans chaque processeur, en choisissant intelligemment une petite fraction des itemsets pour échanger leurs supports avec les autres processeurs. Ce qui permettra de réduire le cout de communication.

L'algorithme FPM (Fast Parallel Mining) [49] adopte l'algorithme CD en incorporant deux nouvelles techniques d'élagage, l'élagage distribué et l'élagage global. FPM nécessite un seul passage de communication à chaque itération. Aussi, Les deux techniques d'élagage réduit de manière remarquable le nombre de candidats. Par conséquence, FPM minimise le cout de communication.

PPAR (Partitioned Parallel Association Rules) [50] est un algorithme parallèle de l'algorithme séquentiel Partition. Dans PPAR, chaque processeur rassemble les itemsets fréquents localement de toutes les tailles en parcourant sa base de données locale. Ensuite, broadcast tous les itemsets fréquents aux autres processeurs. Puis, chaque processeur calcul les supports de ses candidats globaux (réunion des itemsets fréquents locales) dans le deuxième parcourt de la base de données locale. Par cet effet, l'algorithme ne nécessite que deux passage de communication. En contrepartie, les données échangées entre les processeurs sont volumineuses puisque elles contiennent les itemsets fréquents localement de toutes les tailles.

L'algorithme PFP (Parallel FP-Growth) [51] est un autre modèle de parallélisation basé sur l'algorithme séquentiel FP-Growth. L'algorithme PFP partitionne le calcul entre les processeurs d'une façon que chaque processeur calcul les itemsets fréquents indépendamment des autres processeurs. Ce partitionnement élimine la synchronisation, les dépendances entre les processeurs et même les communications.

II.6. Conclusion

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.

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. Pour les algorithmes basés sur la répartition des itemsets, les

échanges nécessaires des tous les transactions (généralement volumineuses) entre les nœuds demandent beaucoup de communications y étant très pénalisantes. D'autre part, les algorithmes se basant sur la répartition des transactions échangent des informations calculés (les supports des itemsets), qu'on peut considérer comme communications très courtes. Ce pendant ces communications sont bloquantes (indispensables entre chaque deux itérations). Le nombre de communication dans ces algorithmes s'explique par le fait que la recherche des itemsets fréquents doit s'effectuer à partir des critères globaux. Les algorithmes présentés dans ce chapitre introduisent différentes propositions pour réduire les couts de communication et le nombre de balayage de la base de données.

Après une étude approfondie de ces algorithmes, l'algorithme Partition s'avère intéressant. Cet algorithme ne demande que deux parcours de la base de données et par conséquent deux passages de communication dans la version parallèle de Partition.

D'autre part, comme les partitions de la base peuvent influencés sur les résultats de l'algorithme Partition parallèle, et vu que le partitionnement de la base se fait de manière aléatoire. Pour cela, nous avons pensé à introduire des modifications pour améliorer les performances de notre algorithme. Ainsi, nous proposons à faire un partitionnement intelligent de la base de données utilisant le clustering, pour étudier ses impacts sur le temps d'exécution. Afin d'étudier les algorithmes de clustering, nous détaillons dans le chapitre suivant les différentes propositions pour le clustering distribué.

CHAPITRE III : CLUSTERING DISTRIBUÉ

III.1. Introduction

Le clustering est défini comme une division de l'ensemble de données en groupes appelés Clusters. Les tâches de clustering ont été exploitées dans beaucoup de domaines comprenant le traitement d'image/vidéo, apprentissage machine, data mining, biochimie et bioinformatique. Selon les propriétés de données ou le but de clustering, différents types d'algorithmes de clustering ont été développés, comme, le clustering de partitionnement, hiérarchique, basé sur la densité...etc.

La majorité des algorithmes de clustering exigent des procédures itératives pour trouver les solutions localement ou globalement optimales à partir d'un ensemble de données de haut-dimension. En outre, les données réelles présentent très rarement une solution de clustering unique et il est également difficile d'interpréter la représentation de cluster. Par conséquence, il exige beaucoup d'expérimentations avec différents algorithmes ou avec différents dispositifs du même ensemble de données, ce qui demande des calculs très couteux. De cet effet, la parallélisation des algorithmes de clustering est une approche très pratique, et les méthodes parallèles seront diverses pour différents algorithmes. Dans ce chapitre, une introduction au clustering sera présentée avec les différents concepts de base. Nous discuterons par la suite les algorithmes adéquats pour notre objectif, les algorithmes de partitionnement. Ainsi, nous fouillerons nos recherches sur les algorithmes de partitionnement parallèles.

III.2. Le clustering

Le clustering groupe les objets ou les données en se basant sur les informations décrivant les objets et leurs relations. Le but est que les objets dans le même groupe soient similaires (ou liés) et différents ou non liés aux objets des autres groupes. Plus la similarité dans un groupe est grande et plus la différence entre les groupes est grande, le clustering produira de meilleurs résultats ou des groupes plus distincts.

Dans plusieurs applications, la notion de cluster n'est pas bien définie. Pour mieux comprendre la difficulté de décider que constitue le cluster, considérant la figure qui montre 20 point et trois façons pour diviser ces points dans des clusters. La figure III-1.b et la figure III-1.d divisent les données en deux et six clusters respectivement. Cependant, la division apparente de chacun des deux grands clusters en trois sous-clusters peut être simplement réalisée par le système visuel humain. Aussi, il se peut raisonnable de voir que ces points forment quatre clusters, comme le montre la figure III-1.c. Cette figure illustre que la définition de cluster est imprécise et que la meilleure définition dépend d'un côté de la nature de données et d'un autre côté du résultat désiré.

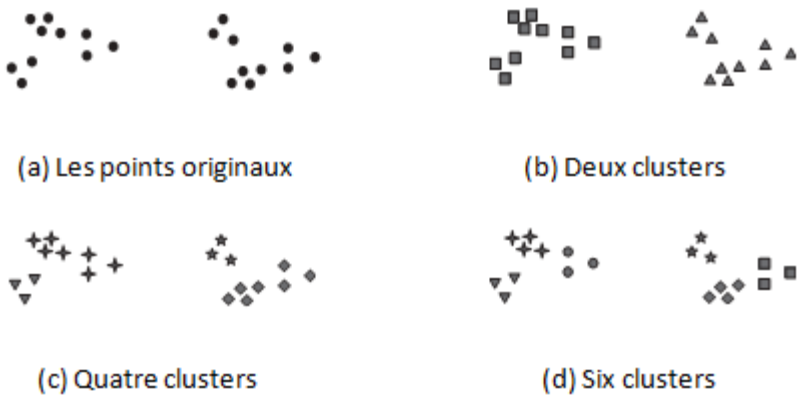

Figure III-1 : différentes manières de clustering pour le même ensemble de points

On peut dire que Le clustering est définie comme organisation d'une collection de modèles dans des clusters basant sur la similarité. La difficulté se situe dans la définition et la portée du cluster dans l'ensemble de données. Cependant, Il existe différentes interprétations du cluster :

1) le cluster est un ensemble d'objets semblables et les objets dans différents clusters ne devraient pas être semblables ;
2) un cluster est l'ensemble de points qui sont agrégés dans l'environnement d'essai tels que la distance entre deux points dans un cluster est moins que la distance entre les points des autres clusters;
3) les clusters peuvent être des régions denses dans un espace multidimensionnel séparé par des points reliés librement.

Basant sur le but de l'application, différents concepts de clusters et différents algorithmes de clustering peuvent être appliqués. Le premier concept se concentre sur l'intra-cluster minimum, alors que le second considère les entre-clusters et les intra-clusters en même temps. D'autre algorithmes de clustering basés sur la densité sont développés dont la plupart sont basé sur le dernier concept.

III.2.1. le but de clustering

Les applications de clustering sont nombreuses, elles sont classées en trois groupes principaux selon l'objectif à atteindre. On a la segmentation, la classification et l'extraction de connaissances.

La segmentation d'une base de données a pour objectif principal de réduire la taille de l'ensemble des données considérées afin de favoriser leur traitement. On parle dans ce cas de condensation ou de compression de données, celle-ci pouvant alors être considérées collectivement. Une telle technique est par exemple utile en segmentation d'images, afin d'identifier les différentes zones homogènes de l'espace décrit (les champs, maisons, routes, fleuves, etc.). De manière plus générale, une telle méthode est utile pour segmenter l'espace dans les bases de données spatiales. Parfois, la segmentation est utilisée pour discrétiser une base de données, c'est-à-dire transformer la description complexe des objets par un unique attribut caractérisant leur appartenance à une classe identifiée automatiquement.

La classification concerne typiquement l'identification de sous-population ayant des caractéristiques proches dans les bases de données. Par exemple, dans une base de données client, l'objectif principal est d'établir des profils de clients aux comportements similaires, afin de comprendre leurs attentes et leurs besoins en fonctions de leur profil. De telles informations permettent alors d'adapter les actions marketing envers les clients en fonction de leur appartenance à une population données. Les applications principales de telle technique concernent donc la gestion de la relation client.
Finalement, l'utilisation de clustering pour l'extraction de connaissances n'a pas d'objectif préétabli. Il s'agit dans ce cas d'aider à la compréhension de la structure des données en les organisant en groupes homogènes et en faisant ainsi émerger certains sous-concepts dans les données. On parle dans ce cas de génération d'hypothèses ou de modélisation prédictive, par l'inférence de règles caractérisant les données. L'intérêt de telles techniques est par exemple avéré pour effectuer des diagnostics médicaux sur des bases de données de patients, en ciblant par exemple différents groupes de patients atteints d'une maladie données, reflétant ainsi différentes causes possibles pour la maladie. De même, cette technique peut être utilisée en analyse du web, en analyse des données textuelles, en analyse de ventes, en bioinformatique, en analyse génétique, etc.

III.2.2. Les étapes du processus de clustering

Le processus de clustering peut donner comme résultat différents partitionnement de la base de données, selon des critères spécifiques utilisés dans le clustering. Ainsi, un prétraitement est nécessaire avant d'assumer la tâche de clustering sur la base de données. Les étapes de base pour développer le processus de clustering sont présentées dans la figure III-2, il est devisé en quatre étapes [52].

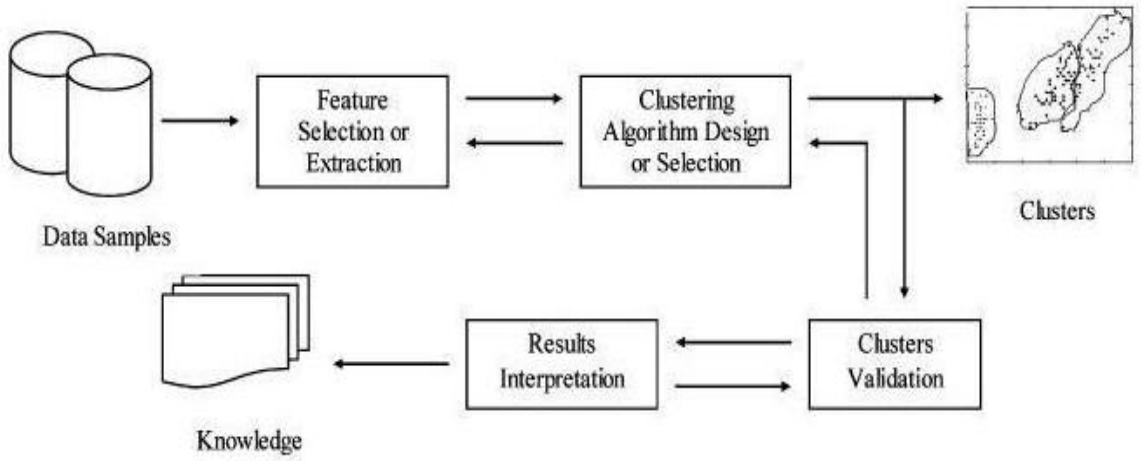

Figure III-2 : Les étapes du processus de clustering

1. **Sélection des attributs :** Le but est de choisir correctement les attributs sur lesquels le clustering doit être exécuté afin de coder autant que possible l'information concernant la

tâche désirée. Ainsi, le prétraitement des données peut être nécessaire avant leur utilisation dans la tâche de clustering.

2. **Algorithme groupant :** Cette étape dépend du choix de l'algorithme résultant dans la définition d'un bon schéma de clustering pour l'ensemble de données. La mesure d'proximité et le critère groupant caractérisent principalement un algorithme de clustering aussi bien que son efficacité pour définir les clusters représentant la base de données.
   - ***La mesure de proximité.*** C'est une mesure qui quantifie comment les deux points de repères sont similaires (c.-à-d. vecteurs d'attributs sont similaires). Dans la plupart des cas nous devons nous assurer que tous les attributs choisis contribuent également au calcul de la mesure de proximité et il n'y a aucun attribut qui domine d'autres.
   - ***Critère groupant.*** Dans cette étape, nous devons définir le critère groupant, qui peut être exprimé par l'intermédiaire d'une fonction de coût ou d'un autre type de règles. Nous devrions soumettre à une contrainte que nous devons tenir compte du type de cluster désirés dans l'ensemble de données. Ainsi, nous pouvons définir un bon critère groupant, menant à un partitionnement qui forme des puits d'ajustement de l'ensemble de données.
3. **Validation des résultats :** L'exactitude des résultats de l'algorithme de clustering est vérifiée en utilisant des critères et des techniques appropriés. Comme les algorithmes de clustering définissent des clusters qui ne sont pas connus a priori, indépendamment des méthodes de clustering, la partition finale des données exige un certain genre d'évaluation dans la plupart des applications.
4. **Interprétation des résultats** : Dans beaucoup de cas, les experts des domaines d'application doivent intégrer les résultats de clustering avec d'autres analyses expérimentales afin de tirer la bonne conclusion.

III.3. Concepts de base

III.3.1. définitions et notations

Des termes utilisés fréquemment dans le processus de clustering seront définis dans ce qui suit :

- ***Un modèle (un objet, un point, un individu ou vecteur d'attributs) X :*** c'est une donnée élémentaire simple utilisé par l'algorithme de clustering. elle se compose typiquement d'un vecteur de mesures $X=(x_1, \ldots x_d)$
- ***L'ensemble d'objets ou de modèles $I=\{X_1, \ldots X_n\}$:*** le $i^{éme}$ objet dans l'ensemble d'objets est dénoté $Xi=(x_{i,1}, \ldots x_{i,d})$, généralement un ensemble de modèles à grouper est considéré comme une matrice de taille n×d.
- ***Cluster ou groupe :*** un ensemble d'objets ou d'individus qui sont le plus possible similaires.
- ***Les mesures de ressemblance :*** La notion de ressemblance (ou proximité) a fait l'objet d 'importantes recherches dans des domaines extrêmement divers . Pour qualifier cet opérateur, plusieurs notations comme la similarité, la dissimilarité ou la distance peuvent être utilisées.

Nous appelons similarité ou dissimilarité toute application à valeurs numérique qui permet de mesurer le lien entre les objets d'un même ensemble. Pour une similarité, le lien entre deux objets sera d'autant plus fort que sa valeur est grande. Pour une dissimilarité le lien sera d'autant plus fort que sa valeur de la dissimilarité est petite.

- **Indice de dissimilarité**
  Un opérateur de ressemblance d : X*X → R+ défini sur l'ensemble d'individus X={x_1, x_2, …, x_n} est dit indice de dissimilarité (ou dissimilarité), s'il vérifie les propriétés suivantes :
  1. $\forall\ x_i, x_j \in X$; $d(x_i, x_j) = d(x_j, x_i)$ (priorité de symétrique)
  2. $\forall\ x_i \in X$; $d(x_i, x_j) \geq d(xi, xi) = 0$ (priorité de positivité)
- **Indice de similarité**
  Un opérateur de ressemblance s : X*X →[0, 1] défini sur l'ensemble d'individus X={x_1, x_2, …, x_n} est dit indice de similarité (similarité), s'il vérifie en plus de la propriété de symétrique (1) les propriétés suivantes :
  3. $\forall\ x_i, x_j \in X$; $d(x_i, x_j) \geq 0$ (priorité de positivité)
  4. $\forall\ x_i, x_j \in X$ et $x_i \neq x_j$; $d(x_i, x_i) = s(x_j, x_j) > s(x_i, x_j)$ (priorité de maximisation)
- **La distance**
  Un opérateur de ressemblance d : X*X → R+ défini sur l'ensemble d'individus X={x_1, x_2, …, x_n} est dit distance, s'il vérifie enplus des deux propriétés 1 et 2 les propriétés d'identité et d'inégalité triangulaire suivantes :
  5. $\forall\ x_i, x_j \in X$; $d(x_i, x_j) = 0 \Rightarrow x_i = x_j$ (propriétés d'identité)
  6. $\forall\ x_i, x_j, x_k \in X$; $d(x_i, x_j) \leq d(x_i, x_k) + d(x_k, x_j)$ (inégalité triangulaire)

- ***Matrice de similarité :*** la similarité entre les individus calculée par la fonction de similarité ***s (x_i, x_j)***, est représentée sous forme d'une matrice appelée la matrice de similarité.
- ***Le point initial de cluster :*** le premier individu d'un cluster est défini comme initiateur de ce cluster c.-à-d. chaque individu entrant est comparée avec l'initiateur. L'initiateur s'appelle le point initial de cluster.

III.3.2. la similarité entre les objets ou individus

Formellement, la similarité entre les individus est évaluée par une fonction de distance, définie entre paire d'individus. La définition de la fonction de distance se diffère selon le type de données. Généralement les données traitées appartiennent à deux classes :

- ***Les variables quantitatives :*** Une variable quantitative prend des valeurs ordonnées (comparable par la relation d'ordre ≤) pour lesquelles des opérations arithmétiques telle que différence et moyenne aient un sens. Une variable quantitative peut être :

- Binaire : les variables binaires ne peuvent prendre que deux valeurs, le plus souvent associées à {0, 1}. Exemples : {absence, présence}, {succès, échec}
- Continue : les variables continues ou d'échelle sont les variables dont les valeurs forment un sous-ensemble infini de l 'ensemble des réels, exemples : le salaire, le coût du séjour.
- Discrète : les variables discrètes sont celles dont les valeurs forment un sous-ensemble fini ou infini de l'ensemble des entiers naturels, exemples : le nombre de jours d'hospitalisation, le nombre d'enfants.

- ***Les variables qualitatives :*** Une variable qualitative (catégorielle) est une donnée dont l'ensemble des valeurs est fini. Elle prend des valeurs symboliques qui désignent des catégories ou aussi modalités, par exemples : le code de la ville, la couleur des cheveux. On ne peut effectuer aucune opération arithmétique sur ces variables. Une variable catégorielle peut être :
  - Ordonnée : les variables ordonnées ou ordinales peuvent être rangées dans la famille des variables discrètes et traitées comme telles. Exemple : (faible, moyen, fort)
  - Non ordonnée : les variable non ordonnées sont dites nominales et lorsque ces variables ont pour valeur des textes non codés, écrits en langage naturel, elles sont dites variables textuelles, par exemples : rapports, nom de film.

Soit X= {X_1, X_2, …, X_n} un ensemble d'individus à organiser sous la forme d'un ensemble de classes à la fois homogènes et centralisées. L'ensemble d'individus X est décrit généralement sur un ensemble de m variable Y= {Y_1, Y_2, …, Y_m}. La distance entre les individus est définie suivant les types de m variables comme suit :

III.3.2.1. la distance entre les données binaires

Considérant les n individus décrits par m variables binaires codées 0 ou 1. La ressemblance entre deux individus X_i et X_j se calcule à partir des informations du tableau de contingence 2x2 ci-dessous. Ce tableau permet de compter le nombre de concordance (a+d) et le nombre de discordance (b+c) entre les individus.

		X_j	
		1	**0**
X_i	**1**	**a**	**b**
	0	**c**	**d**

Tableau III-1 : Tableau de contingence

Dans le calcul d'une mesure de ressemblance entre les individus, le rôle des modalités d'une variable binaire est très important. En effet, une variable binaire peut être symétrique (les modalités 0 et 1 de cette variable ont la même importance) ou asymétrique (les deux

modalités n'ont pas la même importance). Cependant, de nombreux indices de similarité (ou distances) entre deux individus X_i et X_j sont apparus, parmi les plus connus :

- L'indice de Jaccard [53]: il est défini pour des variables asymétriques et ne tient pas compte de l'absence conjointe :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \frac{a}{a + b + c}$$

- L'indice de Sokal et Sneath [54]: il ignore également l'absence conjointe mais contrairement à l'indice de Jaccard compte doublement les discordances :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \frac{a}{a + 2 * (b + c)}$$

- L'indice de Russel et Rao où l'absence conjointe n'est pas considérée comme une similarité :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \frac{a}{a + b + c + d}$$

- L'indice Simple Match de Sokal et Michener [55]: est défini pour des variables symétriques et l'absence conjointe est considérée comme une similarité :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \frac{a + d}{a + b + c + d}$$

La distance euclidienne peut être utilisée pour définir les ressemblances entre individus représentés par des données binaires.

III.3.2.2. la distance entre les données numériques (continues ou discrètes)

Les mesures de distance les plus courantes entre deux individus Xi et Xj décrits par m données quantitatives discrètes ou continues sont les suivantes :

- La distance de Minkowski d'odre $\propto$ definie dans R^m par :

$$\forall X_i, X_j \in X; \; d(X_i, X_j) = \left(\sum_{h=1}^{m} \left| Y_h(X_i) - Y_h(X_j) \right|^{\propto} \right)^{\frac{1}{\propto}}$$

Où $\alpha \geq 1$, avec si :

- α=1, ***d*** est la distance de City-block ou Manhattan :

$$d(X_i, X_j) = \sum_{h=1}^{m} \left| Y_h(X_i) - Y_h(X_j) \right|$$

- α=2, ***d*** est la distance Euclidienne classique :

$$d(X_i, X_j) = \sqrt{\sum_{h=1}^{m} \left(Y_h(X_i) - Y_h(X_j) \right)^2}$$

- $\alpha=\infty$, ***d*** est la distance de Chebyshev :

$$d(X_i, X_j) = \max_{1\le h\le m} |Y_h(X_i) - Y_h(X_j)|$$

On a le plus souvent recours à la distance euclidienne mais la distance de Manhattan est aussi parfois utilisée, notamment pour atténuer l'effet de large différences dues aux point atypiques (aberrants ou outliers) puisque leurs coordonnées ne sont pas élevées au carré. Dans la plupart des cas, la distance de Manhattan donne des résultats semblables à ceux de la distance euclidienne. D'autre part, Il existe de différentes définitions de distance autres que la distance de Minkowski, on peut citer :

- Le cosinus :

$$\forall X_i, X_j \in X;\ d(X_i, X_j) = Cos(X_i, X_j) = \frac{X_i \cdot X_j}{\|X_i\| \cdot \|X_j\|} = \frac{\sum_{h=1}^{m} \left(Y_h(X_i) \times Y_h(X_j)\right)}{\sqrt{\sum_{h=1}^{m} \left(Y_h(X_i)\right)^2 \sum_{h=1}^{m} \left(Y_h(X_j)\right)^2}}$$

- La corrélation de Pearson, cosinus de l'écart à la moyenne :

$$\forall X_i, X_j \in X;\ d(X_i, X_j) = Pearson(X_i, X_j) = \frac{\sum_{h=1}^{m} \left(Y_h(X_i) - \overline{X_i}\right)\left(Y_h(X_j) - \overline{X_j}\right)}{\sqrt{\sum_{h=1}^{m} \left(Y_h(X_i) - \overline{X_i}\right)^2 \sum_{h=1}^{m} \left(Y_h(X_j) - \overline{X_j}\right)^2}}$$

Avec $\overline{X_i}$ la moyenne des valeurs de l'individu X_i sur l'ensemble des attributs :

$$\overline{X_i} = \frac{\sum_{h=1}^{m} Y_h(X_i)}{m}$$

III.3.2.3. la distance entre les données ordinales

Dans la définition de la distance entre individus décrits par les données qualitatives ordinales. Les valeurs d'attributs $Y_h(X_i)$ sont remplacées par leurs rangs $R_h(X_i)$, $R_h(X_i)=1, 2, \ldots, p_h$, où p_h est le nombre de valeurs distinctes de la variable Y_h. Ces valeurs sont transformées par la suite en utilisant la formule suivante, cette dernière fournit une variation $Z_h(X_i)$ entre 0 et 1 de $Y_h(X_i)$:

$$\forall X_i \in X;\ Z_h(X_i) = \frac{R_h(X_i) - 1}{P_h - 1}$$

Ainsi, à partir des variations Z_h considérées comme des données numériques. La distance entre deux individus X_i et X_j est calculée.

III.3.2.4. la distance entre les données nominales

Etant donné deux individus X_i, X_j décrits par m variables nominales, la distance la plus utilisée pour ce type de données est celle de Hamming. Notant que, entre X_i et X_j la distance de Hamming est donnée par le nombre de caractéristiques de X_i qui différent de celle de X_j comme suit :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \sum_{h=1}^{m} Y_h(X_i) \oplus Y_h(X_j)$$

Où

$$Y_h(X_i) \oplus Y_h(X_j) = \begin{cases} 1, & Y_h(X_i) \neq Y_h(X_j) \\ 0, & sinon \end{cases}$$

Une autre solution consiste à prendre également en compte les fréquences d'apparition de chaque catégorie sur l'ensemble de la base de données :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \sum_{h=1}^{m} \frac{N_{Y_h(X_i)} + N_{Y_h(X_j)}}{N_{Y_h(X_i)} \times N_{Y_h(X_j)}} \times \left(Y_h(X_i) \oplus Y_h(X_j)\right)$$

Où $N_{Y_h(X_i)} = \left|\{j \in D | Y_h(X_j) = Y_h(X_i)\}\right|$ représente le nombre total d'individus de la base pour les quels la catégorie sur l'attribut ***h*** est $Y_h(X_i)$.

Il existe une autre distance utilisée considérablement pour les données nominales, c'est la distance de Hamming normalisée. Elle est basée sur le calcul du nombre d'appariements entre les individus. Soit deux individus X_i et X_j présentant les mêmes modalités pour ***p*** parmi les m variables nominales, la distance de Hamming normalisée est définie comme suit :

$$\forall X_i, X_j \in X; d(X_i, X_j) = \frac{m - p}{m}$$

Enfin, il est également possible de prendre en compte les connaissances d'un expert du domaine dans le calcul de distance entre individus. Celui-ci affecte alors un poids à chaque attribut en fonction de l'importance de cet attribut pour le problème considéré.

III.3.3. les différents types de cluster

Le clustering vise à trouver des groupes utiles d'individus (clusters), où l'utilité est définie par les buts de l'analyse de données. Notamment, il y a différentes notions de cluster [56] qui s'avèrent utile dans la pratique :

- Bien séparé
- Basé sur prototype
- Basé sur le graphe
- Basé sur la densité
- Propriété partagée (clusters conceptuels)

III.4. Évaluation de la qualité de clustering

De Nombreuse méthodes d'évaluation ont été proposées dans la littérature [57], [58], [59].Elles sont groupées principalement en trois familles. La première famille contient les mesures non supervisées qui utilisent uniquement des informations internes aux données comme par exemple la distance entre les objets. Ces mesures sont également appelées

mesures de qualité internes. La deuxième famille contient les mesures supervisées qui calculent le degré de correspondance entre le clustering produit par l'algorithme et un partitionnement connu des données. Ces mesures sont aussi connues sous le nom de *mesures de qualité externes*. Le troisième groupe contient des *mesures relatives*, qui permettent pour un même algorithme de comparer les clusterings produits par celui-ci. Les mesures relatives sont donc simplement l'utilisation de critères internes ou externes pour faire un choix parmi plusieurs résultats produit par un même algorithme.

III.4.1. les mesures de qualité internes

Ces mesures [60][61][62][63][64] se basent souvent sur la dentition la plus simple du clustering qui définit que les individus d'un même cluster doivent être les plus proches possible entre eux et que les individus de deux clusters distincts doivent être les plus éloignés possible. Alors, des mesures de distance sont calculées pour évaluer si un clustering respecte cette définition intuitive. Ces mesures non supervisées calculent la distance entre les représentants des clusters et les individus du résultat permettant d'évaluer la compacité ainsi que la séparabilité des clusters. Certains de ces critères d'évaluation peuvent être directement utilisés comme fonction objective et être optimisés par un algorithme de clustering. Tandis, que d'autres sont trop coûteux à évaluer pour être calculés au cours de l'exécution d'un algorithme et sont par conséquent destinés à être calculés à l'issue de l'application de l'algorithme.

III.4.2. les mesures de qualité externes

Pour évaluer la qualité d'un clustering en fonction des classes connues de chaque individu, deux mesures [65][66] de la qualité externe du clustering sont classiquement utilisées : la F-mesure et l'Entropie.

III.5. Problèmes du clustering

Le clustering pose de nombreux problèmes malgré qu'il existe un nombre important de méthodes de clustering ainsi que leur utilisation avec succès dans de nombreux domaines. Les problèmes de clustering soulèvent de la difficulté de définir une mesure de similarité entre individus, de la définition d'une fonction objective pour un problème donné ou encore le manque de précision dans la définition de ce qu'est réellement un cluster. Dans [67] les auteurs ont listé un ensemble de questions qu'il est nécessaire de se poser lorsqu'on veut effectuer une tâche de clustering :

- Qu'est-ce qu'un cluster?
- Quels attributs doivent être utilisés?
- Les données doivent-elles être normalisées?
- Les données contiennent-elles des objets atypiques?
- Comment est définie la similarité entre deux objets?
- Combien de clusters sont présents?
- Quelle méthode de clustering doit-on utiliser?

- Est-ce que les données contiennent des clusters?
- Est-ce que la partition découverte est valide?

Ces questions génèrent un certains problèmes et limites fréquentatifs en clustering, ces problèmes peuvent être exprimés comme suit :

III.5.1. le choix du nombre de clusters

Le choix du nombre de clusters a été souvent étudié comme un des problèmes les plus difficiles en clustering. En effet, il est généralement nécessaire de fournir le nombre de clusters souhaité comme paramètre de l'algorithme. De ce fait, l'algorithme est couramment exécuté plusieurs fois indépendamment avec un nombre de clusters différent. Pour comparer les résultats, un critère de sélection est généralement utilisé permettant de choisir la meilleure solution, ce choix est toujours particulier et fortement dépendant du critère sélectionné.

Un algorithme ISODATA [68] est basé sur l'algorithme k-means permet une évolution du nombre de clusters pendant l'exécution. Son principe consiste à fusionner des clusters si le nombre d'individus dans un des clusters est inférieur à un seuil ou si deux clusters sont plus proches qu'un certain seuil. De plus, les clusters peuvent également être séparés en deux si leur dispersion (évaluée par exemple par l'écart type) dépasse un seuil. Cet algorithme permet d'avoir un nombre de clusters variables pendant l'exécution. En contrepartie, il nécessite la définition de nombreux seuils et ne fait souvent que déplacer le problème du choix du nombre de clusters à la définition de ces seuils.

Généralement, deux approches moins subjectives sont utilisées se basant sur les critères de Minimum Message Length (MML) [69] et Minimum Description Length (MDL) [70]. Ses principes consistent à débuter avec un nombre de clusters relativement élevé, puis à fusionner itérativement deux clusters pour optimiser les critères (MML ou MDL). Il existe également d'autres critères classiquement utilisés pour la sélection de modèles tels que Bayes Information Criterion (BIC), Akiake Information Criterion(AIC) et Gapstatistics [71]. Ces critères peuvent être plus difficiles à mettre en place lors de l'utilisation d'autres types d'approches. Aussi, elles sont relativement coûteuses et nécessitent de nombreuses exécutions des algorithmes.

D'autre outil pour effectuer le choix du nombre de clusters est l'étude de la validité des clusters découverts. Mais dans l'idéal, le choix du nombre de clusters reste à l'appréciation de l'expert qui est à même, avec ou sans l'aide d'indices, de choisir le nombre de clusters qui lui paraît adapté.

III.5.2. paramétrage des algorithmes

Les algorithmes de clustering aient un certain nombre de paramètres, ces derniers influent de manière plus au moins importante sur les résultats obtenus. Néanmoins, pour que l'algorithme soit relativement générique et soit applicable dans plusieurs cas, ces paramètres sont nécessaires. Autrement dit, plus le nombre de paramètres augmente plus l'algorithme est adaptable et peut être appliqué à une gamme de problèmes plus large. Tandis qu'un nombre

important de paramètres requiert également une connaissance importante de la part de l'expert sur ses données et sur le fonctionnement de l'algorithme. Un exemple de ces paramètres a été déjà discuté est le nombre de clusters. Bien qu'il existe bien d'autres paramètres qu'il serait impossible de les citer entièrement, on a le problème de l'initialisation qui est un problème important et qui concerne une grande partie des algorithmes existants. On peut prendre l'exemple d'initialisation des centroïdes dans l'algorithme k-means, dont les résultats peuvent être fortement différents en fonction du choix effectué pour cette initialisation. Il est possible d'étudier les formes fortes pour réduire ce problème, ce qui consiste à identifier les groupes d'individus qui sont régulièrement regroupés ensemble lors de plusieurs exécutions différentes de l'algorithme. Il est cependant difficile de choisir comment faire varier les paramètres et combien d'exécutions sont nécessaires.

La figure III-8, illustre un exemple simple pour ce problème d'initialisation, six individus sont représentés et seront regroupés en 2 clusters en utilisant l'algorithme k-means.

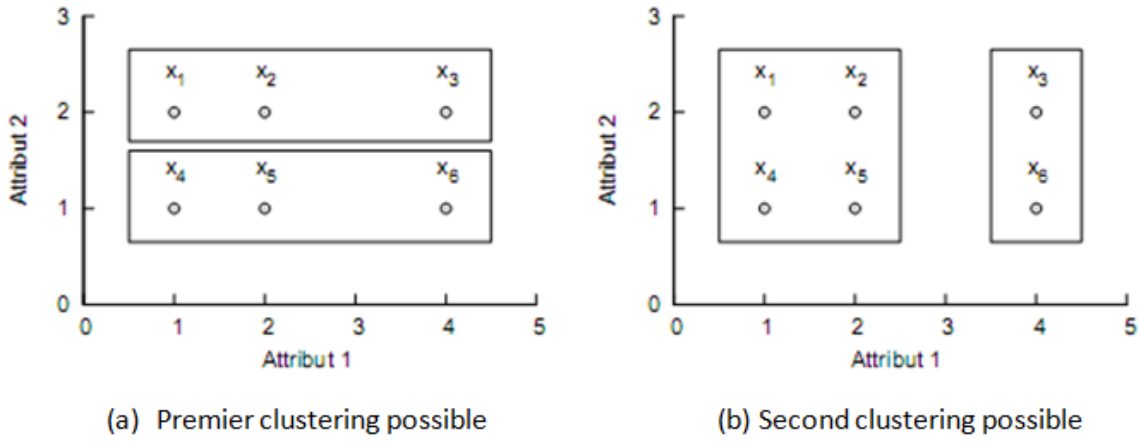

Figure III-3 : Exemple de données produisant des résultats différents suivant l'initialisation de k-means

Si à l'initialisation les individus x_1 et x_5 sont sélectionnés comme centres initiaux, l'algorithme convergera vers la solution $\{\{x_1, x_2, x_3\}, \{x_4, x_5, x_6\}\}$ (figure 43(a)) qui n'est pas une solution optimale de la fonction objective de k-means. Si par contre les individus x_2 et x_3 sont sélectionnés comme centres initiaux, l'algorithme convergera vers la solution $\{\{x_1, x_2, x_4, x_5\}, \{x_3, x_6\}\}$ (figure 43(b)) qui est la solution optimale de la fonction objective de k-means pour K= 2.

III.5.3. validité des clusters

Le but d'un algorithme de clustering est de découvrir des clusters. Néanmoins, il est nécessaire d'étudier les résultats obtenus pour s'assurer de la pertinence des clusters identifiés. La vérification de la validité des clusters découverts consiste à utiliser une procédure qui évalue le résultat obtenu de manière quantitative et objective.

Les critères d'évaluation de la qualité de clustering déjà vus dans la section précédente, permettent de juger la qualité des clusters selon certaines caractéristiques qui donnent des indications sur ceux-ci, par exemple la compacité et la séparabilité. Également, ces critères peuvent servir de guide pour choisir le nombre de clusters idéal pour le jeu de données selon un critère donné.

Une autre notion peut être également utilisée comme un critère de qualité et de validité. C'est la stabilité des clusters [72], elle peut être définie comme la variation entre deux résultats de clustering produits à partir de différents échantillonnages des données. Il existe de différents mesures de variations peuvent être utilisées pour obtenir une évaluation de la stabilité. Par exemple, dans les algorithmes à base de modèle (les centroïdes pour k-means) la distance entre deux modèles trouvés sur différents échantillons peut être utilisée comme mesure de stabilité. D'autre coté, La notion de validation croisée utilisée en apprentissage supervisé peut également être utilisée pour étudier la stabilité des solutions.

III.5.4. le choix du meilleur algorithme de clustering

Malgré qu'il existe des critères d'évaluation de la qualité ou de la stabilité permettant d'aider l'expert à choisir l'algorithme de clustering le plus adéquat. Généralement dans le cas de classification non supervisée, il est relativement difficile de faire un choix parmi de nombreuses méthodes de clustering, surtout quand l'expert a peu de connaissances sur ses données.

Pour répondre à ce problème, les auteurs dans [73] ont comparé des algorithmes entre eux par rapport aux résultats qu'ils produisent. Pour se faire, les auteurs ont utilisé 35 algorithmes de clustering différents sur 12 jeux de données. Les résultats produits ont ensuite été comparés, et les méthodes de clustering ont été regroupées en fonction de la similarité des résultats produits. Cette étude, qui ne donne pas de réponse à la question : Quel est le meilleur algorithme?, permet tout de même d'obtenir des informations sur les méthodes ayant sensiblement le même comportement.

D'autres études plus théoriques d'algorithmes de clustering ont également été menées en tentant de comparer les fonctions objectives des algorithmes. D'après [74], il est nécessaire de faire la différence entre méthode de clustering et algorithme de clustering. Une méthode de clustering est une stratégie générale employée pour résoudre un problème de clustering. Un algorithme de clustering est alors une instance d'une de ces méthodes.

Cependant, toutes ces études arrivent à la même conclusion : il n'existe pas de méthode de clustering meilleure que toutes les autres, autrement dit, qu'aucun algorithme de clustering ne satisfait de manière simultanée un ensemble d'hypothèses de base. C'est-à-dire, un algorithme de clustering parfait est censé de répondre à un ensemble de critères. Du fait que certains de ces critères sont contradictoires, il n'est pas possible de les satisfaire tous en même temps.

III.6. les algorithmes de clustering

De nombreux algorithmes de clustering ont été proposés dans la littérature. Ces algorithmes peuvent être classifiés selon plusieurs critères, le type de données à traité, le critère groupant définissant la similarité entre les individus, la théorie ou les concepts fondamentaux sur lesquels les techniques de clustering sont basées…etc. Ainsi selon la méthode adoptée pour définir les clusters, les algorithmes peuvent être largement groupés dans deux grandes familles les algorithmes hiérarchiques et les algorithmes par partitionnement. Néanmoins, d'autres algorithmes particuliers ont été proposés cités ci-après.

III.6.1. les algorithmes hiérarchiques

Les approches hiérarchiques produisent une séquence de partitions emboîtées d'hétérogénéités croissantes de la plus fine à la plus grossière, conduisent à des résultats sous forme d 'arbre hiérarchique indicé connu aussi sous le nom de dendrogramme. Chaque nœud contient ses clusters enfants, et les nœuds frères partitionnent les objets contenus dans leurs parents. Ce type d'approche permet d'explorer les données à différents niveaux de granularité. Les algorithmes hiérarchiques sont décomposés en deux types :

1) ***Les algorithmes agglomératifs (ascendants) :*** démarrent avec autant de clusters que d'individus puis fusionnent successivement l'ensemble des clusters selon un certain critère jusqu'à ce que tous les individus soient finalement regroupés dans un unique cluster stocké à la racine de la hiérarchie. Le schéma d'un algorithme hiérarchique ascendant est le suivant :
    1. Les clusters initiaux sont les individus eux-mêmes.
    2. On calcule les distances entre les clusters.
    3. Les deux clusters les plus proches sont fusionnés et remplacée par un seul.
    4. Le processus reprend en 2 jusqu'à n'avoir plus qu'un seul cluster, qui contient toutes les individus.

    Un algorithme agglomératif fonctionne donc en recherchant à chaque étape les clusters les plus proches pour les fusionner, et l 'étape la plus importante dans l 'algorithme réside dans le choix de la distance entre deux clusters. De nombreuses distances ont été définies. Cependant, Les algorithmes les plus classiques définissent la distance entre deux clusters à partir de la mesure de dissimilarité entre les individus constituant chaque cluster. Parmi les algorithmes les plus connus développé récemment, on a BIRCH [3], CURE [4] et CHAMELEON [5] et ROCK [75].

2) ***Les algorithmes divisifs (descendants) :*** Démarrent avec un unique cluster regroupant l'ensemble des individus, puis divisent successivement les clusters selon un certain critère jusqu'à ce que les individus se retrouvent dans des clusters différents stockés aux feuilles de la hiérarchie. Les algorithmes descendant hiérarchique sont itératifs et procèdent à chaque itération au choix du cluster de l'arbre hiérarchique à diviser, et au partitionnement de ce cluster. La différence entre les algorithmes divisifs développés dans la littérature, figure dans les critères qu'elles utilisent pour choisir le cluster à diviser ainsi dans la manière dont elles divisent le

cluster. Le choix de tels critères dépend généralement de la nature des variables caractérisant les individus à classer. Différentes propositions de ce type d'algorithme ont été proposées dans [76], [77], [78].

III.6.2. Les algorithmes de partitionnement

Les algorithmes de partitionnement cherchent la meilleure partition en k clusters disjointes des données, le nombre de classes (clusters ou groupes) k étant fixé a priori. Les algorithmes de partitionnement utilisent un processus itératif en fonction du nombre k qui consiste à affecter chaque individu au cluster le plus proche selon une distance ou un indice de similarité en optimisant une certaine fonction objectif . Les algorithmes K-means et K-medoid sont les plus connus dans cette famille d'algorithme. Dans notre travail nous somme intéressée par ce type d'algorithme pour faire un partitionnement intelligent de la base de données. Afin d'étudier ces algorithmes, nous détaillerons dans la section suivante les différents algorithmes de partitionnement.

III.6.3. Les algorithmes basés sur la densité

Le principe des algorithmes basés sur la densité est de définir un cluster comme étant un ensemble d'individus de forme quelconque, mais "dense selon un critère de voisinage et de connectivité é. C es algorithmes sont fondées sur les concepts de densité , noyau , point limite, accessibilité e t connectivité . les algorithmes les plus utilisés sont, CLIQUE[79], MAFIA[80], DBSCAN[81]et DENCLUE[82].

III.6.4. les algorithmes basés sur les graphes

Les algorithmes basé sur les graphes consiste à former le graphe connectant les individus entre eux et dont la somme des valeurs des arcs, correspondant aux distances entre les individus, est minimale, puis à supprimer les arcs de valeurs maximales pour former les clusters. Certains de ces algorithmes sont basés sur l'idée de considérer les voisins les plus proches d'un point tel que, Shared Nearest Neighbors Clustering [83] et Mutual Nearest Neighbor Clustering [84]. Néanmoins, il existe d'autres travaux proposés dans [85], [86], [87] basés sur le principe de graphe.

III.6.5. Les algorithmes basés sur les grilles

Les algorithmes basés sur les grilles utilisent une grille pour partitionner l'espace de description des individus en différentes cellules, puis identifie les ensembles de cellules denses connectées pour former les clusters. Parmi les proposions qui représentent cette catégorie d'algorithme on peut citer, STING [88]et WaveCluster [89].

III.6.6. Les algorithmes basés sur un modèle

Les algorithmes basés sur un modèle permettre de fournir de bonnes approximations des paramètres du modèle, afin que ce dernier s'ajuste au mieux aux données. Ils sont toutefois plus proches des algorithmes basées sur la densité, en ce sens qu'ils donnent naissance à des classes particulières qui améliorent un certain modèle préconçu. Dans ce cadre, nous pouvons

distinguer deux familles d'approches : les approches neuronales [90]et les approches probabilistes [91], [92].

III.7. Les algorithmes de partitionnement en Clustering

Contrairement aux algorithmes hiérarchiques, les algorithmes de partitionnement subdivisent l'ensemble d'individus où chacun est attaché à un seul cluster en employant une stratégie d'optimisation itérative dont le principe général est de générer une partition initiale, puis de chercher à l'améliorer en réattribuant les individus d 'un cluster à l 'autre. Ces algorithmes recherchent donc à optimiser une fonction objectif traduisant le fait que les individus doivent être similaires au sein d 'un même cluster, et dissimilaires d'un cluster à un autre. Les clusters de la partition finale, sont alors d 'intersection vide et chacun est représenté par un prototype qui peut être un individu de la population ou un point de l'espace selon l'algorithme utilisé.

Les premiers algorithmes de partitionnement proposés, k-means [6] et k-medoids [93]. Les deux algorithmes sont basés sur l'idée que le centre représente le cluster. Pour k-means on utilise la notion de centroîd, qui est la moyenne ou la médiane de l'ensemble d'individus. Notant que dans la plupart des cas le centroîd ne correspond pas à un individu. Dans K-medoid, on utilise la notion de medoid, qui est l'individu le plus représentatif (centré) dans le cluster, sachant que le medoid est un individu actuel dans le cluster. Néanmoins, il existe d'autres algorithmes qui proposent de nombreuses améliorations de ces deux algorithmes et qui seront discutées dans cette section.

III.7.1. l'algorithme K-means

L'algorithme k-means [6] est la plus connu et très utilisée. Son principe est très simple, dans la première étape, elle consiste à tirer aléatoirement k individus de la population représentants les centres provisoires des k clusters qui formeront la partition initiale. Ensuite, les autres individus sont regroupés autour de ces k centres en affectant chacun d'eux au centre le plus proche. L'étape suivante consiste à recalculer les k nouveaux centres (centroïdes ou centres de gravité) des k clusters, sachant qu'un centre n'est pas nécessairement un individu de la population. Le processus est répété plusieurs fois jusqu'à stabilité des centres des clusters. En pratique, l'algorithme k-means cherche à minimiser l'inertie interclasse définie par la somme des écarts des centroïdes aux points de leurs clusters et donc à maximiser aussi l'inertie interclasse de la partition donnée par la somme des écarts entre les centroïdes des clusters.

Soit k le nombre de clusters recherchés, l'algorithme k-means est illustré dans la figure suivante :

1. Choisir aléatoirement k individus de la base qui formeront les centroïdes initiaux représentants les k clusters recherchés ;

2. Assigner chaque individu au cluster dont le centroïdes est le plus proche ;

3. Puis tant qu'au moins un objet change de cluster d'une itération à l'autre :
   - Mettre à jour les centroïdes des clusters en fonction des objets qui leur sont associés ;
   - Mettre à jour les assignations des individus aux clusters en fonction de leur proximité aux nouveaux centroïdes ;

Figure III-4 : Algorithme k-means

L'avantage majeur de cet algorithme est sa simplicité mais aussi sa complexité algorithmique raisonnable. Tandis que cet algorithme a des limites, d'une part, le calcul de moyenne qu'il utilise est très sensible aux points aberrants et restreint leur application aux données numériques. D'autre part, le problème du choix des centres initiaux qui influe sur la partition finale obtenue. De plus, l'algorithme k-means ne produit ainsi que des clusters de forme hyper sphérique et n'est pas capable de gérer des clusters proches dont les tailles sont différentes.

Pour pallier ces limites, des solutions ont été proposés. La première est celle de Mac-Queen [94], selon cette approche les centres sont recalculés après chaque affectation d'un individu dans un cluster, plutôt que d'attendre l'affectation de tous les individus avant de mettre à jour les centres. Ce qui a conduit à des meilleurs résultats, ainsi la convergence est plus rapide. Pour l'application de k-means sur les données catégoriques, un nouvel algorithme a été proposé k-modes [95], [96] et qui définit une nouvelle mesure de dissimilarité pour les attributs catégoriques. Ensuite, l'algorithme k-prototypes [97] qui introduit une nouvelle mesure de similarité pour les données mixtes (numériques et catégoriques), notant que les deux algorithmes calculent dynamiquement les centres, c.-à-d. après chaque affectation d'un individu les centres sont recalculés.

D'autres perfectionnements de k-means ont été introduits dans [98], dont ils ont utilisé la structure k-dtree pour améliorer la rapidité de calcul. Egalement, l'algorithme global k-means [99], utilise la structure k-dtree et exécute l'algorithme k-means N fois (N est le nombre d'individus) en modifiant les centres des clusters pour obtenir des meilleurs résultats et accélérer le calcul.

D'autre part, Le choix des centres initiaux dans l'algorithme k-means est une étape critique. Pour répondre à ce problème, il est possible de faire plusieurs exécutions de l'algorithme K-means avec différents ensembles de centres initiaux et choisir celle qui donne une bonne partition.

III.7.2. l'algorithme k-medoids

L'objectif de l'algorithme K-medoıds [93] est de trouver un ensemble de clusters non chevauchés, de sorte que chaque cluster a un point le plus représentatif, c'est-à-dire, un point qui le plus centré au sein du cluster selon certaine mesure. Par exemple, la distance. Ces points représentatifs sont appelés medoıds. Les étapes de l'algorithme k-medoids sont illustrées dans la figure suivante :

1. Sélectionnez K individus initiaux. Ces individus sont les medoıds candidats et sont destinées à être les individus les plus centraux de leurs clusters ;
2. Remplacer l'un des individus sélectionnés (medoids) par l'un des individus non sélectionnés : La distance de chaque individu non sélectionné au medoıd candidat le plus proche est calculé, et cette distance est additionnée sur tous les individus. Cette distance représente le coût de la configuration actuelle. Tous les échanges possibles d'un individu non sélectionné pour un individu sélectionné (medoid) sont prisent en compte, et le coût de chaque configuration est calculé ;
3. Sélectionnez la configuration avec le coût le plus bas. S'il s'agit d'une nouvelle configuration, répétez alors l'étape 2 ;
4. Dans le cas contraire, associer chaque individu non sélectionné avec son individu sélectionné (medoıd) le plus proche et arrêter.

Figure III-5 : L'algorithme k-medoids

À l'étape 3, le medoid i est évalué par la formule suivante: $\sum_{j=1}^{n_i} p_{ij}$

Où p_{ij} est la distance entre le medoid i et l'individu j dans le cluster. Pour la similarité (dissimilarité), la somme doit être la plus grande (petite) que possible. Sachant que cet algorithme n'est pas limité aux espaces euclidiens et est susceptible d'être plus tolérant aux valeurs aberrantes. Cependant, trouver le meilleur medoid nécessite d'évaluer tous les individus qui ne sont pas actuellement medoıds ce qui demande un cout d'exécution très élevé.

L'algorithme PAM (Partitioning Around Medoids) [100] est un algorithme de clustering à base de k-medoıds qui tente à regrouper un ensemble d'individus en k clusters en effectuant les étapes de l'algorithme k-medoids. D'après PAM, les k medoids initiaux sont choisis aléatoirement. Ensuite, l'algorithme examine tous les couples d'individus tels que l'un est un medoid et l'autre un non medoid, en évaluant si l'échange du deux individus permet d'améliorer une fonction objective. L'échange qui permet la plus grande amélioration de la fonction objectif est réalisé et une nouvelle itération a alors lieu. A la fin, les différents individus sont affectés aux clusters des medoids les plus proches. Cet algorithme présente l'avantage de pouvoir utiliser tout type de distance, puisqu'il n'est pas nécessaire de définir la moyenne des individus. Cependant, elle représente une grande complexité algorithmique.

Afin de minimiser le cout de calcul qui est très élevé avec l'algorithme PAM, ils ont introduit une adaptation de PAM pour le traitement des grands ensembles de données nommée CLARA (Clustering LARge Applications) [100]. Cet algorithme effectue une recherche locale des medoids en opérant sur plusieurs échantillons de données de taille fixée. L'algorithme PAM est appliqué à chacun d'entre eux et le résultat retenu est le meilleur parmi les différents résultats. En conséquence, CLARA peut traiter des jeux de données de taille beaucoup plus importante que ceux traités par PAM. Suite à des expérimentations, Les auteurs ont indiqué que 5 échantillons de tailles respectives 40+2k individus suffisent pour donner un bon résultat. Par contre, l'efficacité de CLARA dépend de la taille et de la forme d'échantillons considérés.

Un autre algorithme proposé, dit CLARANS (Clustering Large Applications based upon RANdomized Search)[101] et qui combine les deux algorithmes PAM et CLARA. L'algorithme CLARANS utilise un graphe de classification pour résoudre le problème de recherche des k meilleurs medoids. Chaque sommet de ce graphe représente k individus (k medoids). Deux sommets sont dits voisins si leurs ensembles ne se diffèrent que par un seul individu. En conséquence, le problème de déterminer un ensemble de k meilleurs medoids devient le problème de recherche dans ce graphe du meilleur sommet. Le critère pour estimer qu'un sommet est meilleur qu'un autre est d'assurer une amélioration d'une fonction objectif (la somme des carrés des distances aux medoids) par l'échange d'un medoid par un individu non medoid suite au déplacement d'un sommet à un autre voisin. Cependant, PAM parcourt tous les sommets voisins d'un sommet, ce qui devient très coûteux. CLARA opère sur une région localisée en parcourant un sous graphe, ce qui ne donne pas toujours le bon résultat. En fait, CLARANS est une combinaison des deux algorithmes. Plutôt que d'essayer tous les échanges entre chaque medoid et chaque non medoid, CLARANS choisit aléatoirement un sommet dans le graphe et n'examine pas tous les voisinages. Il détermine un nombre de voisins aléatoires (pas tous les voisins, il s'agit d'un paramètre de l'algorithme) pour rechercher un voisin dont le coût de remplacement de ce sommet par son voisin est minimisé. Si ce coût est négatif, c.à.d. le remplacement ne peut plus optimiser le résultat, l'algorithme stipule que le sommet trouvé assure le meilleur résultat, et le processus peut s'arrêter. Les classes obtenues par CLARANS sont en général de meilleure qualité que celles obtenues par PAM et CLARA. Tandis que la seule limitation de cet algorithme est le nombre de paramètres à fixer a priori.

Les algorithmes de partitionnement cités dans cette section sont les plus populaires. Cependant, il existe de nombreuses approches proposées pour améliorer ces algorithmes. Après avoir étudié les algorithmes de partitionnement, nous approfondis notre recherche plus précisément sur les algorithmes de partitionnement parallèles, afin de les minés pour partitionner la base sur une grille de données parallèlement.

III.8. les algorithmes de partitionnement parallèles

Comme le cas pour la majorité des algorithmes du data mining, le problème majeur des algorithmes de partitionnement est le cout de calcul qui est très grand, surtout pour un grand jeu de données. Pour cela, le recours au parallélisme est parmi les solutions les mieux optés, et

plus particulièrement dans un environnement distribué. Notamment que les bases de données sont distribuées sur plusieurs sites. Dans ce cadre, il existe pas mal de solutions pour adapter les algorithmes de partitionnement dans un environnement parallèle et distribué.

III.8.1. Un algorithme de clustering sur Multiprocesseurs à Mémoires Distribuées

Les auteurs proposent une implémentation parallèle de l'algorithme du clustering k-means basée sur le modèle passage de messages (message passing). L'algorithme proposé [102] exploite le principe de parallélisme de données pour l'algorithme de k-means. Le but de cet algorithme est d'extraire et analyser des parties du texte des documents non structurés. Après des tests dans un environnement distribué, cette approche a prouvé une scalabilité et une rapidité de calcul notamment pour une grande masse de données et un nombre élevé de clusters.

III.8.1.1. L'algorithme k-means parallèle

La conception parallèle de l'algorithme est basée sur le modèle Un Seul Programme Multiples Données ou en anglais Single Program Multiple Data (SPMD) en utilisant le passage de messages qui est actuellement le modèle le plus répandu pour calculer sur les multiprocesseurs à mémoires distribuées.

Il est supposé qu'il existe P processeurs chacun avec une mémoire locale. Ces processeurs sont reliés en utilisant un réseau de communication. Une topologie spécifique d'intercommunication n'est pas exigée pour ce réseau, mais seulement un processeur accède à sa propre mémoire locale de la même qualité pour communiquer avec un autre processeur.

Le modèle passage de messages forme en principe un ensemble de processeurs qui ont des mémoires locales mais peuvent communiquer avec d'autres processeurs en envoyant et en recevant des messages. C'est la définition du modèle passage de messages, les transferts des données à partir de la mémoire locale d'un processeur à la mémoire locale d'un autre processeur, exigent des opérations effectuées par les deux processeurs.

Comme présenté dans l'algorithme ci-dessous, les auteurs dans [102] ont utilisé l'interface de passage de message MPI (Message Passing Interface) [103] pour la conception et l'implémentation de cet algorithme. Dans lequel, Chaque processeur est assigné par un identifiant unique entre 0 et P-1. Selon cet identifiant du processeur, chaque processeur peut suivre un chemin distinct d'exécution dans le même code. Ces processeurs peuvent communiquer avec d'autres en appelant des routines appropriées dans la bibliothèque de MPI qui encapsule les détails des communications entre les divers processeurs.

1. P=MPI_Comm_size() ;
2. μ=MPI_Comm_rank() ;
3. MSE=LargeNumber ;
4. **Si** (μ=0)**alors**
5. *Sélectionner les centre initiaux des clusters* $\{m_j\}_{j=1}^{k}$;
6. **Finsi**
7. MPI_Bcast($\{m_j\}_{j=1}^{k}$,0) ;
8. **Faire {**
9. OldMSE=MSE;
10. MSE'=0;
11. **Pour** j **de** 1 **à** k **faire**
12. m'_j=0 ; n'_j=0 ;
13. **FinPour**
14. **Pour** i de μ *(n /p)+1 **à** (μ+1) *(n /p) **faire**
15. **Pour** j **de** 1 **à** k **faire**
16. *Calculer la distance euclidienne carrée* $d^2(X_i, m_j)$
17. **FinPour**
18. *Trouver le centre le plus proche* m_l *à* X_i
19. $m'_l = m'_l + X_i$; $n'_l = n'_l + 1$;
20. MSE'= MSE'+$d^2(X_i, m_l)$;
21. **FinPour**
22. **Pour** j **de** 1 **à** k **faire**
23. MPI_Allreduce(n'_j, n_j, MPI_SUM);
24. MPI_Allreduce(m'_j, m_j, MPI_SUM);
25. $n_j = max(n_j, 1); m_j = m_j/n_j;$
26. **FinPour**
27. MPI_Allreduce(MSE', MSE, MPI_SUM);
28. **}Tantque** (MSE< OldMSE)

Figure III-6 : L'algorithme K-means parallèle

Les diverses routines de MPI employées dans la version parallèle de k-means sont résumé dans la table suivante :

ROUTINES MPI	DESCRIPTIONS
MPI_Comm_size()	*retourner le nombre de processeurs*
MPI_Comm_rank()	*retourner l'identificateur du processeur appelant*
MPI_Bcast(msg,root)	*diffusion en broadcasting des messages depuis le processeur root à tous les autres processeurs*
MPI_Allreduce(A,B,MPI_SUM)	*additionner toutes les copies locales de A dans tous les processeurs et placer le résultat dans B sur tous les processeurs.*

Tableau III-2 : Syntaxe et fonctionnalités des routines de MPI

Selon cet algorithme le parallélisme de k-means est conçu comme suit, un processeur choisit aléatoirement les centres initiaux (means) et les diffuse aux autre processeurs. Chacun de ces processeurs attribue les individus dans sa base de données locale aux centres les plus proches. Après, les processeurs changent l'information pour calculer les nouveaux centres.

Finalement, ce processus s'arrête quand une fonction objective est optimisée (la somme des distances).

III.8.2. Le Clustering K-Means Parallèle basé sur MapReduce

MapReduce [104] est un modèle de programmation et une exécution associée pour traiter et produire de grands ensembles de données. Aussi, il est adéquat à une large variété de tâches réelles. Les utilisateurs spécifient le calcul en termes de carte (Map) et fonction de réduction, et le système fondamental d'exécution parallélise automatiquement le calcul à travers des clusters à grande échelle de machines, gère les perdes des machines et les communications des programmes entre les machines pour une utilisation efficace du réseau et des disques.

Les auteurs adaptent l'algorithme k-means dans le cadre de MapReduce pour rendre l'algorithme de clustering applicable aux données à grande échelle. Proprement dit, Par l'application de la paire <clef, valeur>, l'algorithme proposé peut être exécuté efficacement en parallèle. Après des expériences complètes pour évaluer l'algorithme proposé. Les résultats démontrent que l'algorithme peut efficacement traiter des ensembles de données à grande échelle.

III.8.2.1. L'algorithme Pkmeans basé sur MapReduce

Selon l'algorithme k-means parallèle (PKMeans) [105] basé sur MapReduce, les calculs nécessaires peuvent être formalisés comme fonction de carte (Map-function) et fonction de réduction (reduce-function). La fonction de carte effectue la procédure d'attribution de chaque individu au centre le plus proche tandis que la fonction de réduction exécute la procédure de mise à jour des nouveaux centres. Afin de diminuer le coût de la communication du réseau, une fonction de combinaison (combine-function) est conçue pour faire face à la combinaison partielle des valeurs intermédiaires avec la même clé au sein de la même tâche de carte.

- **Map-function :** L'ensemble de données d'entrée est stocké sur HDFS [106] comme un fichier de séquence des paires <clef, valeur>, dont chacun représente un enregistrement (individu) dans l'ensemble de données. La clef est le décalage en bytes de cet enregistrement par rapport au point de début du fichier de données, et la valeur est une chaîne de caractères relative au contenue de cet enregistrement. L'ensemble de données est divisé et globalement diffusé à tous les machines dites mappers. Par conséquence, les calculs de distance sont exécutés en parallèle. Pour chaque tâche de carte (Map), PKMeans construit les centres globaux variables sous forme de tableau contenant les informations sur les centres des clusters. Compte tenu de l'information, un mapper peut calculer le centre le plus proche pour chaque échantillon. Les valeurs intermédiaires sont alors composés de deux parties: l'index du centre le plus proche et l'information sur l'échantillon. Le pseudo-code de Map-function est illustré dans la figure III-12.

Algorithme 1. *map (clef, valeur)*

Entrée: Centres globaux variables, La clef, la valeur d'échantillon

Sortie : la paire *<clef', valeur'>*, où *clef'* est l'index du centre le plus proche et *valeur'* une chaine de caractères contenant l'information d'échantillon

```
1. Construire l'exemple d'échantillon à partir de la valeur;
2. minDis = Double.MAX_VALUE;
3. index = -1;
4. Pour i=0 à centers.length faire
        dis= ComputeDist(instance, centers[i]);
        Si dis < minDis {
                minDis = dis;
                index = i;
                }
5. Fin Pour
6. Prendre l'index sous forme clef';
7. Construire valeur' sous forme de chaine de caractères contenant la valeur de différentes dimensions ;
8. Sortie  paire <clef', valeur'>r;
9. Fin
```

Figure III-7 : Map-function

Notons que les étapes 2 et 3, initialisent les variables minDis et index, l'étape 4 calcul le centre le plus proche pour l'échantillon, d'où la fonction ComputeDist(instance, centers[i]) retourne la distance entre l'échantillon et le centre centers[i]; l'étape 8 produit les données intermédiaires qui seront employées dans les procédures suivantes.

- **Combine-function :** Après chaque tâche de carte, une combinaison de données intermédiaires de la même tâche de carte est appliquée. Étant donné que les données intermédiaires sont stockées dans le disque local de l'hôte, la procédure ne demande pas un coût de communication. Dans cette fonction, on additionne partiellement les valeurs des points assignés au même cluster. Afin de calculer la valeur moyenne des points pour chaque cluster, il faut enregistrer le nombre d'échantillons dans le même cluster dans la même tâche de carte. Le pseudo-code pour cette fonction est illustré dans la figure III-13.

Algorithme 2. *combine (clef, V)*
Entrée: *clef* est l'index du cluster, V est la liste des échantillons assignés au même cluster
Sortie: la paire < *clef', valeur'*>, où clef' est l'index du cluster, valeur' contenant la somme des échantillons dans le même cluster et le nombre d'échantillons
1. Initialiser un tableau pour enregistrer la somme des valeurs pour chaque dimension des échantillons contenants dans le même cluster, les échantillons dans la liste V;
2. Initialiser le compteur *num* à 0 pour enregistrer le nombre d'échantillons dans le même cluster;
3. **Tant que** (V.hasNext()){
Construire l'exemple d'échantillon de *V.next();*
Ajouter les valeurs de différent dimensions d'instance au tableau ;
num++;
4. }
5. prendre Clef sous forme clef';
6. Construire valeur' sous forme de chaine de caractères contenant la somme des valeurs des diffèrent dimensions et le *num*.
7. **sortie** la paire < clef', valeur'>;
8. **Fin**

Figure III-8 : Combine-function

- **Reduce-function :** L'entrée de cette fonction est les données obtenues à partir de Combine-function pour chaque hôte. Comme décrit dans la fonction combine, les données incluent la somme partielle des échantillons dans le même cluster et le nombre d'échantillons. Dans Reduce-function, on peut additionner tous les échantillons et calculer tout le nombre d'échantillons assignés au même cluster. Par conséquence, on peut obtenir les nouveaux centres qui seront utilisés pour la prochaine itération. Le pseudo-code pour cette fonction est illustré dans la figure III-14.

Algorithme 3. reduce (clef, V)
Entrée: clef est l'index du cluster, V est la liste des sommes partielles des échantillons à partir des différents hôtes.
Sortie la paire < *clef', valeur'>,* où clef' est l'index du cluster, valeur' est une chaine de caractères qui représente le nouveau centre
1. Initialiser un tableau pour enregistrer la somme des valeurs pour chaque dimension des échantillons contenants dans le même cluster, les échantillons dans la liste V;
2. Initialiser le compteur NUM à 0 pour enregistrer le nombre d'échantillons dans le même cluster;
3. **Tant que** (V.hasNext()){
Construire l'exemple d'échantillon de V.next();
Ajouter les valeurs de différent dimensions d'instance au tableau ;
NUM += num;
4. }
5. Diviser les entrées du tableau par NUM pour obtenir à nouveau les coordonnées du centre ;
6. prendre Clef sous forme clef';
7. Construction valeur' sous forme d'une chaine de carctères comportent des coordonnées du centre ;
8. **Sortie:** la paire < *clef', valeur'>;*
9. **Fin**

Figure III-9 : Reduce-function

III.8.3. Un Algorithme de Clustering Parallèle pour les données Catégoriques

Dans cette contribution, les auteurs dans [107] proposent un algorithme de clustering k-mode parallèle sur des multiprocesseurs à mémoire distribuée c.-à-d., sur des machines parallèles. Spécifiquement, la version parallèle de k-mode est basée sur le modèle passage de message (message passing) pour un calcul parallèle. Pour tester cette approche, les auteurs ont examiné l'algorithme k-mode parallèle sur des données réelles, dont les résultats montrent un grand gain de temps d'exécution par rapport à la version séquentiel et pour une grande échelle.

III.8.3.1. L'algorithme k-Modes parallèle

L'algorithme k-mode séquentiel cité dans la section précédente a été conçu pour traiter des données catégoriques. Inspirant de cet algorithme, k-modes parallèle proposée est basé principalement sur le modèle Un Seul Programme Plusieurs Données (SPMD) utilisant la bibliothèque MPI pour calculer sur des multiprocesseurs à mémoire distribuée. Une comparaison entre la version séquentielle et parallèle de k-mode est montrée dans la figure suivante :

1. P=MPI_Comm_size() ;
2. μ=MPI_Comm_rank() ;
3. COST=LargeNumber ;
4. **Si** (μ=0)**alors**
5. *Sélectionner les modes initiaux* $\{m_j\}_{j=1}^{k}$;
6. **Finsi**
7. MPI_Bcast($\{m_j\}_{j=1}^{k}$) ;
8. **Répéter {**
9. OldCOST=COST;
10. COST'=0;
11. $q_{j,l}(\omega) = 0(\omega \epsilon \Omega_l, 1 \leq l \leq d, 1 \leq j \leq k)$
12.
13. **Pour** i de μ *(n /p)+1 **à** (μ+1) *(n /p) **faire**
14. **Pour** j **de** 1 **à** k **faire**
15. *Calculer la dissimilarité* $d(X_i, m_j)$
16. **FinPour**
17. *Trouver le centre le plus proche m_{j*} à X_i*
18. COST'= COST'+$d(X_i, m_{j*})$;
19. **Pour** *l* **de** 1 **à** *d* **faire**
20. $q_{j*,l}(X_{i,l}) = q_{j*,l}(X_{i,l}) + 1$
21. **FinPour**
22. **FinPour**
23.
24. *q'=q ;*
25. **Pour** j **de** 1 **à** k **faire**
26. **Pour** *l* **de** 1 **à** *d* **faire**
27. **Pour** *tous les* $\omega \epsilon \Omega_l$ **faire**
28. MPI_Allreduce($q'_{j,l}(\omega)$, $q_{j,l}(\omega), MPI_SUM$);
29. **FinPour**
30. $\boldsymbol{m_{i,l} = argmax}(q_{j,l}(\omega): \omega \epsilon \Omega_l)$**;**
31. **FinPour**
32. **FinPour**
33. MPI_Allreduce(COST', COST, MPI_SUM);
34. **} jusqu'à** (COST ≥ Old COST)

Figure III-10 : Algorithme de k-mode Parallèle

Les routines de MPI utilisées dans la version parallèle de k-modes sont décrites dans la table III-2. Par l'implémentation de la version k-modes parallèle sur des machine distribuées avec P processeurs, il est prévu de réduire le temps de calcul par presque un facteur P. une stratégie simple de parallélisation est de diviser n points de données sur P blocs (chacune de la taille n/P) et calculer les lignes 14-22 (figure III-10) pour chacun de ces blocs en parallèle sur des processeurs différents. Où, chacun des processeurs peut effectuer les calculs de dissimilarité d'une façon asynchrone, si les modes $\{m_j\}_{j=1..K}$ sont disponibles dans chaque processeur. Pour permettre ceci, une copie locale des modes $\{m_j\}_{j=1..K}$ est maintenu dans chaque processeur.

Sous cette stratégie de parallélisation, chaque processeur doit traiter seulement les données n/P, et par conséquence, le temps de calcul pour k-modes parallèle diminue. Autrement dit, il est compté que la charge de calcul sera partagée également sur tous les P processeurs. Cependant, il existe également un coût attaché à cet avantage, c'est le coût de communication. Avant que chaque nouvelle itération de k-mode pourra débuter, tous les P processeurs doivent communiquer pour recalculer les nouveaux modes $\{m_j\}_{j=1..K}$. Cette communication globale (synchronisation) est représentée par les lignes 25-32 (figure III-10). Comme une conclusion, chaque itération de l'algorithme k-modes parallèle se compose d'une phase asynchrone de calcul suivie d'une phase synchrone de communication.

III.8.4. Algorithme de clustering k-means parallèle sur un réseau de stations de travail

Cette approche [108] est conçue pour être utilisée dans un réseau de stations de travail homogènes avec le réseau Ethernet et utilisant le passage de messages pour la communication entre les processeurs. En se basant sur le modèle un Seul Programme Multiple Données, Les auteurs ont utilisé l'architecture maître-esclave. Les expérimentations de cette approche montrent une efficacité de 50% pour le temps d'exécution par rapport à la version séquentielle de k-means. Également, la version parallèle permet d'utiliser de plus grandes tailles de données à cause de sa simplicité de calcul et sa nature distributive. L'approche parallèle proposée est donné dans la figure ci-dessous, elle est basée sur le principe de l'algorithme k-means.

Processus Maitre

1. Former aléatoirement K sous ensemble égaux de l'ensemble S
2. Envoyer chaque sous ensemble à chaque K esclaves
3. Recevoir K sous ensemble résultat par K esclaves.

Processus Esclave

1. Recevoir le vecteur du sous ensemble P du processus maitre.
2. Tant que l'Erreur E est non stable
3. Calculer la moyenne $\overline{X}_{myrank}$ du sous ensemble P
4. Diffuser la moyenne $\overline{X}_{myrank}$ aux autres esclaves
5. Calculer la distance d(i, j), $1 \leq i \leq K, 1 \leq j \leq |P|$ pour chaque vecteur dans P sachant que $d(i,j) = \left\|\overline{X}_i - v_j\right\|$
6. Choisir les vecteurs membres des K nouveaux sous-ensembles selon leur distance la plus proche.
7. Diffuser K sous ensemble calculés dans l'étape 6 à tous les autres esclaves.
8. Former le nouveau sous-ensemble P par la collection des vecteurs qui appartiennent à $\overline{X}_{myrank}$ et qui ont été envoyés par les autres esclaves dans l'étape 7.
9. FIN
10. Envoyer le sous ensemble P au processus Master.

Figure III-11 : l'algorithme k-means distribué

Comme le montre cet algorithme, le maitre divise la base de données en k partitions et les envoie aux esclaves. Alors, chaque esclave calcul la moyenne (mean) de sa partition et la diffuse à tous les autres esclaves. Ensuite, au niveau de chaque esclave la distance entre chaque individu de sa partition et chaque centre (mean) est calculée en assignant chaque individu au centre le plus proche. Les esclaves partagent l'information et forment les nouvelles partitions (clusters). A la fin, le maitre reçoit les nouveaux clusters envoyés par les esclaves.

III.8.5. L'algorithme CLARANS parallèle

Cet algorithme [109] est une amélioration de CLARANS pour l'utiliser dans un environnement distribué. Les auteurs ont utilisé le mécanisme PVM (Parallel Virtual Machine) pour la mise en œuvre de cet algorithme. Les résultats montrent une accélération dans le temps d'exécution et des résultats optimaux comparant avec CLARANS séquentiel, notamment pour un nombre élevé de clusters et d'individus. Ainsi, un tel algorithme parallèle est plus efficace dans un réseau distribué.

Les performances de CLARANS sont fortement influencées par les deux paramètres, le nombre maximal de voisins (maxneighbor) et le nombre local (numlocal). Si les deux paramètres étaient grands, un clustering optimal demande un cout d'efficacité plus élevé. D'un autre coté, si ces deux paramètres sont faibles, les résultats de clustering ne peuvent pas être garantis. Ainsi, afin d'assurer un résultat de clustering optimal par l'algorithme CLARANS, les deux paramètres sont souvent grands.

L'idée principale est que l'algorithme CLARANS a un bon intuitif au parallélisme. C'est-à-dire, le processus de recherche des k medoıds dans le graphe peut être réalisé avec une concurrence. Dont, le traitement de chaque medoıd est une sous-tâche parallèle typique, alors que la qualité des medoıds résultants peut être exécutée en même temps. Subséquent, l'algorithme CLARANS parallèle est amélioré comme suit :

Entrer les paramètres *numlocal* et *maxineighbor*
Initialiser *mincost* à un grand nombre
Pour j **de** 1 **à** *numlocal* **faire**
Sélectionner aléatoirement un nœud comme nœud courant C dans le graphe ;
Assigner des tâches aux nœuds M.

a) Choisir au hasard un voisin N de C; Et calculer la différence de coût de deux nœuds
b) **Si** Cost(N, D) < Cost(C, D) **alors**
Assigner N comme nœud courant C;
Pendant ce temps, le diffuser à d'autres nœuds.
Lorsque les nœuds reçoivent le message, ils renouvellent la liste des ***modoids*** selon ce dernier.
c) Lorsque les nœuds T renvoient les résultats, comparez le coût du courant avec *mincost*. Si le premier est inférieur à *mincost*, fixer *le mincost* au coût du courant et définir le meilleur nœud au courant

Fin pour
Sortie meilleur nœud et arrêt

Figure III-12 : L'algorithme CLARANS parallèle

Le coût d'exécution pour cet algorithme est le coût de calcul pour toutes les paires d'individus O_i et O_j dont O_i est actuellement sélectionné et O_j non sélectionné. Ce sont les voisins de O_i, où O_j est dans chaque nœud. Alors, si les voisins de O_i sont calculée simultanément dans chaque nœud le résultat de clustering sera optimale et le temps de calcul sera diminué.

III.9. Conclusion

Ce chapitre a été consacré au clustering et plus précisément au clustering de partitionnement parallèle et distribué. En premier lieu, une étude sur les différents concepts de base pour le clustering a été faite, les différents types de clusters, les introductions distinctes de distances pour différents types de données et l'évaluation de la qualité du clustering. Également, les problèmes majeurs de clustering sont cités et discutés dans ce chapitre. Pour avoir une idée sur les algorithmes de clustering, nous avons donné un aperçu sur les types de clustering existants. Le but de notre étude est de faire un partitionnement intelligent de la base de données dans un environnement distribué. Pour cela, nous avons approfondi les recherches sur les algorithmes de partitionnement pour distribuer d'une façon optimale la base de données sur k sites. Cependant, malgré la simplicité de ces algorithmes, ces derniers demandent un cout de calcul et un temps d'exécution énormes, et plus particulièrement pour les grandes bases de données. D'un autre côté, pour bénéficier de l'environnement distribué, une étude ciblée sur les algorithmes de partitionnement parallèles et distribués a été faite pour concevoir une approche afin de répartir la base de données. Après une appréciation globale, nous avons jugé que l'algorithme k-prototypes est le plus convenable pour traiter des données réelles. Aussi, la parallélisation de l'algorithme peut être inspirée des différentes approches proposées pour concevoir un algorithme k-prototype parallèle et distribué.

Dans le chapitre suivant, nous présenterons notre contribution pour le partitionnement parallèle de la base de données sur plusieurs sites. Ainsi, l'approche proposée pour l'extraction de règles d'association à partir d'une base de données distribuée est détaillée.

CHAPITRE IV : APPROCHES PROPOSÉES

IV.1. Introduction

Actuellement, les bases de données sont très volumineuses et distribuées sur plusieurs sites. Alors, l'extraction de règles d'association à partir de ces bases de données est devenue une tache très couteuse en terme, temps d'exécution et complexité de calcul. Pour cela, le recours au parallélisme est indispensable pour l'extraction de règles d'association. Nous avons abordé plusieurs techniques d'extraction de règles d'association séquentielles et parallèles dans le chapitre II. Généralement, les problèmes majeurs de ces algorithmes est le nombre élevé de balayage des données et les étapes multiples de synchronisations et de communication qui dégrade leurs performances. Ces hypothèses nous ont amené à considérer l'algorithme séquentiel Partition comme algorithme intéressant, ce dernier ne demande que deux parcours de la base. Nous avons donc pensé à améliorer l'algorithme Partition pour réduire le nombre de balayage de la base en un seul, ensuite concevoir une version parallèle pour l'algorithme amélioré.

D'autre part, pour optimiser les performances de cet algorithme. Le partitionnement intelligent de la base s'avère intéressant en utilisant le clustering, et plus précisément le clustering distribué. A la lumière de l'état de l'art présenté au chapitre précédent, nous avons considéré de plus près les algorithmes de partitionnement séquentiels et parallèles, qui nous semblent les plus appropriés pour partitionner ou distribuer la base de données sur plusieurs sites d'une façon homogène. Nous proposons également un algorithme de clustering parallèle basé sur l'algorithme k-prototypes. Cet algorithme semble le plus convenable pour traiter les données réelles qui contiennent des valeurs numériques et catégoriques.

Le reste de ce chapitre est structuré comme suit : dans une première partie, nous présenterons d'abord l'algorithme d'extraction de règles d'association proposé dans un environnement distribué. Nous détaillons l'algorithme Partition, nous introduisons par la suite, les perfectionnements apportées à cet algorithme et le parallélisme de l'algorithme Partition amélioré. La section suivante est consacrée à l'algorithme de clustering parallèle. En premier lieu, nous détaillons l'algorithme k-prototype, ensuite, nous montrons les modifications apportées à k-prototypes pour concevoir un algorithme k-prototypes parallèle.

IV.2. l'algorithme d'extraction de règles d'association parallèle

L'objectif des algorithmes parallèles et distribués pour l'extraction des règles d'associations est de réduire le coût de communication. Généralement, ces algorithmes souffrent d'un nombre élevé de balayage des données et par conséquence plusieurs étapes de synchronisations et de communication qui dégradent leurs performances.

Afin de remédier ces inconvénients nous avons choisi l'algorithme séquentiel Partition [14], et nous avons été intéressé par 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.

IV.2.I. l'algorithme Partition

Partition est parmi les algorithmes proposés pour améliorer l'algorithme Apriori. l'approche adaptée par cet algorithme est de diviser la base de données en partitions horizontales de même taille où la taille de chaque partition dépend de la taille de la mémoire de telle sorte qu'une partition entière peut résider dans la mémoire afin qu'elle soit lue qu'une seule fois dans chaque phase, dépend aussi de la taille moyenne des transactions, le seuil actuel de support minimal et le nombre d'items. Il est exécuté séquentiellement 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. La figure suivante présente les deux phases de l'algorithme :

1) P= base de donnée (D) partitionnée
2) n= nombre de partitions
   // Phase I
3) **for** i =1 **to** n **do begin**
4) lire-dans-partition ($p_i \in P$)
5) L^i = gen-large-itemsets(p_i)
6) **end**
   // Phase de fusion
7) **for** (i=2; $L_i^j \neq \emptyset$, j =1,2,…, n; i++) **do**
8) $C_i^G = \cup_{j=1,2,\ldots,n} L_i^j$
9) **end**
   // Phase II
10) **for** i=1 **to** n **do begin**
11) lire-dans-partition ($p_i \in P$)
12) Pour tous candidats $c \in C_i^G$ gen-final-count (c, p_i)
13) **end**
14) L^G= { $c \in C_i^G$ | c.count ≥ minSup }
15) Sortie = L^G

Figure IV-1 : L'algorithme Partition

IV.2.1.1. Phase I

Prend *n* itérations, pendant l'itération *i*, seule la partition p_i est considérée. La fonction *gen_large_itemsets* illustrée dans la figure IV-2 prend la partition p_i 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 de la base de données)

1) L_1^p= {les 1-itemsets frequents avec leurs tidlists }
2) **for** (k=2 ; $L_k^p \neq \emptyset$; k++) **do begin**
3) **for all** itemsets $l_1 \in L_{k-1}^p$ **do begin**
4) **for all** itemsets $l_2 \in L_{k-1}^p$ **do begin**
5) **if** $l_1[1] = l_2[1] \wedge l_1[2] = l_2[2] \wedge \ldots \wedge l_1[k-1] < l_2[k-1]$ **then**
6) $c = l_1[1].l_1[2] \ldots l_1[k-1].l_2[k-1]$
7) **if** c ne peut pas être élagué **then**
8) c.tidlist= l_1 .tidlist ∩ l_2 .tidlist
9) **if** $|c.\text{tidlist}| / |p| \geq$ minSup **then**
10) $L_k^p = L_k^p \cup \{c\}$
11) **end**
12) **end**
13) **end**
14) **return** $\cup_k L_k^p$

Figure IV-2 : La fonction gen_large_itemsets

Les transactions sont sous la forme : <TID , i_j , i_k , ...i_n> , Les éléments d'une transaction sont supposés être triés dans l'ordre lexicographique, Les items dans un itemset sont également conservés triés selon cet ordre.

Pour déterminer à la fois les itemsets fréquents pour chaque partition, et compter les supports globaux au cours de la phase comptage, l'algorithme de partition emploi une nouvelle représentation de données TID-liste, TID-liste associe à chaque itemset X une liste de tous les TID des transactions où cet itemset apparait. Autrement dit la base de données est transformée en paires <X, TID-liste>.

Les TID-liste sont triées d'une façon pour que leur intersection puisse être calculée de manière simple dans la phase de jointure, qui nécessite qu'un seul parcours des deux listes. D'autre part, elles permettent l'élimination de toutes les données inutiles, parce que les TID-listes d'items non-fréquents peuvent être supprimés facilement. L'exemple de la figure IV-3 illustre la représentation d'une base de données initiale et les résultats intermédiaires, le support minimum est équivalent à 3 occurrences.

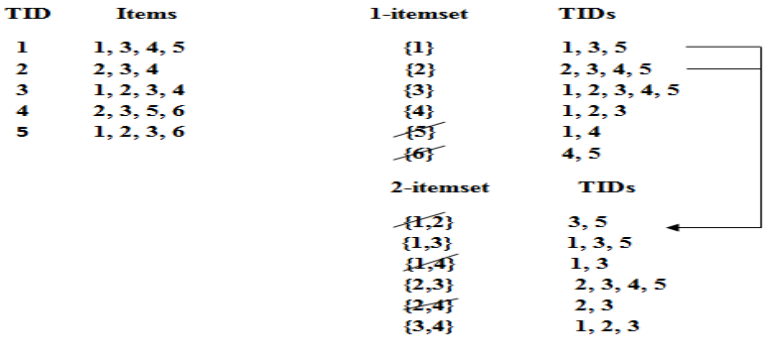

Figure IV-3 : Représentation TIDs et TID-Liste

Après la génération des itemsets locaux fréquents, la phase de fusion réuni les itemsets de même longueur de toutes les partitions n pour générer l'ensemble des itemsets candidats globaux. L'ensemble d'itemsets candidat globaux de longueur j est calculé comme suit :

$$C_j^G = \bigcup_{i=1\ldots n} L_j^i$$

IV.2.1.2. Phase II

La procédure *gen_final_count* décrite dans la figure IV-4 met en place un compteur pour chaque itemset candidat global, parcourt la base de données une deuxième fois et génère les TID-listes des 1-items. La procédure réalise les intersections entre les TID-listes des items dans l'itemset candidat, donc la cardinalité de l'intersection est le support de l'itemset candidat dans la partition.

Procédure **gen-final-count** (C^G : ensemble de candidats globaux, p : partition de la base de données)

1) **forall** 1-itemsets do
2) générer son tidlist
3) **for** (k=2 ; $C_k^G \neq \emptyset$, k ++) **do begin**
4) **forall** k-itemset $c \in C_k^G$ **do begin**
5) *templist* = c[1].tidlist ∩ c[2].tidlist ∩ …∩ c[k].tidlist
6) *c.count* = *c.count* + $|templist|$
7) **end**
8) **end**

Figure IV-4 : La procédure gen_final_count

IV.2.2. l'algorithme Partition parallèle

Suivant notre approche la topologie adaptée est vue comme un ensemble de *N* sites (clients) gérés par un site appelé le coordinateur. Considérons que 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'est à dire chaque site disposera d'un ensemble de transaction (partition), où chacun va se charger d'identifier l'ensemble d'itemsets fréquent localement pour générer les itemsets fréquents globaux.

Afin d'optimiser l'algorithme parallèle et pour faire un seul scan de la base de données, nous proposons d'utiliser les calculs de la première phase pour compter les fréquences des itemsets candidats globaux dans la deuxièmes phase. L'opinion est de garder les TID-listes des 1-itemsets calculés dans la première phase au niveau de chaque site. Cependant, la procédure *gen-final-count* dans la deuxième phase compte les supports des itemsets candidats globaux par l'intersection des TID-listes des items appartenant à l'itemset candidat et ceci sans parcourir la base de données une deuxième fois pour produire les TID-listes des 1-itemsets, ce qui permet de minimiser la complexité de calcul. Cette notion nous soumet à redéfinir la procédure qui calcule la fréquence des itemsets candidats globaux, dont elle est surnommée *count-candidats-final* :

Procedure **count-candidat-final** (*c* : candidat global, *p* : partition de la base de données)

1) *templist* = c[1].tidlist ∩ c[2].tidlist ∩ …∩ c[k].tidlist
2) *c.count* = *c.count* + $|templist|$
3) **return** *c.count*
4) **end**

Figure IV-5 : La fonction count-candidat-final

L'avantage dans la version parallèle est que chaque site peut stocker les TID-listes des 1-items pour sa partition et les charger dans la mémoire pour le traitement dans la deuxième phase. Par contre, dans la version séquentielle et pour une base de données volumineuse les TID-listes des 1-items de la partition actuelle sont effacés pour une exécution suivante (partition suivante). Ainsi, il est impossible de traiter à la fois les TID-liste de tous les 1-items pour une base de données à haute dimension.

IV.2.2.1. la version parallèle de Partition

Selon une architecture centralisée et reposant sur les modifications apportées à l'algorithme Partition pour une exécution parallèle, nous avons proposé une approche pour l'extraction de règles d'association en parallèle dans un environnement distribué. L'approche proposée utilise au niveau de chaque site la fonction *gen-large-itemsets* et la fonction *count-candidats-final* pour générer les itemsets locaux fréquents et calculer les fréquences locales des itemsets candidats globaux respectivement. Alors que le coordinateur se charge de cumuler les données envoyées par les sites. La figure IV-6 montre le rôle du coordinateur et les différents sites dans chaque phase de l'algorithme.

Entrée : Base de données D, le nombre de sites N, le support minimal min-sup

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 … N **Faire en Parallèle**

$L^{(i)}$ = ***gen-large-itemsets*** (p_i) // trouver l'ensemble d'itemsets fréquents locaux ;

Site : Envoie $L^{(i)}$ 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 tous les sites ;

PHASE 3 : //Compte les supports des candidats globaux

Pour chaque Site I= 1 … N **Faire en Parallèle**

Pour chaque $c \in C^G$ **Faire**

$c.compte^{(i)}$ = ***count-candidats-final*** (c, p_i);

Site : Envoie les résultats au coordinateur ;

FinPour

Fin de Parallèle

PHASE 4 : //Le coordinateur calcul les supports globaux des candidats globaux

Coordinateur : Pour chaque $c \in C^G$ **Faire**

$c.compte = \sum_{i=1}^{N} c.compte^{(i)}$ //combiner les comptes;

Return $L^G = \{c \in C^G | \ c.compte \ / |D| \geq \min_sup\}$;

FinPour

Figure IV-6 : L'algorithme Partition parallèle

Les quatre phases de l'algorithme Partition parallèle décrits les taches réalisées par chaque site :

Phase 1 : chaque site 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 $L^{(i)}$, ensuite il envoie le résultat au coordinateur. Comme la taille des données locales dans tous les sites est approximativement égale, cette phase prendra un temps à peu près égal dans tous les sites en réalisant un équilibrage de charge. En plus, seulement les données disponibles localement sont traitées. Par conséquent, il n'y a

aucun coût de communication pour la synchronisation ou le transfert de données entre les sites.

Phase 2 : Après avoir reçu le résultat de la phase 1, le coordinateur va se charger d'identifier l'ensemble des candidats globaux C^G, en faisant l'union de tous les ensembles d'itemsets fréquents localement de même taille pour chaque site (traitement réalisé dans la phase fusion de l'algorithme Partition), le résultat est diffusé à tous les sites. Après la fin de cette phase, tous les sites auront exactement le même ensemble d'itemsets candidats globaux.

Phase 3 : Correspond à la phase II de l'algorithme séquentiel, où chaque site détermine les fréquences locaux pour tous les itemsets dans C^G par l'application de la fonction count-candidat-final, ensuite les envoyer au coordinateur.

Phase 4 : Après avoir reçue le résultat de la phase 3, le coordinateur identifie l'ensemble d'itemsets fréquents globaux par la sommation des fréquences des itemsets sur chaque site.

On peut noter que l'algorithme partition parallèle ne demande que deux phases de synchronisation. On a la deuxième phase, pendant laquelle le coordinateur reçoit les itemsets fréquents locaux. Egalement pour la phase 4, le coordinateur doit attendre les fréquences des itemsets candidats globaux de tous les sites. Le schéma suivant illustre l'exécution parallèle de l'algorithme Partition dans un environnement distribué :

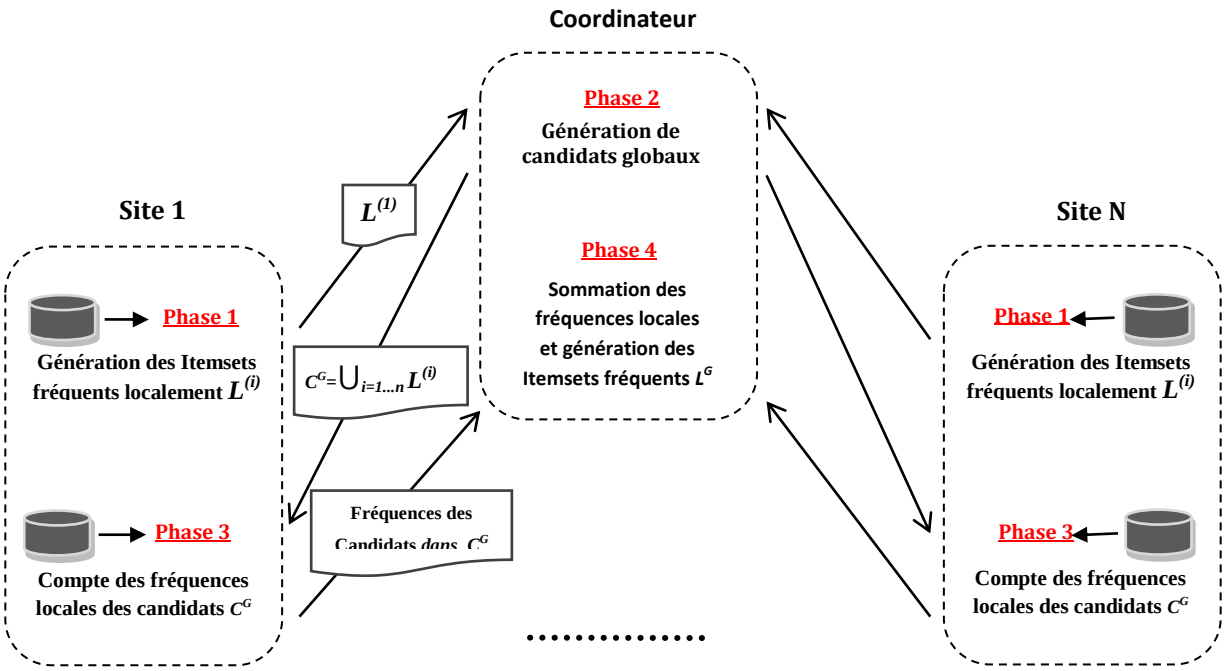

Figure IV-7: Schéma de l'algorithme Partition parallèle

IV.2.2.2. Exemple de déroulement de l'algorithme Partition parallèle

Considérons l'ensemble de données présenté dans la table suivante :

TID	Itemset
1	ABCD
2	DEF
3	BCD
4	ADEF
5	CDE
6	ACF

Tableau IV-1 : La base de données initiale

Cet exemple montre le fonctionnement de l'algorithme Partition parallèle sur deux sites Site1 et Site2, dont la base de données est partitionnée sur les deux sites comme suit :

Site1	
TID	**Itemset**
1	ABCD
2	DEF
3	BCD

Tableau IV-2 : La base de données dans le Site1

Site2	
TID	**Itemset**
4	ADEF
5	CDE
6	ACF

Tableau IV-3: La base de données dans le Site2

Phase 1 : Génération des itemsets fréquents localement (support minimal=50%)

1-Item	TID_liste	Fréquence
A	1	1
B	1, 3	2
C	1, 3	2
D	1, 2, 3	3
E	2	1
F	2	1

→

2-Items	TID_liste	Fréquence
BC	1, 3	2
BD	1, 3	2
CD	1, 3	2

→

3-Items	TID_liste	Fréquence
BCD	1, 3	2

Figure IV-8 : Génération des itemsets fréquents localement dans le Site1

1-Item	TID_liste	Fréquence
A	4, 6	2
C	5, 6	2
D	4, 5	2
E	4, 5	2
F	4, 6	2

→

2-Items	TID_liste	Fréquence
AC	6	1
AD	4	1
AE	4	1
AF	4, 6	2
CD	5	1
CE	5	1
CF	5	1
DE	4, 5	2
DF	4	1
EF	4	1

Figure IV-9: Génération des itemsets fréquents localement dans le Site2

Chaque site cherche les TID_listes des itemsets générés et élimine les itemsets qui ont un support inferieur au support minimal pour déduire les itemsets fréquents. Les figures IV-8 et IV-9 montrent les itemsets fréquents dans Site1(B, C, D, BC, BD, CD, BCD) et dans Site2(A, C, D, E, F, AF, DE).

Phase 2 : Génération des candidats globaux

	itemset
1-item	A
	B
	C
	D
	E
	F
2-items	AF
	BC
	BD
	CD
	DE
3-items	BCD

Tableau IV-4 : Génération des candidats globaux par le coordinateur

Le coordinateur reçoit les itemsets fréquents envoyés par le site1 et le site2, génère l'ensemble des candidats globaux de différentes tailles. (voir tableau IV-4)

Phase 3 : Compte des fréquences locales des candidats

Site1	
Itemset	**fréquence**
A	2
B	2
C	2
D	3
E	1
F	1
AF	0
BC	2
BD	2
CD	2
DE	1
BCD	2

Tableau IV-5 : Compte des fréquences des candidats globaux dans le Site1

Site2	
Itemset	**fréquence**
A	2
B	0
C	2
D	2
E	2
F	2
AF	2
BC	0
BD	0
CD	1
DE	2
BCD	0

Tableau IV-6 : Compte des fréquences des candidats globaux dans le Site2

Chaque site reçoit les candidats globaux envoyés par le coordinateur et calcule la fréquence locale pour chaque itemset par l'intersection des TID-listes des items constituants l'itemset (Voir Tableaux IV-5 et IV-6).
Par exemple, la fréquence de l'itemset CD dans le site1=2 et dans le site2=1.

Phase 4 : Génération des itemsets fréquents (support minimal=50%)

	Itemset	Fréquence globale
1-item	A	4
	B	2
	C	4
	D	5
	E	3
	F	3
2-items	AF	2
	BC	2
	BD	2
	CD	3
	DE	3
3-items	BCD	0

Tableau IV-7 : Les itemsets fréquents

Le coordinateur reçoit les fréquences des itemsets candidats globaux envoyés par le site1 et le site2. Il calcule les fréquences par la sommation des fréquences locales pour définir les itemsets fréquents (voir Tableau IV-7). Les itemsets qui ont un support supérieur ou égal au support minimal sont maintenus (C, D, E, F, CD, DE).

IV.3. l'algorithme de clustering parallèle et distribué

La recherche de règles d'association présente certains problèmes, parmi lesquels l'espace de recherche qui est très grand. Autrement dit, le nombre énorme d'itemsets recherchés, dont l'algorithme calcul les fréquences de toutes les combinaisons possibles pour les items existants dans la base de données. Ce problème persiste même pour les algorithmes parallèles pour l'extraction de règles d'association. Pour mieux interpréter l'idée, chaque partition de la base de données globale contient un nombre élevé d'items.

Pour pallier à ce problème, nous avons pensé à partitionner la base de données intelligemment, de façon à ce que chaque site contient les transactions qui ont des items similaires. Donc, les itemsets similaires seront regroupés dans le même site et non distribués sur plusieurs, permettant de minimiser l'espace de recherche. C'est à dire, le nombre d'itemsets recherchés est beaucoup plus moins. Ainsi, ceci réduit la complexité de calcul dans chaque site où la recherche des itemsets fréquents localement sera plus efficace et rapide.

Soit n le nombre de transactions dans une partition et m le nombre d'items, la complexité de la phase de génération des itemsets fréquents et donc de l'ordre O(n2m). Néanmoins, utilisant un partitionnement intelligent de la base, le nombre d'items dans chaque partition sera m` où m`<m. A cet effet, la complexité de calcul sera réduit à O(n2m`). Pour l'algorithme Partition, cela peut être expliqué par le nombre des 1-items fréquents trouvé dans chaque partition.

L'étape de codage pour traiter les données demande ainsi une étape de décodage pour interpréter les informations extraites comme le cas pour les règles d'association. Ce prétraitement et post-traitement demandent des calculs couteux. Pour faire face à ce critère,

nous avons opté l'algorithme k-prototypes qui traite les données mixtes (numériques et catégoriques), cet algorithme est basé sur le principe de k-means qui est le plus répondu pour la tache de clustering. Afin de concevoir un algorithme de clustering parallèle et distribué, nous avons apporté des modifications sur l'algorithme k-prototypes, et en inspirant du parallélisme proposé dans [102] basé sur le modèle passage de messages et sur une architecture décentralisée, nous proposons notre algorithme de clustering k-prototypes parallèle pour une architecture centralisée.

Pour exhiber notre contribution, nous présenterons dans ce qui suit l'algorithme k-prototypes et l'algorithme k-prototypes parallèle proposé.

IV.3.1. l'algorithme k-prototypes

Les algorithmes k-means sont efficaces pour traiter de grands ensembles de données, mais la limite principale pour ces algorithmes est qu'ils sont souvent limités aux données numériques. L'algorithme k-prototypes [97] basé sur le paradigme de k-means corrige cette limitation en préservant son efficacité. Dans k-prototypes, les individus sont groupés autour de k -prototypes. Ces derniers sont mis à jour dynamiquement (après chaque affectation d'un individu).

IV.3.1.1. les préliminaires mathématiques

- Soit X_i un individu et Q_l le prototype du cluster l, la mesure de similarité utilisée dans k-prototypes est définit par :

$$d(X_i, Q_l) = \sum_{j=1}^{m_r}(x_{ij}^r - q_{lj}^r)^2 + \gamma_l \sum_{j=1}^{m_c} \delta\,(x_{ij}^c, q_{lj}^c) \qquad (4.1)$$

Où $\delta(p,q) = 0$ pour $p=q$ *et* $\delta(p,q) = 1$ pour $p \neq q$. x_{ij}^r *et* q_{lj}^r sont les valeurs des attributs numériques, considérant que x_{ij}^c *et* q_{lj}^c sont les valeurs des attributs catégoriques pour l'individu i et le prototype du cluster l, m_r et m_c sont les nombres des attributs numériques et catégoriques. γ est le poids des attributs catégoriques pour cluster l.

De ce fait, le coût total pour l'assignement de X_i au cluster l est :

$$E_l = \sum_{i=1}^{n} y_{il} \sum_{j=1}^{m_r}\left(x_{ij}^r - q_{lj}^r\right)^2 + \gamma_l \sum_{i=1}^{n} y_{il} \sum_{j=1}^{m_c} \delta\left(x_{ij}^c, q_{lj}^c\right) = E_l^r + E_l^c \qquad (4.2)$$

Où $y_{il}=1$ indique que l'individu X_i est assignés au cluster l.
E_l^r et E_l^c sont les coûts totaux de tous les attributs numériques et catégoriques respectivement pour les individus du cluster l.

- Soit C_j l'ensemble contenant toutes les valeurs dans l'attribut catégorique j et $p(c_i \in C_j \backslash l)$ la probabilité d'occurrence de la valeur c_j dans le cluster l. E_l^c dans l'équation (4.2) peut être réécrite comme suit:

$$E_l^c = \gamma_l \sum_{i=1}^{m_c} n_l\,(1 - p(q_{lj}^c \in C_j \backslash l) \qquad (4.3)$$

Où n_l est le nombre d'individus dans le cluster l. la solution pour minimiser E_l^c est donnée par le lemme ci-dessous.

- ***Lemme 1 :*** Pour un spécifique cluster l, E_l^c est minimisé si et seulement si $p(q_{lj}^c \in C_j \backslash l) \geq p(c_j \in C_j \backslash l)$ pour $q_{lj}^c \neq c_j$ pour tous les attributs catégoriques.

- Finalement, la fonction de coût E est définit comme suit :

$$E = \sum_{l=1}^{k}(E_l^r + E_l^c) = \sum_{l=1}^{k} E_l^r + \sum_{l=1}^{k} E_l^c = E^r + E^c \qquad (4.4)$$

L'équation (4.4) est la fonction de coût du clustering pour l'ensemble de données avec des valeurs numériques et catégoriques. Puisque E^r et E^c ne sont pas négatifs, la minimisation de E sera achevée par la minimisation de E^r et E^c, le coût total des attributs numériques et catégoriques dans tous les clusters. E^r peut être minimisé par le calcul des éléments numériques des k prototypes par l'équation (4.5), par contre E^c peut être minimisée par la sélection des éléments des k prototypes suivant le lemme 1.
Par conséquent, l'équation (4.5) et le *Lemme1* définirent la façon de choisir les prototypes pour minimiser la fonction de coût.

- Pour un attribut numérique *j* le prototype est calculé par :

$$q_{lj} = \frac{1}{n_l}\sum_{i=1}^{n} y_{il} x_{ij} \ \ pour\ j = 1, \ldots\ldots, m \qquad (4.5)$$

Où $n_l = \sum_{i=1}^{n} y_{il}$ est le nombre d'individus dans le cluster *l*.

IV.3.1.2. l'algorithme

L'algorithme de k-prototypes peut être décrit comme suit :

(1) Sélectionner les k prototypes initiaux à partir de l'ensemble de données X, un pour chaque cluster.
(2) Assigner chaque individu dans X au cluster du prototype le plus proche selon l'équation (4.1). Mise à jour du prototype de cluster après chaque assignement.
(3) Après que tous les individus ont été assignés aux clusters, re-tester la similarité des individus aux prototypes courants. Si un individu est trouvé plus proche au prototype d'un autre cluster plutôt que son courant, réassigner l'individu à ce cluster et mettre à jour les prototypes des deux clusters.
(4) Répéter (3) jusqu'à ce qu'aucun individu n'ait changé de clusters après un test cyclique de X.

L'algorithme est établi sur trois processus, *choix initial de prototypes, attribution initiale, et réattribution.* Le premier processus sélectionne aléatoirement les k individus comme prototypes initiaux pour les clusters. Le deuxième processus est présenté sur la figure IV-10. À partir d'un ensemble de prototypes initiaux de cluster, ce processus assigne chaque individu à un cluster et met à jour le prototype de cluster après chaque assignement.

```
Pour i de 1 à NumberOfObjects Faire
        Mindistance= Distance(X[i],O_prototypes[1]) +  gamma* Sigma(X[i],C_prototypes[1])
        Pour j de 1 à NumberOfClusters Faire
                distance= Distance(X[i],O_prototypes[j])+ gamma * Sigma(X[i],C_prototypes[j])
                Si (distance < Mindistance) Alors
                        Mindistance=distance
                        cluster = j
                FinSi
        FinPour
        Clustership[i] = cluster
        ClusterCount[cluster] + 1
        Pour j de 1 à NumberOfNumericAttributes Faire
                SumInCluster[cluster,j] + X[i,j]
                O_prototypes[cluster,j]=SumInCluster[cluster,j]/ClusterCount[cluster]
        FinPour
        Pour j de 1 à NumberOfCategoricAttributes Faire
                FrequencyInCluster[cluster,j,X[i,j]] + 1
                C_prototypes[cluster,j]=HighestFreq(FrequencyInCluster,cluster,j)
        FinPour
FinPour
```

Figure IV-10 : Processus d'attribution initial

Dans la figure IV-10, *X[i]* représente l'individu *i* et *X[i,j]* la valeur de l'attribut *j pour l'*individu *i*. *O_prototypes[]* et *C_prototypes[]* stockent les attributs numériques et catégoriques des prototypes de clusters respectivement. *O_prototypes[i,j]* et *C_prototypes[i,j]* sont les deux éléments numériques et catégoriques du prototype pour le cluster *i*. *Distance()* est une fonction de distance euclidienne carrée et *Sigma()* est une implémentation de la fonction *δ()* dans l'équation (4.1). *Clustership[]* enregistre les clusters des individus et *ClusterCount[]* enregistre les nombres d'individus dans les clusters. *SumInCluster[]* additionne les valeurs numériques des individus dans les clusters, elle est utilisée pour mettre à jour les attributs numériques des prototypes de clusters. *FrequencyInCluster[]* enregistre les fréquences de différentes valeurs des attributs catégoriques dans les clusters. La fonction *HighestFreq()* est une implémentation du *lemme1* pour mettre à jour les attributs catégoriques des prototypes.

Le processus de réattribution donné sur la figure IV-11 est semblable au processus d'attribution initiale sauf qu'après la réattribution d'un individu, les prototypes de deux clusters précédent et courant de l'individu sont mis à jour. La variable moves enregistre le nombre d'individus qui ont changé de clusters dans le processus.

```
moves=0
Pour i de 1 à NumberOfObjects Faire
        …
        (Pour trouver le cluster dont le prototype est le plus proche pour l'individu i. comme montré dans la
        figure 54)
        …
        Si (Clustership[i]<>cluster) Alors
                moves+1
                oldcluster=Clustership[i]
                ClusterCount[cluster] + 1
                ClusterCount[oldcluster] - 1
                Pour j de 1 à NumberOfNumericAttributes faire
                        SumInCluster[cluster,j] + X[i,j]
                        SumInCluster[oldcluster,j] - X[i,j]
                        O_prototypes[cluster,j]=SumInCluster[cluster,j]/ClusterCount[cluster]
                        O_prototypes[oldcluster,j]= SumInCluster[oldcluster,j]/ClusterCount[oldcluster]
                FinPour
                Pour j de 1 à NumberOfCategoricAttributes Faire
                        FrequencyInCluster[cluster,j,X[i,j]] + 1
                        FrequencyInCluster[oldcluster,j,X[i,j]] - 1
                        C_prototypes[cluster,j]=HighestFreq(cluster,j)
                        C_prototypes[oldcluster,j]=HighestFreq(oldcluster,j)
                FinPour
        FinSi
FinPour
```

L'algorithme réitère le processus de redistribution jusqu'à ce que moves=0, ce qui indique que l'algorithme a stabilisé dans un optimum de l'équation (4.4).

IV.3.2. l'algorithme k-prototypes parallèle

L'algorithme k-prototypes parallèle proposé est fondé sur les deux processus de l'algorithme k-prototypes, processus d'attribution initiale et processus de réattribution. Ainsi, nous avons introduit des modifications sur l'algorithme k-prototype avant de concevoir la version parallèle. Ces modifications se révèlent sur le poids des attributs catégoriques et sur le procédé de calcule des nouveaux prototypes :

- Le poids des attributs catégoriques est identique pour tous les clusters, c'est-à-dire ; les poids des attributs numériques et catégoriques seront équilibrés pour chaque cluster. Donc, la mesure de similarité (l'équation (4.1)) peut être réécrite comme suit :

$$d(X_i, Q_l) = \sum_{j=1}^{m_r}(x_{ij}^r - q_{lj}^r)^2 + \gamma \sum_{j=1}^{m_c} \delta\,(x_{ij}^c, q_{lj}^c) \qquad (4.6)$$

Où γ est le poids des attributs catégoriques pour tous les clusters.

- Afin de minimiser le cout de communication et de synchronisation entre les sites et le coordinateur dans la version parallèle de k-prototypes, le calcul de nouveaux prototypes ce fait après l'affectation de tous les individus de la base de données. Autrement dit, chaque site affecte tous les individus aux clusters appropriés puis envoie l'information au coordinateur qui calcule les nouveaux k-prototypes.

IV.3.2.1. Le modèle de parallélisme

Nous avons proposé un modèle de parallélisme en inspirant de celui utilisé dans l'algorithme k-means parallèle présenté dans [102]. Cet algorithme utilise le modèle passage de message pour la communication entre les sites dans une architecture décentralisée. Où chaque site diffuse (broadcast) l'information à tous les autres sites pour un résultat global à chaque itération. Comme la diffusion générale à chaque itération (cout de communication) pénalise les performances de l'algorithme parallèle, nous avons adopté l'architecture centralisée. Dans la quelle chaque site envoie l'information à un seul site appelé le coordinateur pendant chaque itération. Alors, l'idée générale de parallélisme conçue s'interprète en six étapes :

(1) Le coordinateur envoie les k centres initiaux aux sites.
(2) Chaque site calcul la distance (mesure de similarité) entre les individus de sa base de données locale et les centres des clusters en affectant les individus aux clusters les plus proches.
(3) Chaque site envoie au coordinateur le nombre et la somme des individus pour chaque cluster.
(4) Le coordinateur calcul les nouveaux centres en fonction des données envoyées par les sites.
(5) Le coordinateur envoie les nouveaux centres aux sites.
(6) Répéter (2), jusqu'à ce qu'aucun individu ne change de cluster.

IV.3.2.2. la version parallèle de k-prototypes

Suivant le modèle de parallélisme présenté ci-dessus, nous avons proposé l'algorithme k-prototypes parallèle décrit dans la figure suivante :

Entrée : base de données D, nombre de clusters K
Sortie : les paires (k, O_i), où i est l'indice du cluster et O_i le $I^{ième}$ objets dans la base de données
PHASE 1: //Le coordinateur envoie les k prototypes aux sites
Coordinateur : Envoie $\{Prototypes[j]\}_{j=1..K}$
Iteration=0
Répéter
Compte=0
Iteration +
PHASE 2: //Chaque site affecte les individus aux clusters et envoie l'information au coordinateur
Pour chaque Site S= 1…K **Faire en Parallèle**
$Compte_S$=0

Pour i **de** 1 **à** NumberOfObjects **Faire**
Mindistance= ***Distance***(X[i],O_prototypes[1]) + gamma* ***Sigma***(X[i],C_prototypes[1])
Si (Iteration = 1) **Alors**
Clustership[i] = 1
ClusterCount[1] + 1
FinSi
Pour j **De** 1 **à** K **Faire**
distance= ***Distance***(X[i],O_prototypes[j]) + gamma* ***Sigma***(X[i],C_prototypes[j])
Si (distance < Mindistance) **Alors**
Mindistance=distance
cluster = j
FinSi
FinPour
Si (Clustership[i] <> cluster) **Alors**
$Compte_S$ +1
ClusterCount[Clustership[i]] - 1
Clustership[i] = cluster
ClusterCount[cluster] + 1
FinSi
Pour j **De** 1 **à** NumberOfNumericAttributes **Faire**
SumInCluster[Clustership[i],j] + X[i,j]
FinPour
Pour j **De** 1 **à** NumberOfCategoricAttributes **Faire**
FrequencyInCluster[Clustership[i],j,X[i,j]] + 1
FinPour
FinPour
Site: Envoie au coordinateur ($\text{ClusterCount}[i]^S_{i=1..k}$, $\text{SumInCluster}[i,j]^S_{i=1..k,j=1..\text{NumberOfNumericAttributes}}$,
$\text{FrequencyInCluster}[i,j,X[i,j]]^S_{i=1..k,j=1..\text{NumberOfCategoricAttributes}}$, $Compte_S$)
Fin de Parallèle
PHASE 3: // Le coordinateur calcul les nouveaux prototypes
Coordinateur :
Pour i **de** 1 **à** K **Faire**
Pour S **de** 1 **à** K **Faire**
ClusterCount[i] + $\text{ClusterCount}[i]^S$
FinPour
Pour j **de** 1 **à** NumberOfNumericAttributes **Faire**
Pour S **de** 1 **à** K **Faire**
SumInCluster[i,j] + $\text{SumInCluster}[i,j]^S$
FinPour
O_prototypes[i,j]=SumInCluster[i,j]/ClusterCount[i]
FinPour
Pour j **de** 1 **à** NumberOfCategoricAttributes **Faire**
Pour S **de** 1 **à** K **Faire**
FrequencyInCluster[i,j,X[i,j]] + $\text{FrequencyInCluster}[i,j,X[i,j]]^S$
FinPour
C_prototypes[i,j]=***HighestFreq***(FrequencyInCluster,i,j)
FinPour
FinPour
Pour S **de** 1 **à** K **Faire**
Compte + $Compte_S$
FinPour
Coordinateur : Envoie les nouveaux prototypes $\{\text{Prototypes}[j]\}_{j=1..K}$
Jusqu'à compte=0

Figure IV-12 : l'algorithme k-prototypes parallèle

L'algorithme k-prototypes parallèle montre les trois phases primordiales, il peut être formulé comme suit :

Phase 1 : c'est la phase la moins couteuse, avec laquelle le coordinateur sélectionne les k prototypes initiaux aléatoirement et les envoie aux autres sites.

Phase 2 : cette phase demande un cout de calcul très élevé, cependant elle s'exécute en parallèle dans chaque site et sur une partition de la base, ce qui permet de minimiser son cout d'exécution. Au premier lieu, chaque site calcul la distance entre chaque objet de sa base de données et chaque prototype par la formule (4.6) pour assigner les objets aux clusters les plus proches. Ensuite, chaque site envoie au coordinateur, les sommes des attributs numériques pour chaque cluster, le nombre d'objets dans chaque cluster, la fréquence pour chaque valeur de chaque attribut catégorique dans un cluster et le nombre d'objets qui ont changés de cluster (variable $Compte_s$).

Phase 3 : dans cette phase le coordinateur accumule les données envoyées par les sites, calcul le nombre d'objets pour chaque cluster i, par l'addition des nombres d'objets dans chaque site S (*ClusterCount[i]*^{s}*).* Ainsi, la somme de chaque attribut numérique j dans chaque cluster i est calculé par la somme des *SumIncluster[i,j]*^{s} dans chaque site S. Ensuite, le coordinateur calcul les nouveaux prototypes pour les variables numériques en fonction du nombre et de la somme des objets pour chaque cluster.

Pareillement pour les variables catégoriques, où les nouvelles valeurs catégoriques des prototypes sont déduites à partir des fréquences envoyés par les sites pour chaque cluster. Dont le coordinateur combine les fréquences de tous les sites (*FrequencyInCluster[]*^{s}) pour chaque valeur catégorique j et chaque cluster i.

Ce processus se répète à partir de la deuxième phase jusqu'à ce qu'aucun objet ne change de cluster. Le coordinateur vérifie cette condition utilisant les variables $compte_s$ envoyées par les sites.

Le schéma suivant résume l'algorithme k-prototypes parallèle :

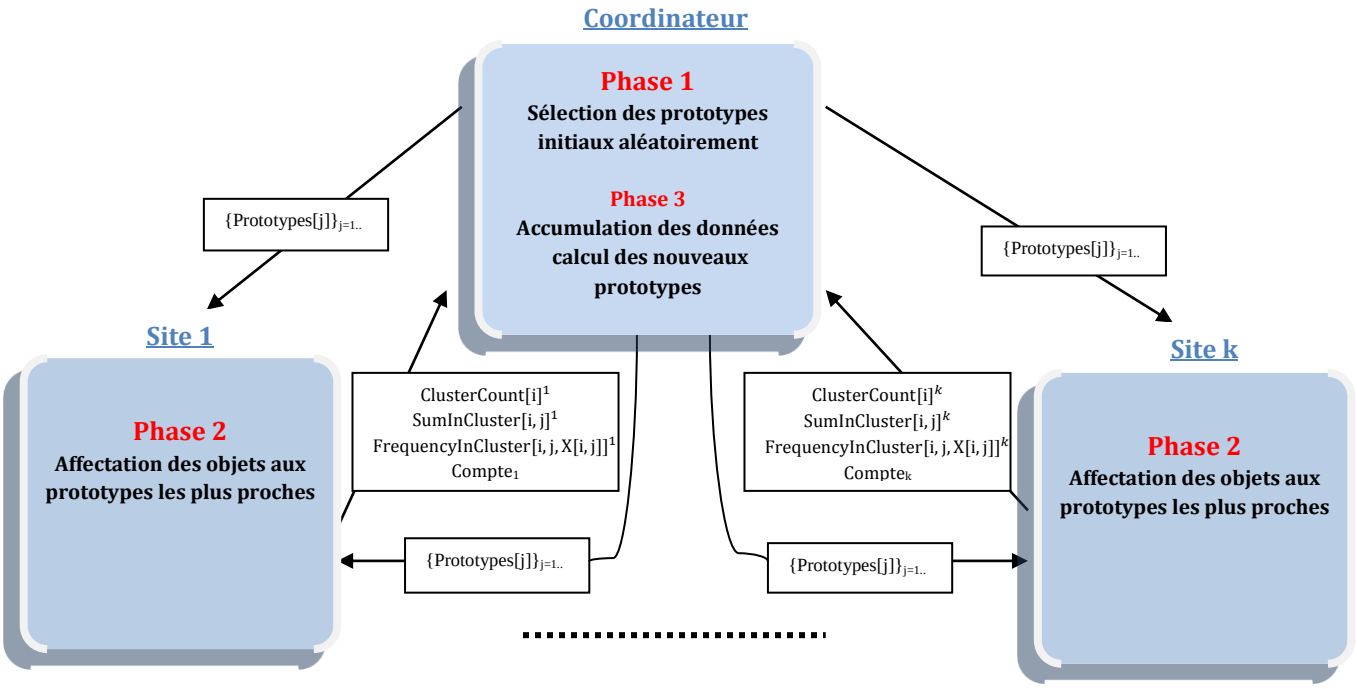

Figure IV-13 : Schéma de l'algorithme k-prototypes parallèle

IV.4. Conclusion

Dans ce chapitre, nous avons proposé une approche pour l'extraction de règles d'association. Cette approche est conçue pour être appliquée en parallèle sur des données distribuée sur plusieurs sites ou nœuds tel qu'une grille de données. L'approche est basée sur une distribution intelligente de la base sur les différents sites, ce prétraitement de la base est fait par un algorithme de clustering. Dans ce cadre, nous avons proposé un algorithme parallèle fondé sur l'algorithme k-prototypes. L'avantage de ce dernier est qu'il permet de traiter des données réelles contenants des valeurs numériques et catégoriques et ne demande pas la phase de codage, ce dernier n'est pas recommandé pour l'extraction de règles d'association et qui est le but de notre approche, vu que le codage nécessite une phase de décodage pour interpréter les règles extraites.

Après une distribution intelligente de la base dans une grille de données, nous appliquerons l'algorithme partition parallèle proposé pour générer les règles d'association. L'algorithme Partition a été optimisé pour parcourir la base une seule fois seulement, de la même, la version parallèle ne demande pas des phases multiples de communication et synchronisation.

Pour tester les performances de ces algorithmes, de multiples expériences seront faites pour mesurer le temps d'exécution achevé en changeant les diffèrent paramètres et en comparent avec d'autre algorithme parallèle. Ainsi, l'algorithme partition parallèle sera appliqué après un partitionnement aléatoire et intelligent de la base pour expérimenter l'influence du partitionnement intelligent sur l'activité et la rapidité de l'algorithme.

CHAPITRE V : RÉSULTATS, ANALYSES ET DISCUSSIONS

V.1. Introduction

Après une étude approfondie sur l'extraction de règles d'association parallèle, nous avons mené à proposer un algorithme nommé Partition parallèle conçu pour une architecture centralisée. Ainsi pour améliorer l'efficacité de l'algorithme Partition parallèle dans un environnement distribué, nous avons pensé à partitionner la base données intelligemment afin de minimiser l'espace de recherche dans chaque site. Pour cela, un algorithme appelé k-prototypes parallèle a été engendré.

Afin de conclure ces études et ces propositions, nous avons accompli des expérimentations diverses. Au premier lieu, nous avons étudié l'algorithme Partition parallèle en examinant ses performances. Dans un deuxième lieu, nous avons réalisé l'algorithme k-prototypes parallèle pour tester son effet sur l'algorithme Partition parallèle. Rappelons que la topologie du réseau choisi pour les deux algorithmes proposés est la topologie centralisée, c'est-à-dire un ensemble de sites gérés par un coordinateur.

Ce chapitre est organisé comme suit, nous décrirons d'abord l'environnement de travail (langage de programmation, simulateur …). Par la suite, nous présenterons les tests effectués qui sont divisés en deux parties : experiences1 et experiences2.

Dans experiences1, les performances de l'algorithme Partition parallèle sont comparées avec celles de l'algorithme Count Distribution sur une base de données numérique. Cependant, dans experiences2, l'algorithme k-prototypes parallèle est exécuté sur une base donnée mixte (contient des attributs numériques et catégoriques). Par la suite, l'algorithme Partition parallèle est considéré avec et sans le prétraitement de l'algorithme k-prototypes parallèle.

V.2. l'environnement de travail

Nous décrivons dans cette partie les outils utilisés pour l'implémentation des algorithmes proposés et décrits dans le chapitre précèdent. Comme simulateur, nous avons choisis GridSim qui répond à nos besoins, c'est une open source créée en JAVA. À cet effet, nous avons utilisé le compilateur JCreator.

V.2.1. JCreator

JCreator est un environnement de Développement Intégré pour Java créé par Xinox Software. Son interface est similaire à celle de Microsoft Visual Studio.

JCreator est un IDE (integrated development environment) performant pour Java. Il est l'outil de développement pour tous les programmeurs, plus rapide, plus efficace et plus fiable que

d'autres IDE Java. Par conséquent, il est l'outil idéal pour les programmeurs de tous niveaux, de débutant au spécialiste.
JCreator fournit à l'utilisateur un large éventail de fonctionnalités telles que: gestion de projet, des modèles de projet, l'auto complétion de code, l'interface du débogueur, éditeur avec coloration syntaxique, des assistants et une interface utilisateur entièrement personnalisable.

Avec JCreator on peut directement compiler ou exécuter notre programme Java sans activer le document principal en premier. JCreator est entièrement écrit en C++, ce qui le rend rapide et efficace par rapport à la rédaction basé sur Java/IDE.

JCreator a trois éditions:

- Lite Edition (LE)
- Pro Edition (Pro)
- Lite-Pro Edition (LE-PRO)

JCreator est disponible uniquement sur le système d'exploitation Windows. Toutefois, les versions LE et Pro de JCreator fonctionnent correctement sur Linux (en utilisant Wine).

V.2.2. Le simulateur GridSim

GridSim est un logiciel open-source de simulation du comportement d'un réseau, écrit en langage Java, généralement utilisé par les universitaires [110] . Cet outil de simulation a été conçu et construit principalement pour l'étude de l'efficacité des algorithmes de programmation.

GridSim est développé par *Australian Melbourne University*. C'est un simulateur consacré à l'ordonnancement des tâches dans les grilles. GridSim fournit un service complet pour la simulation de différentes classes de ressources hétérogènes, des utilisateurs et d'applications.

GridSim permet la flexibilité et l'extensibilité pour incorporer de nouveaux composants dans son infrastructure.

Sa simulation est contrôlée par un objet central (Sim-System) qui contrôle une file d'attente d'événements. Les interactions entre entités de GridSim sont implémentées en utilisant des événements (internes, externes, synchrones et asynchrones). Il est principalement employé pour étudier les algorithmes d'optimisation du coût d'ordonnancement des tâches dans les grilles, en offrant l'architecture essentielle aux modèles économiques avec prise en charge des contraintes de temps et du budget.

La boîte à outils GridSim fournit une installation complète pour la simulation de différentes catégories de ressources hétérogènes, les utilisateurs, les applications, les courtiers de ressources, et des planificateurs. Il peut être utilisé pour simuler les planificateurs d'application pour des systèmes informatiques distribués tels que les clusters et les grilles.

V.2.3. Configuration matériel

Pour la réalisation des expériences, un ordinateur de bureau a été utilisé avec :

- Un processeur Intel Corei7 avec vitesse d'horloges CPU 3.5GHz, 4.2 GHz.
- Mémoire vive RAM installée 8 Go.

Le système d'exploitation utilisé est Windows 7 (64 bits) édition intégrale.
Toutes les expériences sont développées sous JCreator.

V.3. Experiences1

Nous présentons dans cette section les résultats d'expérimentations sur notre algorithme Partition parallèle. Pour faire une étude comparative, nous avons ainsi mise en œuvre l'algorithme Count Distribution présenté dans le deuxième chapitre et qui est très populaire dans le domaine d'extraction de règles d'association parallèle.
Les deux algorithmes sont appliqués sur une base de données contenant 16 attributs numériques avec 70000 transactions.

V.3.1 Comparaison entre Count Distribution et partition parallèle

Comme première étape nous avons comparé le temps d'exécution pour les deux algorithmes, Partition parallèle et Count Distribution. En effet, nous avons varié les valeurs des trois paramètres : taille de la base de données, support minimal et le nombre de sites.

La figure V-1 presente la comparaison entre les algorithmes Partition parallele et Count Distribution, chaque graphe montre la difference en temps d'execution pour un nombre de site fixe et pour un support minimal qui varie entre 30%, 50% et 70%. Nous remarquons que le temps d'execution pour l'algorithme Count Distribution est plus élvé que l'algorithme Partition parallele surtout pour un support minimal plus petit (30%), cela peut etre expliqué par le fait que plus le support minimal est petit plus le nombre d'itemsets frequents est grand.

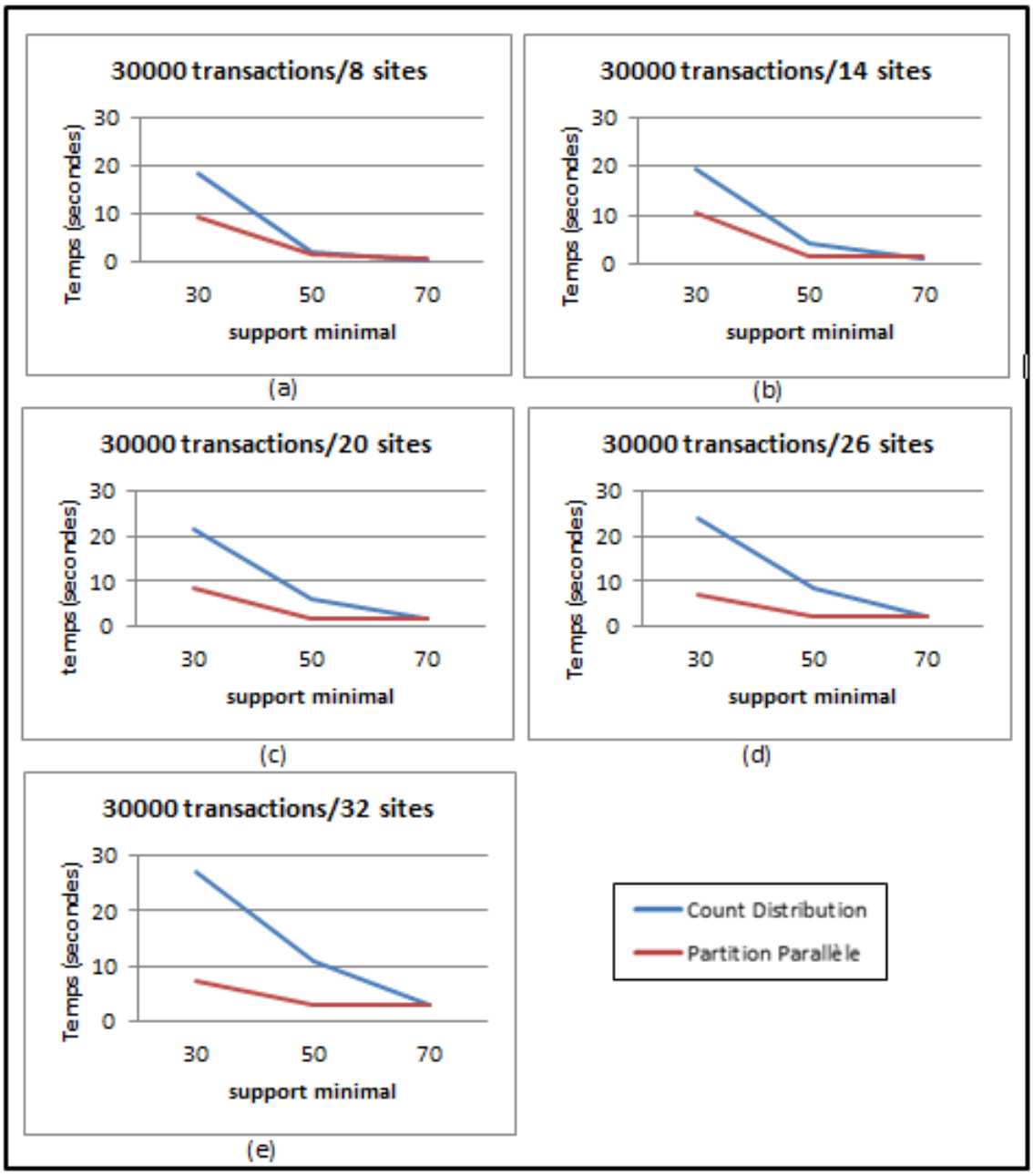

Figure V-1 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 30000 transactions)

De la même, nous remarquons que plus le nombre de site augmente, le temps d'execution pour l'algorithme Count Distribution est plus elvé, alors que pour l'algorithme Partition parallele se dégrade.

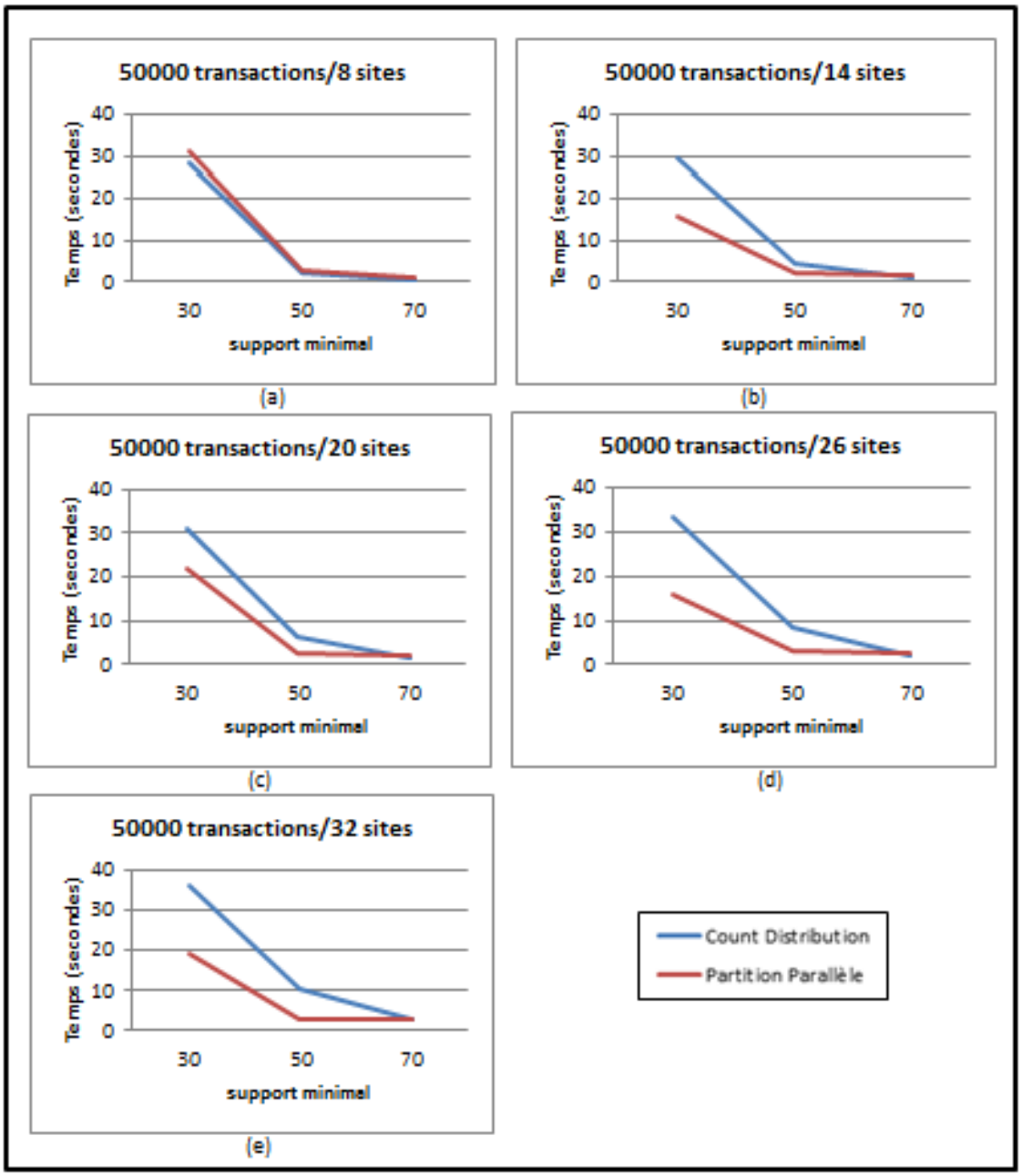

Figure V-2 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 50000 transactions)

La figure ci-dessus présente les même résultats de la figure V-1 mais pour une base de données plus grande (50000 transactions). Nous constatons que plus le taux de parallélisme est élevé, c'est-à-dire le nombre de site est grand, la différence en temps d'exécution entre les

deux algorithmes accroitre. Aussi, La figure V-2(a) qui présente les résultats pour un nombre plus petit des sites, montre un temps d'exécution légèrement identique pour les algorithmes.

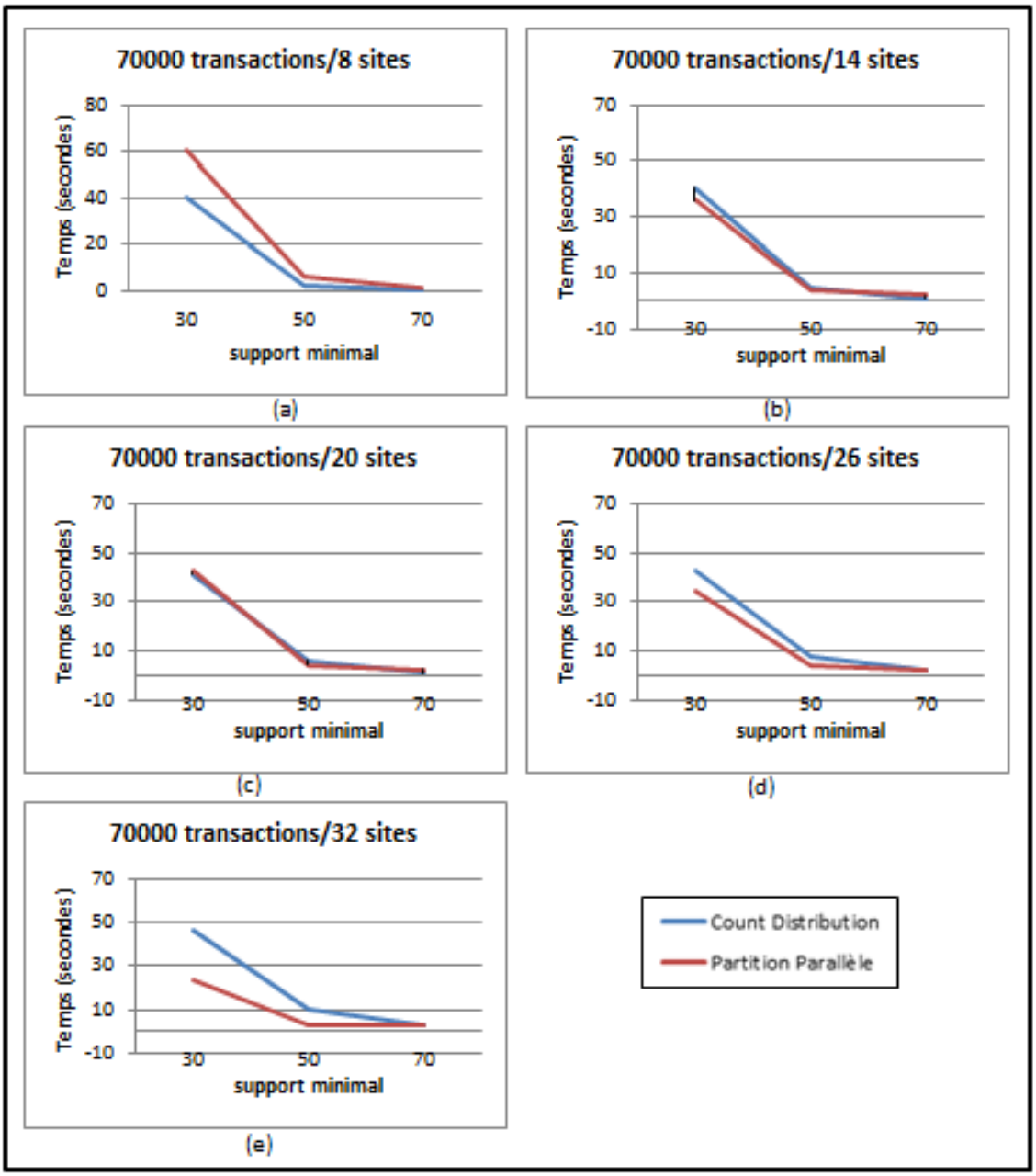

Figure V-3 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 70000 transactions)

La figure V-3 confirme l'influence du nombre de sites sur l'algorithme Partition parallèle et l'algorithme Count Distribution. Pour un nombre de transaction plus élevé (70000 transaction), la figure V-3(a) montre que l'algorithme Partition parallèle a achevé un temps

plus grand que l'algorithme Count Distribution. Alors, qu'à partir de la figure V-3(b) avec laquelle le nombre de cluster est 14 les résultats sont inversés.

Afin de résumer les résultats présentés précédemment et mieux comparer les deux algorithmes Partition parallèle et Count Distribution, nous avons fixé le support minimal à 30%. D'autre part nous avons varié le nombre de sites pour différentes taille de la base de données.

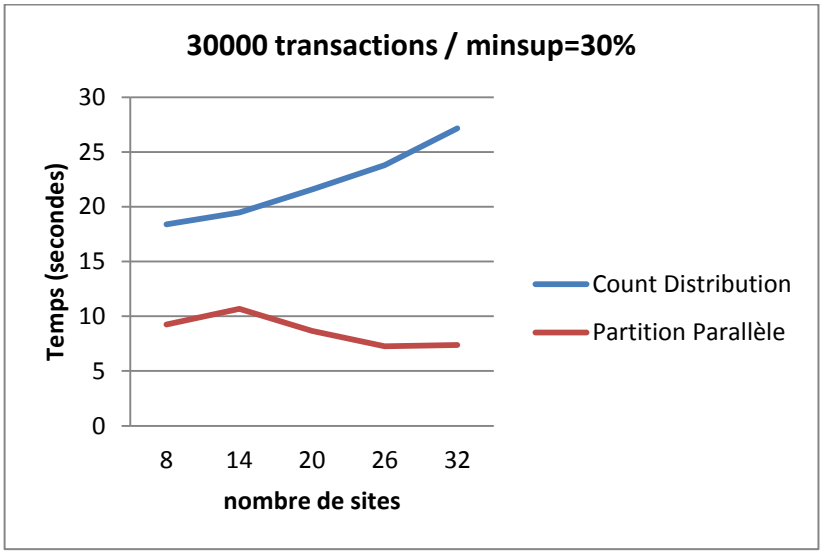

Figure V-4 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 30000 transactions, minsup =30%)

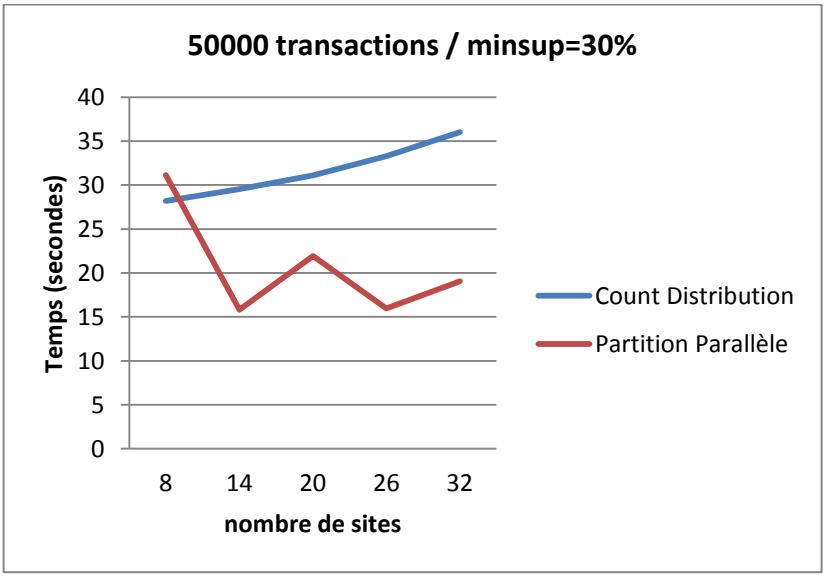

Figure V-5 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 50000 transactions, minsup =30%)

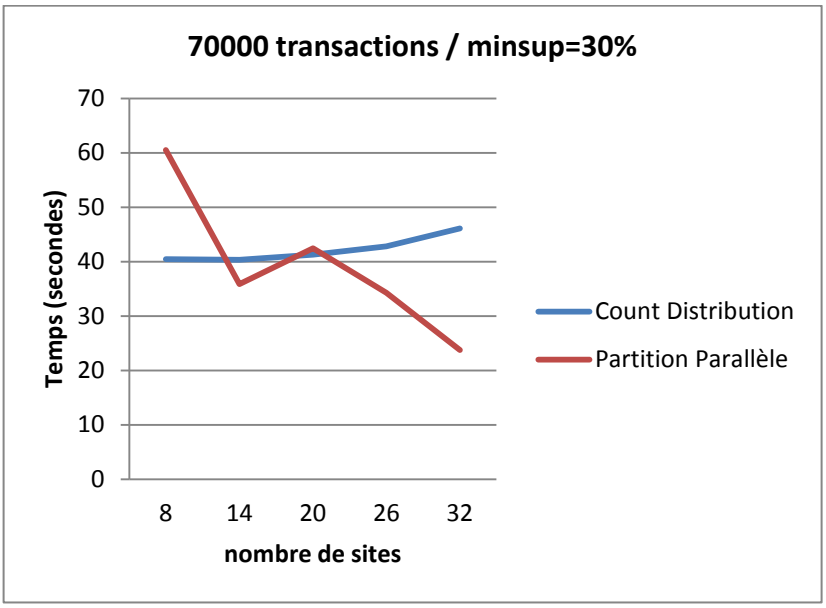

Figure V-6 : Comparaison du temps d'exécution entre Partition parallèle et Count Distribution (16 attributs, 70000 transactions, minsup =30%)

Suivant les trois figures plus haut, nous notons que pour l'algorithme Count Distribution le temps d'exécution augmente avec le nombre de sites. Ceci est suite au cout de communication et de synchronisation qui est indispensable à chaque itération, et quand chaque itération dépend de l'itération précédente ce cout accroitre pour un nombre de sites si élevé.
Contrairement à Count Distribution l'algorithme Partition parallèle est moins couteux pour un taux de parallélisme plus grand. Plus précisément, quand le nombre de sites augment le temps d'exécution diminue.
Les résultats ci-dessus montrent que l'algorithme Partition parallèle est plus actif que l'algorithme Count distribution, particulièrement pour un grand nombre de site.

V.3.2 Évaluation de l'algorithme Partition parallèle en fonction du taux de parallélisme

Pour étudier les performances de de l'algorithme Partition parallèle, nous avons fixé le support minimal à 30% et en changeant le nombre de sites, nous avons présenté le temps d'exécution pour des bases de données de tailles différentes.

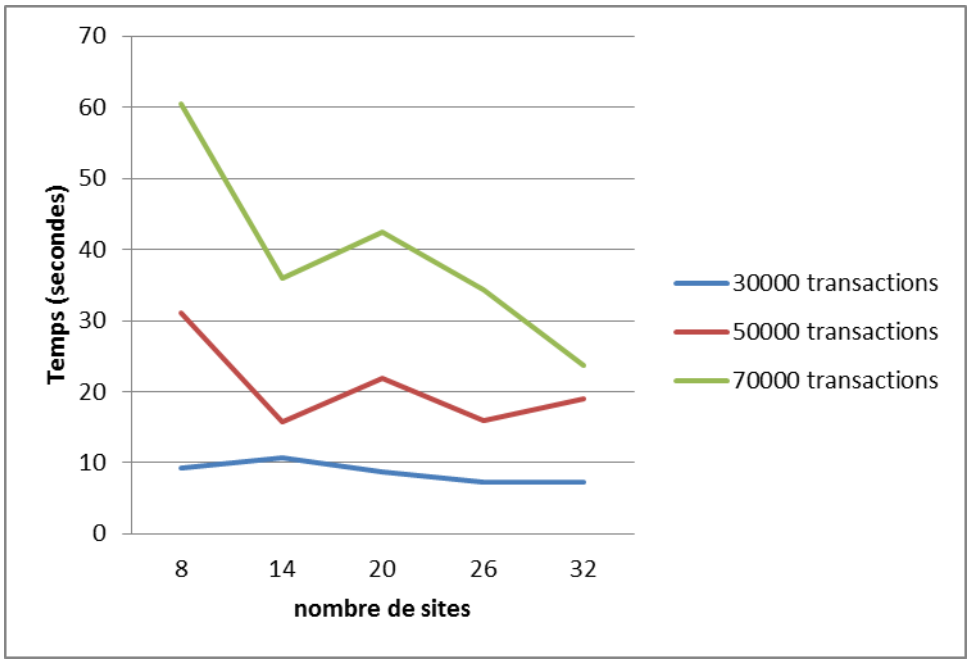

Figure V-7 : L'algorithme Partition parallèle sur 30000, 50000, 70000 transactions

La figure V-7 présente l'efficacité de l'algorithme Partition parallèle en fonction du nombre de sites. L'observation du temps d'exécution pour chaque base confirme cependant une diminution en temps pour une augmentation du nombre de sites. Cela est notamment important pour une base de données plus grande. Cette réduction en temps d'exécution prouve que l'algorithme Partition parallèle n'est pas couteux en termes de communication et de synchronisation entre les sites. Vu que l'algorithme ne demande que deux phases de communication et qu'un seul parcourt de la base de données.
Comme résultats, nous pouvons déduire que l'algorithme Partition parallèle est plus performant pour un taux de parallélisme très haut.

V.4. Experiences2

Dans le but d'analyser l'influence d'un partitionnement intelligent de la base de données sur l'algorithme Partition parallèle. Nous avons mis en œuvre l'algorithme k-prototypes parallèle proposé, le prétraitement de la base de données procédé par l'algorithme k-prototypes parallèle permet de diviser la base de données en partitions ou clusters contenant des transactions similaires. Le but de ce prétraitement est de minimiser l'espace de recherche pour l'algorithme Partition parallèle par le fait que le nombre d'items dans une partition qui contient des transactions similaire est moins que dans une partition de la base.
Dans un premier temps nous réalisons un prétraitement de la base de données par l'application de l'algorithme k-prototypes parallèle sur une base de données mixte. Dans un deuxième temps, l'algorithme Partition parallèle est exécuté sur la base prétraitée par l'algorithme k-prototypes parallèle, ainsi sur une base divisée aléatoirement pour évaluer l'impact du prétraitement de la base sur les performances de L'algorithme Partition parallèle.

V.4.1. La base de données

La base *Census-Income (KDD)* [111] est un ensemble de données qui contient des données de recensement pondérés tirés des enquêtes démographiques en 1994 et 1995 menées par le Bureau du recensement des États-Unis. Cette base de données contient 41 variables démographiques et d'emploi, les variables sont de type numérique et catégorique (13 attributs numériques et 28 attributs catégoriques). C'est une base de données mixte comportant 299285 transactions.

V.4.2. Prétraitement de la base de données

Dans cette section, nous présentons les résultats des expérimentations sur l'algorithme de clustering k-prototype parallèle. Nous avons varié le paramètre K de l'algorithme qui représente le nombre de clusters ou partitions et en même temps le nombre de sites.

Bien qu'on utilise le simulateur, les expériences sont réellement traitées sur un seul PC. Alors, afin de ne pas dépasser la capacité de traitement pour l'algorithme Partition parallèle sur une base de 41 attributs, nous somme limité à 30000 transactions. La figure suivante présente les résultats produits par l'algorithme k-prototype parallèle.

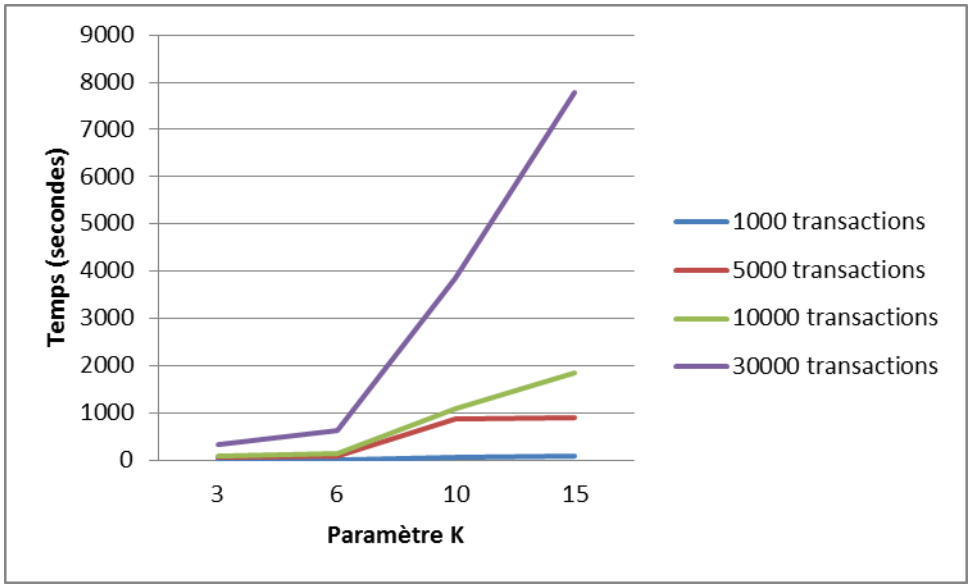

Figure V-8 : L'algorithme k-prototypes parallèle sur 1000, 5000, 10000 et 30000 transactions

Nous constatons que le temps d'exécutions achevé augmente avec le nombre K (nombre de clusters et de sites) et notamment pour un nombre important de transactions, cet accroissement est expliqué par le fait que chaque itération est liée à l'itération précédente sachant que chaque itération demande une synchronisation entre les sites. D'autre part, plus le nombre de transactions augmente, le nombre d'itérations est alors plus grand où l'influence du paramètre K est plus remarquable.

Par ailleurs, afin d'équilibré entre les attributs numériques et catégoriques, nous avons modifié le poids des attributs catégoriques à plusieurs reprise pour avoir un poids équitable

donnant meilleurs résultats. En définitive, le poids été fixé à 15. Bien que, le poids des attributs catégoriques est définit généralement par l'expert.
Suivant les partitions ou les clusters obtenus, nous avons remarqué que l'algorithme k-prototypes parallèle produit des clusters homogènes contenant des individus ou transactions vraiment similaire malgré que le résultat dépende fortement du choix initial des prototypes qui est aléatoire. Nous avons constaté ainsi que les clusters ont des transactions plus similaires pour un nombre plus grand de clusters (paramètre K).
Sachant qu'il existe des mesures permettant d'évaluer la qualité de clustering, notre but été simplement de diviser la base de données en partitions contenant des items similaires. Néanmoins, on peut noter que k-prototypes parallèle peut être utilisé efficacement pour un le problème de clustering dans une grille de données.

V.4.3. Evaluation de l'impact du prétraitement (partitionnement intelligent) sur l'algorithme Partition parallèle

Après le prétraitement de la base, nous avons testé l'algorithme Partition parallèle sur la base prétraitée. Autrement dit, partitionnée intelligemment sur les sites par l'algorithme de clustering k-prototypes parallèle. Alors, nous avons comparé les résultats avec ceux générés par l'algorithme Partition parallèle sur une base non traitée.

Afin de bien étudier l'influence du prétraitement, les tests sont effectués en variant les paramètres suivants : nombre d'attributs, nombre de sites et le nombre de transactions.

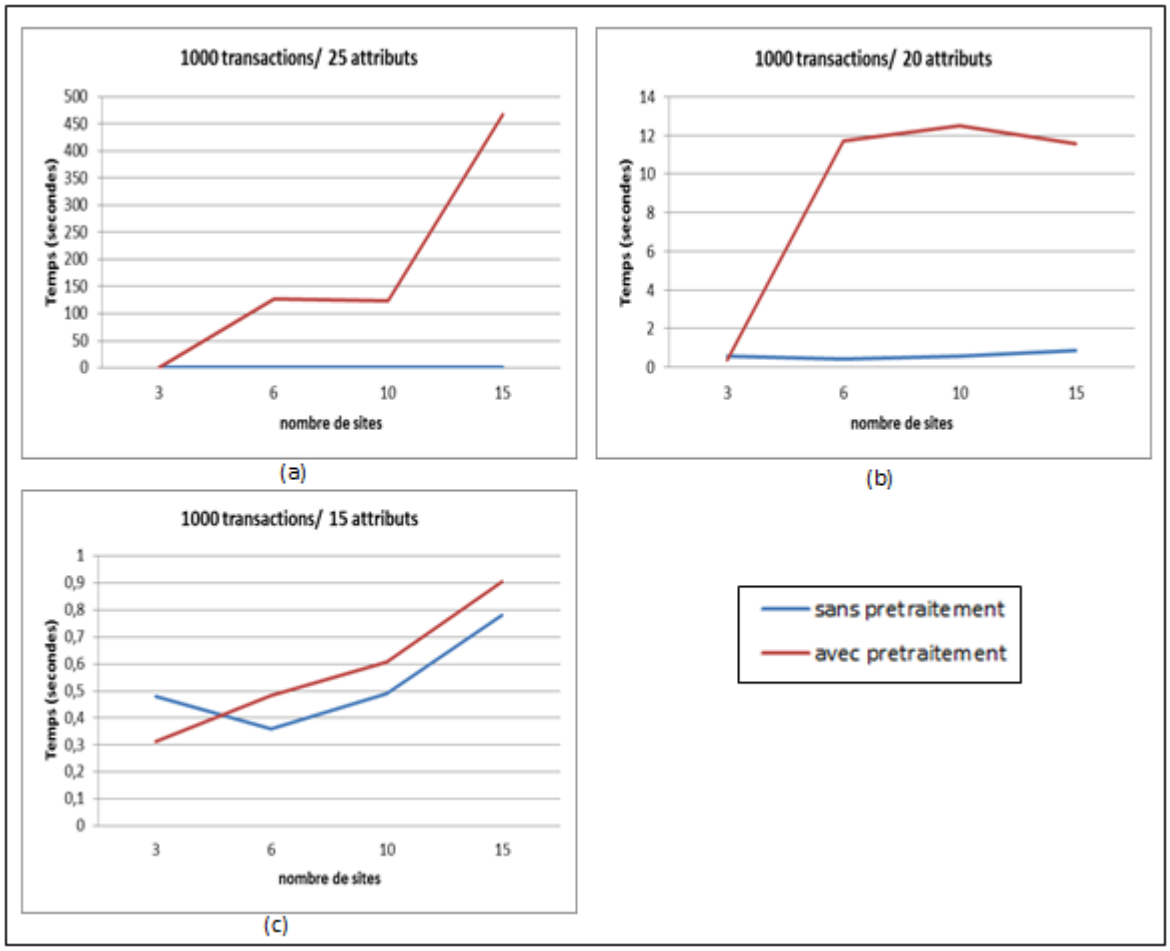

Figure V-9 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 1000 transactions

Les courbes de la figure V-9 présentent des résultats imprévus, nous observons que le temps d'exécution achevé par l'algorithme Partition parallèle sur une base prétraitée est beaucoup plus grand que le temps consommé sur une base non traitée.

Cette différence en temps d'exécution est remarquable surtout pour un nombre d'attributs encore grand comme le montre la figure V-9.(a). D'autre part, nous constatons que le temps augmente considérablement pour un nombre élevé de sites.

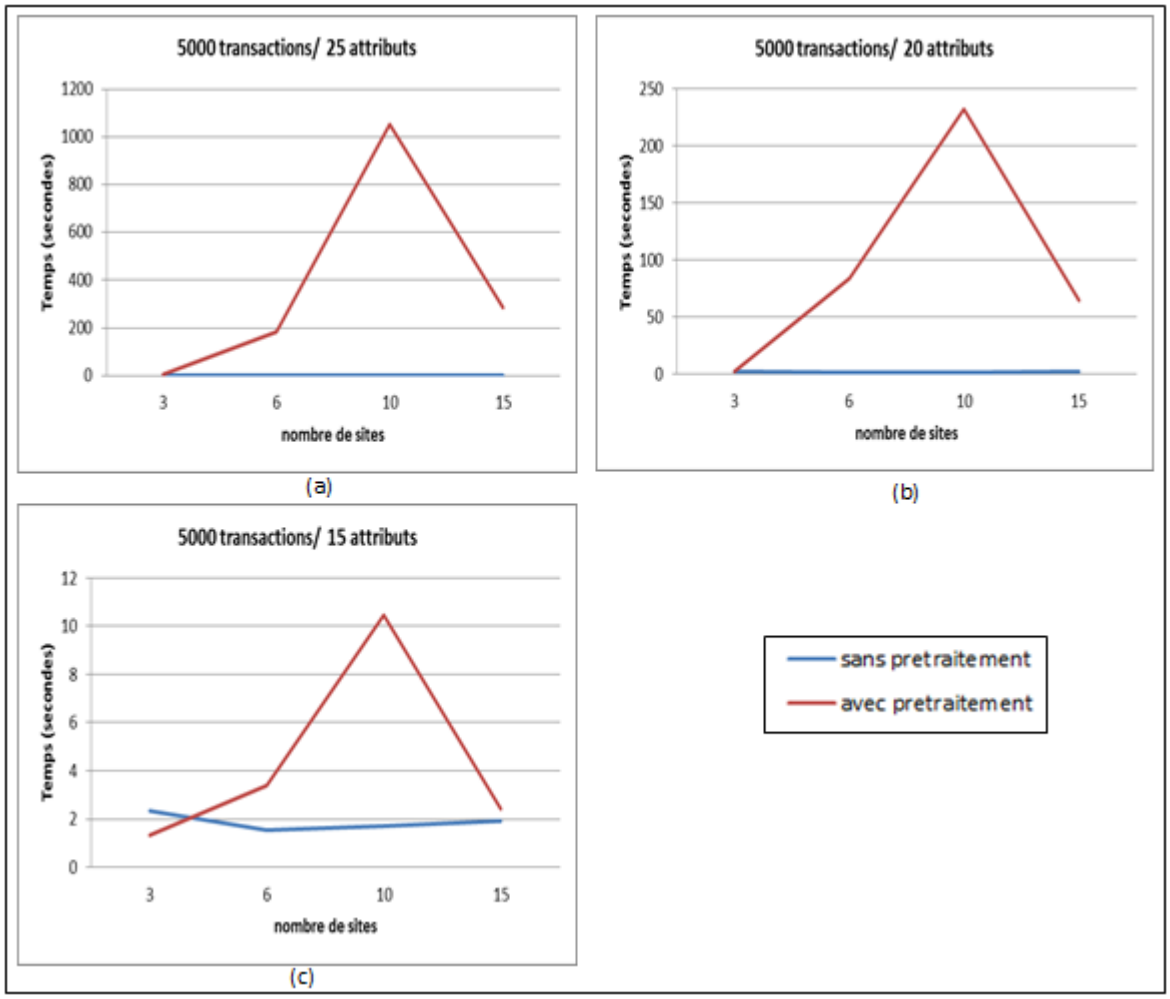

Figure V-10 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 5000 transactions

On constate les mêmes résultats présentés précédemment, sauf une exception avec 10 sites. Avec lesquelles le temps est beaucoup plus élevé même par rapport au celui de 15 sites. Donc, on peut dire que le résultat dépend ainsi du résultat de partitionnement intelligent.

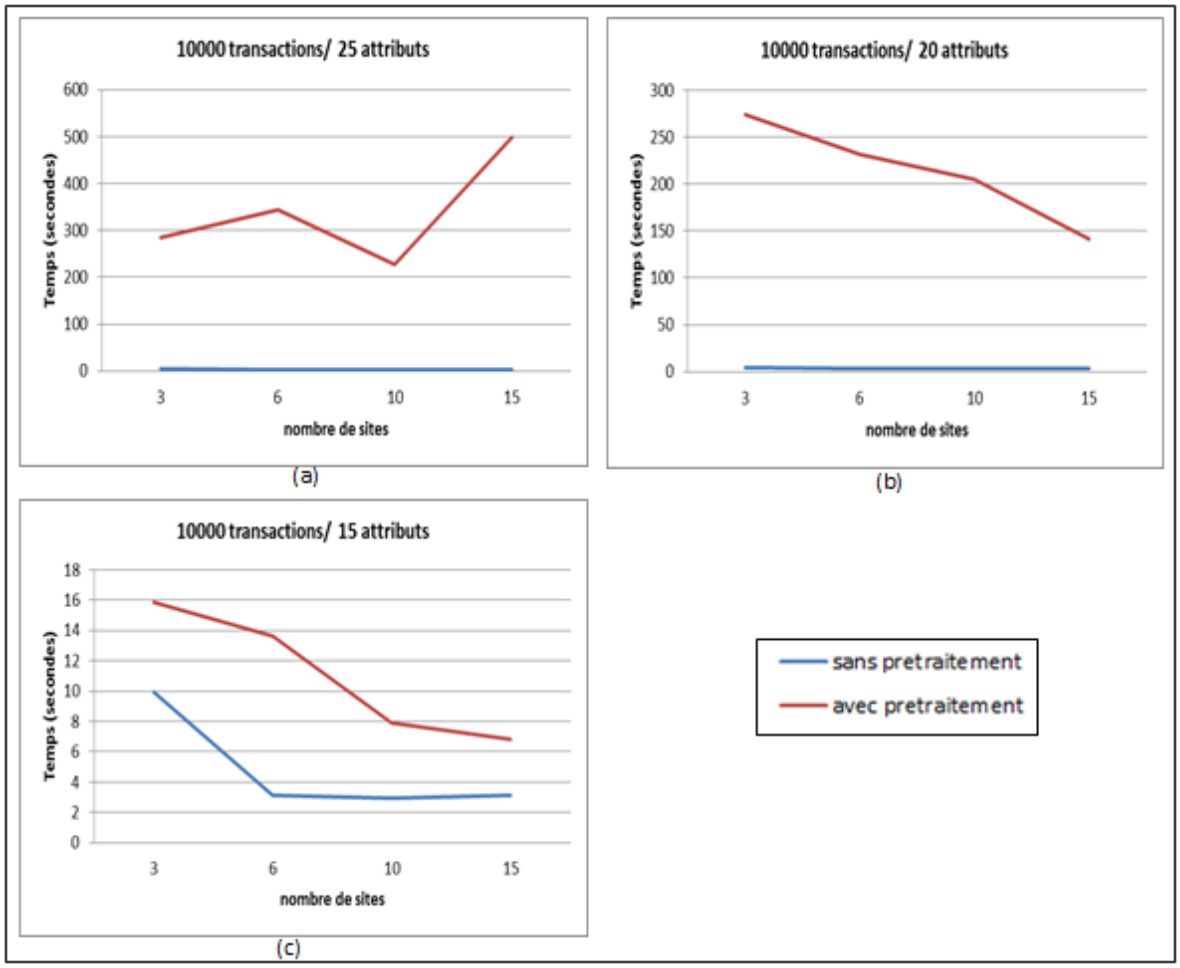

Figure V-11 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 10000 transactions

Contrairement aux résultats de 1000 et 5000 transactions, nous remarquons que l'algorithme Partition parallèle sur une base prétraitée présente un temps d'exécution qui se dégrade avec l'augmentation du nombre de sites pour 20 et 15 attributs.

Ces résultats sont montré dans les figures V-11.(b) et V-11.(c). Cependant, les résultats sont inversés pour 25 attributs dont le temps accroitre avec le nombre de sites.

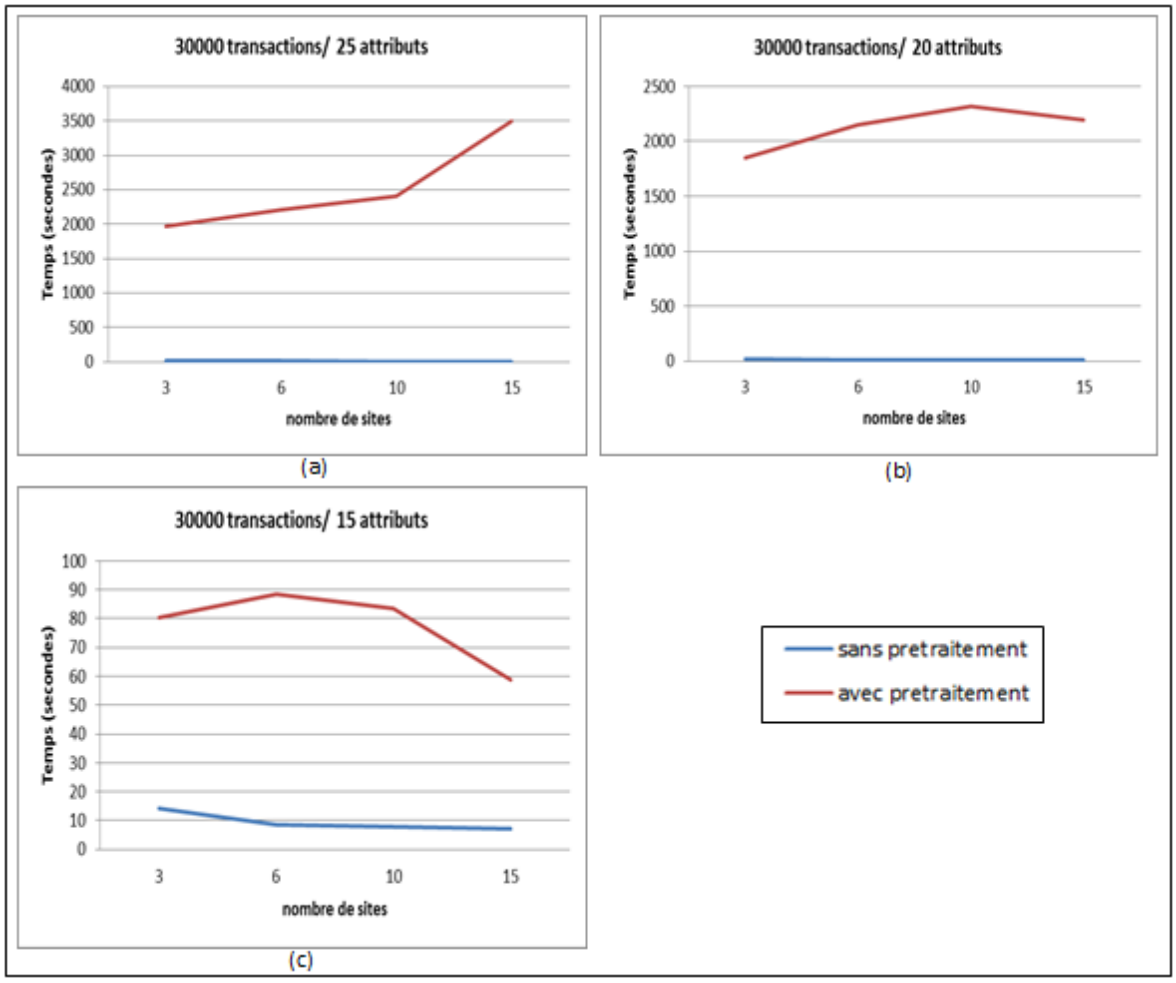

Figure V-12 : L'influence du prétraitement sur l'algorithme Partition parallèle pour 30000 transactions

Pour 30000 transactions, nous constatons que l'algorithme Partition parallèle est toujours plus rapide sur une base non traitée que dans une base traitée. Où l'algorithme est moins performant pour un nombre élevé de sites sauf pour le cas de 15 attributs présenté dans la figure V-12.(c), nous remarquons que le temps pour 15 sites et moins que celui de 10 sites.

Comme conclusion des figures présentées ci-dessus, nous notons que contradictoirement à l'algorithme Partition parallèle sur des données non traitées, l'algorithme sur des données traitées est moins efficace pour un grand nombre de sites. Néanmoins, cette différence n'est pas importante pour un nombre modéré d'attributs.

V.4.4. Évaluation de l'algorithme Partition parallèle après le prétraitement de la base en fonction du nombre d'attributs et de sites

Dans la suite, nous analysons l'algorithme Partition parallèle après le prétraitement de la base en fonction du nombre d'attributs et du nombre de sites pour des tailles différentes de la base de données.

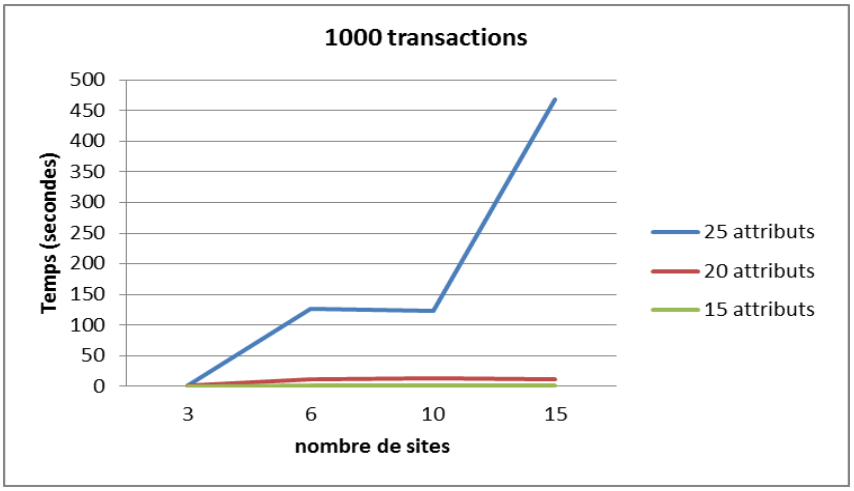

Figure V-13 : L'algorithme Partition parallèle après prétraitement de données sur 1000 transactions

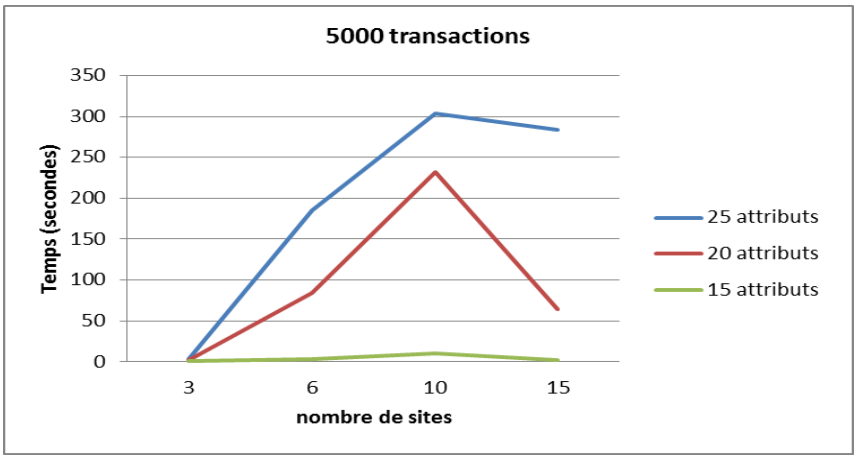

Figure V-14 : L'algorithme Partition parallèle après prétraitement de données sur 5000 transactions

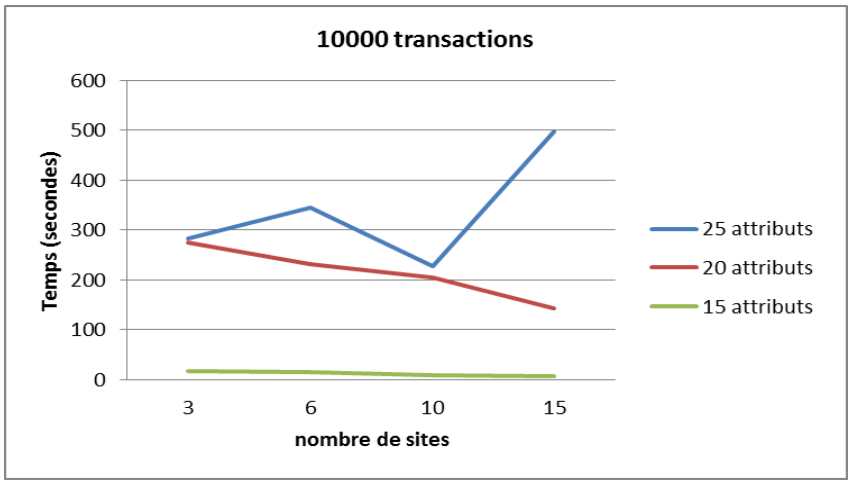

Figure V-15 : L'algorithme Partition parallèle après prétraitement de données sur 10000 transactions

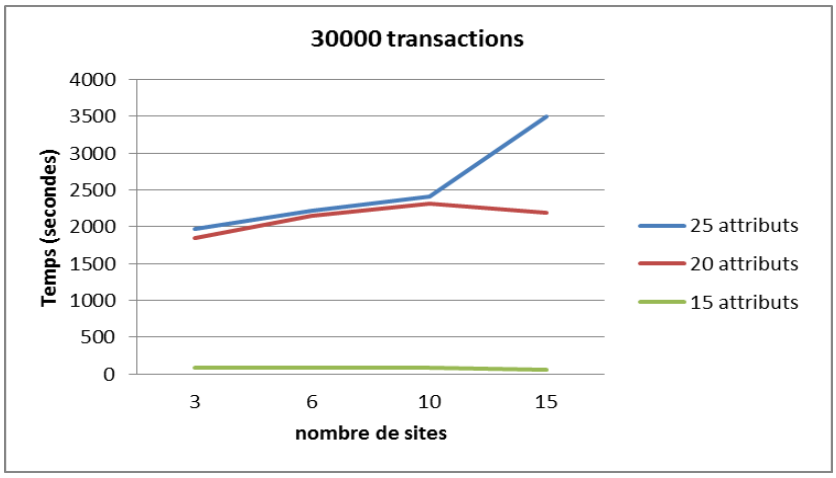

Figure V-16 : L'algorithme Partition parallèle après prétraitement de données sur 10000 transactions

Nous observons suivant les figures présentées que le temps augmente avec le nombre de sites, ceci est plus remarquable pour un nombre élevé d'attributs. Nous avons alors examiné la quantité de travail pour chaque site, de ce fait nous avons remarqué que les sites les plus retardateurs génèrent un nombre important d'itemsets fréquents. Aussi, nous avons remarqué que les transactions dans les sites retardateurs sont énormément similaires.

L'accroissement en temps d'exécution peut être expliqué par le fait que chaque site génère les itemsets fréquents locaux de toutes les tailles au court d'une seule phase. Dans la quel, les itemsets de taille i sont générés à partir des itemsets fréquents de taille i-1. Puisque les transactions sont similaires c.-à-d. contiennent des items similaires, on peut dire que tous les items générés sont fréquents localement. De cet effet, le nombre totale des itemsets fréquents localement (la majorité des items de la base) augmente exponentiellement (2^m) où m est le

nombre d'attribut. Et par conséquence, le nombre d'itemsets fréquents de tous les sites sera considérablement grand.

Afin d'étudier l'influence du nombre de sites sur le temps d'exécution, nous avons considéré 25 attributs pour un nombre variant de sites.

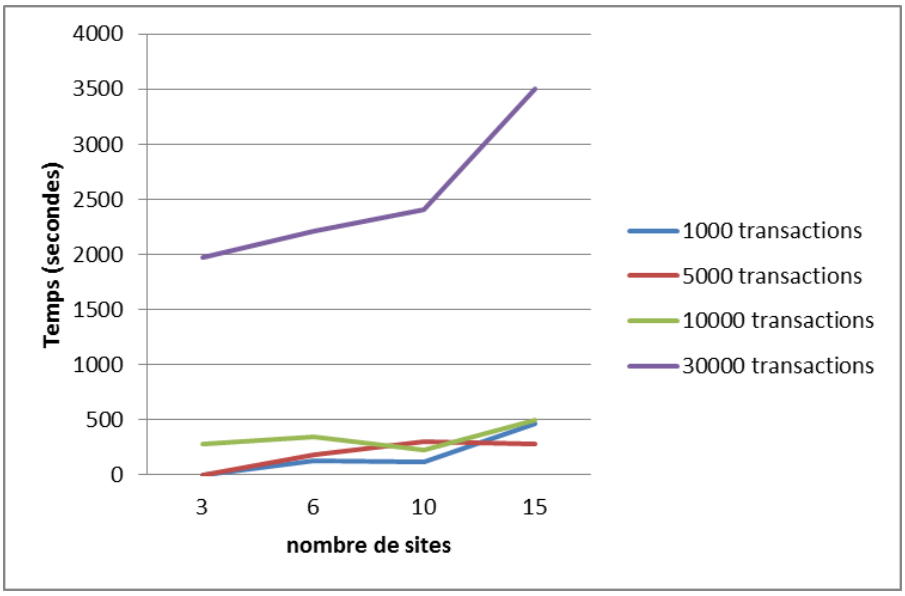

Figure V-17 : Performances de l'algorithme Partition parallèle après prétraitement en fonction du nombre de sites

Généralement, le temps d'exécution augmente avec le nombre de sites. Cette augmentation peut être exprimée par le taux de similarité qui est élevé pour un nombre de sites plus grand. Autrement dit, l'algorithme de clustering K-prototypes parallèle (l'algorithme de prétraitement) produit des clusters ou partitions plus homogènes (transaction plus similaire) en augmentant le nombre de clusters ou sites.

Alors, comme les transactions des sites sont plus similaires, le nombre d'itemsets fréquents localement accroitre. Ce nombre est ainsi multiplié par le nombre de sites qui est également grand. Néanmoins, des cas exceptionnels sont possibles, tel que le cas de 10 sites par rapport à 15 sites pour 5000 transactions et le cas 6 sites par rapport à 10 sites pour 1000 et 10000 transactions. Vu que le taux de similarité dans les sites dépend du résultat de l'algorithme de clustering.

V.5. Conclusion

Dans la première partie des expérimentations réalisées, nous avons pu observer l'efficacité de l'algorithme proposé Partition parallèle. L'algorithme étant tout à fait performant pour un taux de parallélisme élevé. Comparant l'algorithme Partition parallèle avec l'algorithme Count Distribution, nous pouvons déduire que les résultats de notre algorithme sont satisfaisants.

L'algorithme a prouvé son perfectionnement notamment pour une base de données volumineuse sur une grille de données étendue.
Dans la seconde partie des expérimentations, nous avons réalisé un prétraitement de la base pour l'algorithme Partition parallèle. Après la mise en œuvre de l'algorithme de clustering proposé k-prototypes parallèle pour le prétraitement, nous avons remarqué le bon comportement de cet algorithme, qui est capable d'identifier des clusters ou partitions homogènes dans un environnement distribué. Nous avons également appliqué l'algorithme Partition parallèle sur la base prétraitée. Alors, nous avons constaté que le prétraitement n'est pas rentable pour notre algorithme. Cependant, le prétraitement proposé de la base de données peut être testé pour d'autres algorithmes d'extraction de règles d'association tel que l'algorithme FP-Growth parallèle afin de minimiser la taille des FP-tree.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

Dans le cadre de cette thèse, nous avons essayé d'apporter des solutions pour le problème d'extraction de règles d'association sur une grille informatique. Les critères majeurs pour concevoir une solution optimale dans un environnement parallèle et distribué est de minimiser le cout de synchronisation et de communication. Reposant sur ces hypothèses et face à la quantité importante des méthodes existantes dans le domaine d'extraction des règles d'association parallèle, nous avons fait un tour d'horizon des méthodes les plus populaires en détaillant leurs forces et faiblesses.

Cette étude nous a poussé à choisir l'algorithme Partition pour concevoir une approche pour l'extraction de règles d'association sur une grille. L'avantage de l'algorithme Partition est qu'il ne demande que deux balayages de la base ce qui implique que la version parallèle demande moins de synchronisation. Dans ce cadre, nous avons proposé une amélioration de l'algorithme Partition en réduisant le nombre de balayage de données en un seul, par la suite nous avons conçus l'algorithme Partition parallèle en respectant les deux hypothèses, le cout de communication et de synchronisation.

Une autre solution pour perfectionner l'extraction de règle d'association dans un environnement distribué est le partitionnement intelligent de la base à travers les machines. Alors pour cette problématique, nous avons ainsi étudié les techniques de clustering et plus particulièrement le clustering de partitionnement séquentiel et parallèle. Cette étude transversale nous a amené également à proposer l'algorithme k-prototypes parallèle, cette approche est basée sur le modèle passage de message pour communiquer entre les machines. L'avantage de l'algorithme k-prototypes est qu'il permet de traiter des données mixtes ce qui est le cas des données réelles.

Pendant la conception des deux approches, nous avons adopté la topologie étoile (centralisé) pour la grille de données.

Les expérimentations réalisées ont été divisées en deux parties, la première teste les performances de l'algorithme Partition parallèle sur un jeu de données numériques. Tandis que la deuxième partie, étudie l'influence du partitionnement intelligent (réalisé par l'algorithme k-prototypes parallèle) sur l'algorithme Partition parallèle. Dans cette partie les expérimentations sont effectuées sur des données mixtes.

La mise en œuvre de l'algorithme Partition parallèle proposé a prouvé son efficacité particulièrement pour un taux de parallélisme élevé. Les résultats de comparaison entre notre approche et l'algorithme parallèle Count Distribution montre que l'algorithme Partition parallèle est plus performant, son perfectionnement est plus remarquable pour une base de données volumineuse sur une grille de données étendue.

La deuxième approche proposée pour le prétraitement de données (partitionnement intelligent) k-prototypes parallèle a donner de bon résultats. L'algorithme peut être appliqué efficacement sur une grille de données en générant des partitions homogènes dans chaque machine. D'autre part, les expériences ont montré que le partitionnement intelligent n'est pas avantageux pour l'algorithme Partition parallèle. Selon les résultats obtenus, une partition homogène génère un nombre plus élevé d'itemsets fréquents localement avec l'algorithme Partition, ce qui pénalise son efficacité. Par contre, le partitionnement intelligent de la base peut être rentable pour d'autres algorithmes. D'une façon générale, l'algorithme k-prototypes parallèle est une solution intéressante pour le problème de clustering dans un environnement distribué.

Plusieurs pistes restent à explorer afin d'améliorer les résultats obtenus. D'une part, l'étude d'autres algorithmes de règles d'association parallèles sur une base prétraitée par l'algorithme k-prototypes parallèle pourrait conduire à améliorer leurs efficacités. Tel que l'algorithme FP-Growth parallèle, dont la taille des FP-tree dans chaque machine peut être minimisée par le fait que le nombre d'items est réduit. D'autre part, il est intéressant d'évaluer l'algorithme k-prototypes parallèle par les critères d'évaluation de clustering connus dans la littérature afin d'améliorer son productivité.

BIBLIOGRAPHIE

[1] O. R. Zaïane, « Chapter I: Introduction to Data Mining », *CMPUT690 Principles of Knowledge Discovery in Databases*, vol. 17, 1999.

[2] M. Tommasi et R. Gilleron, « Découverte de connaissances à partir de données », *cours des outils, des techniques liées à l'informatique décisionnelle*, 2000.

[3] T. Zhang, R. Ramakrishnan, et M. Livny, « BIRCH: an efficient data clustering method for very large databases », SIGMOD international conference on Management of data p. 103-114, Montréal, Canada, June 04-06, 1996.

[4] S. Guha, R. Rastogi, et K. Shim, « CURE: an efficient clustering algorithm for large databases », ACM SIGMOD international conference Management of Data, p. 73-84, 1998.

[5] G. Karypis, E.-H. Han, et V. Kumar, « Chameleon: Hierarchical clustering using dynamic modeling », *IEEE Computer*, vol. 32, n° 8, p. 68–75, 1999.

[6] E. W. Forgy, « Cluster analysis of multivariate data: efficiency versus interpretability of classifications », *Biometrics*, vol. 21, p. 768–769, 1965.

[7] R. T. Ng et J. Han, « CLARANS: A method for clustering objects for spatial data mining », *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14, n° 5, p. 1003–1016, 2002.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, et C. J. Stone, « Classification and Regression Trees », *Wadsworth, Belmont, California*, 1984.

[9] J. R. Quinlan, «*C4. 5: programs for machine learning»*, Morgan Kaufmann, vol. 1, San Francisco, CA, 1993.

[10] M. Mehta, R. Agrawal, et J. Rissanen, « SLIQ: A fast scalable classifier for data mining », *Advances in Database Technology—EDBT'96*, Springer, p. 18–32, 1996.

[11] J. Shafer, R. Agrawal, et M. Mehta, « SPRINT: A scalable parallel classi er for data mining », The 22th International Conference on Very Large Data Bases, p. 544-555, September 03-06, 1996.

[12] V. Ganti, J. Gehrke, R. Ramakrishnan, et W.-Y. Loh, « A framework for measuring differences in data characteristics », *Journal of Computer and System Sciences*, vol. 64, n° 3, p. 542–578, 2002.

[13] R. Agrawal, R. Srikant, et others, « Fast algorithms for mining association rules », 20th International Conference on Very Large Data Bases, p.487-499, September 12-15, 1994.

[14] A. Savasere, E. R. Omiecinski, et S. B. Navathe, « An efficient algorithm for mining association rules in large databases », The 21th International Conference on Very Large Data Bases, p. 432-444, September 11-15, 1995.

[15] J. M. Delorme, « L'apport de la fouille de données dans l'analyse de texte », *Conservatoire National des Arts et Métiers, Centre régional de Montpellier*, Avril 2002.

[16] I. Foster et C. Kesselman, «*The Grid 2: Blueprint for a new computing infrastructure »*. Morgan Kaufmann Book in Elsevier, 2003.

[17] M. Soberman, « Les grilles informatiques-état de l'art et déploiement », *CNRS/STIC*, 2005.

[18] S. Monnet, « Gestion des données dans les grilles de calcul: support pour la tolérance aux fautes et la cohérence des données. », Thèse de doctorat, Université Rennes 1, 2006.

[19] T. Soueid, « Grilles de calcul: Etat de l'art », Télécom ParisTech. Diplôme d'études approfondies Informatique. ENST, Juin 2003.

[20] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, et others, « *Introduction to grid computing with globus*». IBM Corporation, International Technical Support Organization, 2003.

[21] R. Buyya et S. Venugopal, « A gentle introduction to grid computing and technologies », *CSI Communications, vol. 29, n°1, p. 9–19*, 2005.

[22] P. A. Aranda Gutiérrez, A. Bulanza, M. Dabrowski, B. Kaskina, J. Quittek, C. Schmoll, F. Strohmeier, A. Vidacs, et K. S. Zsolt, « An advanced measurement meta-repository », P 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurement; IPSMoMe 2005, Warsaw, Poland, March 2005.

[23] M. J. Zaki, « Parallel and distributed data mining: An introduction », in *Large-Scale Parallel Data Mining*, Springer, p. 1–23, 2000.

[24] D. DeWitt et J. Gray, « Parallel database systems: the future of high performance database systems », *Communications of the ACM*, vol. 35, no 6, p. 85–98, 1992.

[25] P. Valduriez, « Parallel database systems: Open problems and new issues », *Distributed and parallel Databases*, vol. 1, no 2, p. 137–165, 1993.

[26] R. Kufrin, « Decision trees on parallel processors », *Machine Intelligence and Pattern Recognition* Elsevier-Science, vol. 20, p. 279–306, 1997.

[27] M. V. Joshi, G. Karypis, et V. Kumar, « ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets », The First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, Orlando, FL, USA, p. 573–579. 1998.

[28] M. J. Zaki, C.-T. Ho, et R. Agrawal, « Parallel classification for data mining on shared-memory multiprocessors », 15th IEEE international conference on Data Engineering., p. 198–205, 1999.

[29] A. Srivastava, E.-H. Han, V. Kumar, et V. Singh, «*Parallel formulations of decision-tree classification algorithms*». Data Mining and Knowledge Discovery, vol. 3, n°3, p. 237-261, September 1999.

[30] A. Congiusta, D. Talia, et P. Trunfio, « Parallel and Grid-Based Data Mining–Algorithms, Models and Systems for High-Performance KDD », *Data Mining and Knowledge Discovery Handbook*, Springer, p. 1009–1028, 2010.

[31] F. Neri et A. Giordana, « A parallel genetic algorithm for concept learning », The 6th International Conference on Genetic Algorithms., Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, p. 436-443,1995.

[32] V. Fiolet, « Algorithmes distribués d'extraction de connaissances », Thèse de doctorat en Informatique, université de Lille 1, 2006.

[33] J. Han et Y. Fu, « Discovery of multiple-level association rules from large databases », 21th Int'l Conf. Very Large Data Bases, p. 420-431, Sept 1995.

[34] J. S. Park, M.-S. Chen, et P. S. Yu, « *An effective hash-based algorithm for mining association rules*», ACM, vol. 24. 1995.

[35] R. Agrawal, T. Imieliński, et A. Swami, « Mining association rules between sets of items in large databases », *ACM SIGMOD Record*, vol. 22, p. 207–216, 1993.

[36] M. J. Zaki, S. Parthasarathy, M. Ogihara, W. Li, et others, « New Algorithms for Fast Discovery of Association Rules. », The Third international conference on Knowledge Discovery in Databases and Data Mining, p. 283–286, 1997.

[37] H. Toivonen et others, « Sampling large databases for association rules », 22th International Conference on Very Large Data Bases, p.134-145, September 03-06, 1996.

[38] S. Brin, R. Motwani, J. D. Ullman, et S. Tsur, « Dynamic itemset counting and implication rules for market basket data », ACM SIGMOD international conference on Management of data, p.255-264, May 11-15, 1997, Tucson, Arizona, USA, 1997.

[39] J. Han, J. Pei, Y. Yin, et R. Mao, « Mining frequent patterns without candidate generation: A frequent-pattern tree approach », *Data mining and knowledge discovery*, vol. 8, nº 1, p. 53–87, 2004.

[40] R. Agrawal et J. C. Shafer, « Parallel mining of association rules », *IEEE Transactions on knowledge and Data Engineering*, vol. 8, nº 6, p. 962–969, 1996.

[41] D. W. Cheung, J. Han, V. T. Ng, A. W. Fu, et Y. Fu, « A fast distributed algorithm for mining association rules », Fourth international conference Parallel and Distributed Information Systems, p. 31–42, December 1996.

[42] D. W. Cheung, V. T. Ng, A. W. Fu, et Y. Fu, « Efficient mining of association rules in distributed databases », *Knowledge and Data Engineering, IEEE Transactions on*, vol. 8, nº 6, p. 911–922, 1996.

[43] M. Z. Ashrafi, D. Taniar, et K. Smith, « ODAM: An optimized distributed association rule mining algorithm », *IEEE distributed systems online*, vol. 5, nº 3, p. 1–18, 2004.

[44] E.-H. Han, G. Karypis, et V. Kumar, « Scalable parallel data mining for association rules », *Knowledge and Data Engineering, IEEE Transactions on*, vol. 12, nº 3, p. 337–352, 2000.

[45] M. V. Joshi, E.-H. S. Han, G. Karypis, et V. Kumar, « *Efficient parallel algorithms for mining associations»*. Springer, 2000.

[46] T. Shintani et M. Kitsuregawa, « Hash based parallel algorithms for mining association rules », 4th international conference Parallel and Distributed Info. Systems, p. 19–30, December 1996.

[47] M. J. Zaki, S. Parthasarathy, M. Ogihara, et W. Li, « Parallel algorithms for discovery of association rules », *Data Mining and Knowledge Discovery*, vol. 1, nº 4, p. 343–373, 1997.

[48] J. S. Park, M.-S. Chen, et P. S . Yu, « Efficient parallel data mining for association rules », ACM SIGMOD, p. 175-186, May 1995.

[49] D. W. Cheung et Y. Xiao, « Effect of data skewness in parallel mining of association rules », *Research and Development in Knowledge Discovery and Data Mining*, Springer, p. 48–60, 1998.

[50] A. Mueller, « Fast sequential and parallel algorithms for association rule mining: A comparison », Technical report, Faculty of the Graduate School of The University of Maryland, 1995.

[51] H. Li, Y. Wang, D. Zhang, M. Zhang, et E. Y. Chang, « Pfp: parallel fp-growth for query recommendation », The 2008 ACM conference on Recommender systems, October 23-25, 2008, Lausanne, Switzerland 2008.

[52] M. Halkidi, Y. Batistakis, et M. Vazirgiannis, « On clustering validation techniques », *Journal of Intelligent Information Systems*, vol. 17, nº 2‑3, p. 107–145, 2001.

[53] P. Jaccard, «*Nouvelles recherches sur la distribution florale»*. Bulletin de la Société Vandoise des Sciences Naturelles, vol. 44, p. 223–270, 1908.

[54] R. R. Sokal, P. H. Sneath, et others, « Principles of numerical taxonomy. », Book Edition *Freeman San Francisco*, 1963.

[55] R. R. Sokal, « A statistical method for evaluating systematic relationships », *Univ Kans Sci Bull*, vol. 38, p. 1409–1438, 1958.

[56] P.-N. Tan, M. Steinbach, et V. Kumar, « Cluster analysis: basic concepts and algorithms », *Introduction to data mining*, Boston: Pearson Addison Wesley p. 487–568, 2006.

[57] M. Halkidi, Y. Batistakis, et M. Vazirgiannis, « Clustering validity checking methods: part II », *ACM Sigmod Record*, vol. 31, nº 3, p. 19–27, 2002.

[58] A. K. Jain, « *Algorithms for clustering data »*. Englewood Cliffs, N.J: Prentice Hall, 1988.
[59] P. N. Tan, M. Steinbach, et V. Kumar, « *Introduction to data mining ». Inc*. Boston, MA, USA: Addison-Wesley Longman Publishing Co, 2005.
[60] L. Kaufman et P. J. Rousseeuw, « Finding groups in data: An introduction to cluster analysis» Wiley Interscience , *New York, NY*, 1990.
[61] J. C. Dunn†, « Well-separated clusters and optimal fuzzy partitions », *Journal of cybernetics*, vol. 4, nº 1, p. 95–104, 1974.
[62] D. L. Davies et D. W. Bouldin, « A cluster separation measure », *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, nº 2, p. 224–227, 1979.
[63] C. Wemmert, « Classification hybride distribuée par collaboration de méthodes non supervisées », PhD thesis, Université Louis Pasteur, Strasbourg 2000.
[64] Y. Zhao et G. Karypis, « Criterion functions for document clustering: Experiments and analysis », Technical Report, Department of Computer Science, University of Minnesota, Minneapolis, MN, 2001.
[65] M. Meilă, « Comparing clusterings by the variation of information », *Learning theory and kernel machines*, Springer, p. 173–187, 2003.
[66] A. Patrikainen et M. Meila, « Comparing Subspace Clusterings », IEEE Transactions on Knowledge and Data Engineering, vol.18 n°7, p.902-916, July 2006.
[67] A. K. Jain, M. N. Murty, et P. J. Flynn, « Data clustering: a review », *ACM computing surveys (CSUR)*, vol. 31, nº 3, p. 264–323, 1999.
[68] G. H. Ball et D. J. Hall, « ISODATA, a novel method of data analysis and pattern classification », Technical Report Stanford University, Stanford, CA, 1965.
[69] M. A. Figueiredo et A. K. Jain, « Unsupervised learning of finite mixture models », *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, nº 3, p. 381–396, 2002.
[70] M. H. Hansen et B. Yu, « Model selection and the principle of minimum description length », *Journal of the American Statistical Association*, vol. 96, nº 454, p. 746–774, 2001.
[71] B. G. Tabachnick, L. S. Fidell, et others, « Using multivariate statistics », 5th Edition Allyn & Bacon, Inc. Needham Heights, MA, USA 2006.
[72] T. Lange, V. Roth, M. L. Braun, et J. M. Buhmann, « Stability-based validation of clustering solutions », *Neural computation*, vol. 16, nº 6, p. 1299–1323, 2004.
[73] A. K. Jain, A. Topchy, M. H. Law, et J. M. Buhmann, « Landscape of clustering algorithms », IEEE Conference Pattern Recognition, p. 260-263, Aug 2004.
[74] A. K. Jain, « Data clustering: 50 years beyond K-means », *Pattern Recognition Letters*, vol. 31, nº 8, p. 651–666, 2010.
[75] S. Guha, R. Rastogi, et K. Shim, « ROCK: A robust clustering algorithm for categorical attributes », Information Systems, vol. 25, n° 5, p.345-366, July 2000.
[76] M. Chavent, C. Guinot, Y. Lechevallier, et M. Tenenhaus, « Méthodes divisives de classification et segmentation non supervisée: Recherche d'une typologie de la peau humaine saine », *Revue de statistique appliquée*, vol. 47, nº 4, p. 87–99, 1999.
[77] D. H. Fisher, « Knowledge acquisition via incremental conceptual clustering », *Machine learning*, vol. 2, nº 2, p. 139–172, 1987.
[78] K. Mali et S. Mitra, « Clustering and its validation in a symbolic framework », *Pattern Recognition Letters*, vol. 24, nº 14, p. 2367–2376, 2003.

[79] R. Agrawal, J. Gehrke, D. Gunopulos, et P. Raghavan, «*Automatic subspace clustering of high dimensional data for data mining applications*», ACM. vol. 27, 1998.

[80] S. Goil, H. Nagesh, et A. Choudhary, « MAFIA: Efficient and scalable subspace clustering for very large data sets », Technical Report CPDC-TR-9906-010, Northwestern University, Evanston IL June, 1999.

[81] M. Ester, H.-P. Kriegel, J. Sander, et X. Xu, « A density-based algorithm for discovering clusters in large spatial databases with noise. », 2nd international conference Knowledge Discovery and Data Mining (KDD'96), p. 226-231, 1996.

[82] A. Hinneburg et D. A. Keim, « An efficient approach to clustering in large multimedia databases with noise », 4th international conference Knowledge Discovery and Data Mining (KDD'98), p. 58-65, 1998.

[83] R. A. Jarvis et E. A. Patrick, « Clustering using a similarity measure based on shared near neighbors », *Computers, IEEE Transactions on*, vol. 100, nº 11, p. 1025–1034, 1973.

[84] K. Chidananda Gowda et G. Krishna, « Agglomerative clustering using the concept of mutual nearest neighbourhood », *Pattern Recognition*, vol. 10, nº 2, p. 105–112, 1978.

[85] D. W. Matula, « k-Components, clusters and slicings in graphs », *SIAM Journal on Applied Mathematics*, vol. 22, nº 3, p. 459–480, 1972.

[86] E. Hartuv et R. Shamir, « A clustering algorithm based on graph connectivity », *Information processing letters*, vol. 76, nº 4, p. 175–181, 2000.

[87] Z. Wu et R. Leahy, « An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation », *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, nº 11, p. 1101–1113, 1993.

[88] W. Wang, J. Yang, R. Muntz, et others, « STING: A statistical information grid approach to spatial data mining », The 23rd International Conference on Very Large Data Bases, p.186-195, August 25-29, 1997.

[89] G. Sheikholeslami, S. Chatterjee, et A. Zhang, « Wavecluster: A multi-resolution clustering approach for very large spatial databases », 24th VLDB Conf., pp. 428-439, 1998.

[90] A. El Golli, B. Conan-Guez, F. Rossi, et others, « Self-organizing maps and symbolic data », *Journal of Symbolic Data Analysis*, vol. 2, nº 1, 2004.

[91] A. P. Dempster, N. M. Laird, et D. B. Rubin, « Maximum likelihood from incomplete data via the EM algorithm », *Journal of the Royal Statistical Society. Series B (Methodological)*, p. 1–38, 1977.

[92] E. Diday, G. Celeux, G. Govaert, Y. Lechevallier, et H. Ralambondrainy, « Classification Automatique des Données », *Dunod, Paris*, vol. 49, p. 80, 1989.

[93] L. Kaufman et P. Rousseeuw, «*Clustering by means of medoids*». Statistical data analysis based on the L1 norm p. 405–416 Amsterdam: North-Holland, 1987.

[94] J. Mac-Queen, « Some methods for classification and analysis of multivariate observations », 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California, p. 281‑297, 1967.

[95] Z. Huang, « A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining. », SIG- OD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tech. Report 97-07, UBC, Dept. of CS, 1997.

[96] Z. Huang, « Extensions to the k-means algorithm for clustering large data sets with categorical values », *Data mining and knowledge discovery*, vol. 2, nº 3, p. 283–304, 1998.

[97] Z. Huang, « Clustering large data sets with mixed numeric and categorical values », Proceedings of the First Pacific Asia Knowledge Discovery and Data Mining Conference, Singapore: World Scientific, p. 21–34 1997.

[98] K. Alsabti, S. Ranka, et V. Singh, « An efficient k-means clustering algorithm », IPPS/SPDP Workshop on High Performance Data Mining. IEEE Computer Society Press, 1998.

[99] A. Likas, N. Vlassis, et J. J Verbeek, « The global k-means clustering algorithm », *Pattern recognition*, vol. 36, nº 2, p. 451–461, 2003.

[100] L. Kaufman et P. J. Rousseeuw, « *Finding groups in data: an introduction to cluster analysis»*, vol. 344. John Wiley & Sons, 2009.

[101] R. T. Ng et J. Han, « Efficient and E ective Clustering Methods for Spatial Data Mining », International Conference on Very Large Data Bases, p.144-155, September 12-15, 1994.

[102] I. S. Dhillon et D. S. Modha, « A data-clustering algorithm on distributed memory multiprocessors », in *Large-Scale Parallel Data Mining*, Springer, p. 245–260, 2000.

[103] W. Gropp, E. Lusk, et A. Skjellum, « *Using MPI: portable parallel programming with the message-passing interface»*, vol. 1. MIT press, 1999.

[104] J. Dean et S. Ghemawat, « MapReduce: simplified data processing on large clusters », *Communications of the ACM*, vol. 51, nº 1, p. 107–113, 2008.

[105] W. Zhao, H. Ma, et Q. He, « Parallel k-means clustering based on mapreduce », in *Cloud Computing*, Springer, p. 674–679, 2009.

[106] D. Borthakur, « The hadoop distributed file system: Architecture and design », *Hadoop Project Website*, vol. 11, p. 21, 2007.

[107] Y.-X. Wang, Z.-H. Wang, et X.-M. Li, « A parallel clustering algorithm for categorical data set », in *Artificial Intelligence and Soft Computing-ICAISC 2004*, Springer, p. 928–933, 2004.

[108] S. Kantabutra et A. L. Couch, « Parallel K-means clustering algorithm on NOWs », *NECTEC Technical journal*, vol. 1, nº 6, p. 243–247, 2000.

[109] Y.-P. Zhang, J.-Z. Sun, Y. Zhang, et X. Zhang, « Parallel implementation of CLARANS using PVM », International Conference on, 3:1646–1649 vol.3, August 2004.

[110] R. Buyya et M. Murshed, « Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing », *Concurrency and computation: practice and experience*, vol. 14, nº 13‑15, p. 1175–1220, 2002.

[111] K. Bache et M. Lichman, « UCI machine learning repository », *URL http://archive. ics. uci. edu/ml,* vol. 19, 2013.