N° d'ordre: 75/2019-C/info

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique

Thèse

Présentée pour l'obtention du diplôme de Doctorat

EN : INFORMATIQUE

Option : Intelligence Artificielle

Par : BOUSBACI Abdelhak

Sujet :

Algorithmes parallèles de clustering de données

Soutenue publiquement le 29/07/2019, devant le jury composé de:

Mme. DRIAS Habiba	Professeur à l'USTHB	Présidente
Mme. KAMEL Nadjet	Professeur à l'UFAS1	Directrice de thèse
M. BABA ALI Riadh	Professeur à l'USTHB	Examinateur
M. BOUKRA Abdelmadjid	Professeur à l'USTHB	Examinateur
M. SAÏS Lakhdar	Professeur à l'U. d'Artois	Examinateur

Résumé :

Le Clustering de données est un problème qui consiste à répartir un ensemble de données en groupes, appelés Clusters. La répartition des données se base sur la notion de similarité, de sorte que les données similaires appartiennent au même groupe.

Plusieurs algorithmes de Clustering ont été proposés et utilisés dans divers domaines, mais avec l'augmentation de la taille des données traitées, et leur distribution sur un ensemble de machines géographiquement éloignées, ces algorithmes deviennent non adaptés à ces situations. Pour cela, de nouveaux algorithmes parallèles ou distribués deviennent nécessaires.

Le travail de cette thèse se situe dans ce contexte d'étude. Il consiste à étudier les algorithmes de Clustering en général et les algorithmes de Clustering parallèles en particulier.

L'objectif de cette thèse de doctorat est de proposer un algorithme parallèle de Clustering afin de pouvoir analyser des données volumineuses efficacement.

Notre contribution porte sur la proposition d'un schéma parallèle pour le Clustering de données volumineuses dans un environnement distribué basé sur le Framework MapReduce. Les contributions réalisées consistent en 3 points principaux : **(1)** La proposition d'une parallélisation d'un algorithme de Clustering déjà existant en utilisant le Framework MapReduce ainsi qu'une parallélisation locale au niveau de chaque machine (mémoire partagée). **(2)** La proposition d'une solution basée sur l'algorithme génétique pour la fusion des résultats intermédiaires d'un Clustering parallèle dans un environnement distribué, dans le but d'exploiter ces résultats intermédiaires et d'améliorer le résultat final du Clustering. **(3)** La proposition d'une approche pour la distribution non-aléatoire des données sur l'ensemble des machines utilisées par le Framework MapReduce afin d'améliorer la qualité du Clustering final.

Dans un premier temps, la 2ème et la 3ème contributions étaient proposées principalement afin d'améliorer l'algorithme proposé dans la première contribution, après expérimentations on a constaté que ces améliorations proposées peuvent être appliquées à d'autres algorithmes de Clustering par partitionnement.

Ainsi, nous avons présenté nos différentes propositions sous forme d'un schéma global adapté à la parallélisation des algorithmes de Clustering par partitionnement dans le Framework MapReduce.

Les résultats obtenus lors de l'évaluation de notre contribution montrent l'importance des phases de distribution de données et de la fusion des résultats pour le Clustering dans un environnement distribué et leur influence sur les résultats du Clustering. Ces résultats montrent ainsi l'importance et l'efficacité de notre schéma parallèle proposé en améliorant la qualité du Clustering.

Mots-clés : Clustering de données, Parallélisme, MapReduce, Fusion de résultats, Distribution de données, Algorithme génétique.

Abstract:

Data Clustering is a problem that consists of splitting a set of data into groups, called Clusters. The distribution of data is based on the notion of similarity, so that similar data points belong to a same group.

Several Clustering algorithms have been proposed and used in various fields, but with the increase in the size of the processed data, and its distribution on a set of geographically distant machines, these algorithms become unsuitable to these situations. For this, the use of parallel or distributed algorithms become necessary.

The work of this thesis is in this context of study. It consists of studying Clustering algorithms in general and parallel Clustering algorithms in particular.

The objective of this doctoral thesis is to propose a parallel Clustering algorithm in order to analyze large data sets efficiently.

Our contribution is to propose a parallel schema for Large Data Clustering in a distributed environment based on the MapReduce Framework. The proposed contributions consist of 3 main points: (1) The proposal of a parallelization of an existing Clustering algorithm using the MapReduce Framework as well as a local parallelization on each machine (shared memory). (2) The proposal of a genetic algorithm-based solution for merging the intermediate results of parallel clustering in a distributed environment, in order to exploit these intermediate results and improve the final Clustering result. (3) The proposal of an approach for the distribution of data on the machines used by the MapReduce Framework to improve the quality of the final Clustering. In this approach we use a non-random data distribution.

Initially, the 2nd and 3rd contributions were proposed specifically to improve the algorithm proposed in the first contribution, after experiments it was found that these proposed steps can be applied to other partitioning Clustering algorithms.

Thus, we presented our different propositions in the form of a global schema adapted to the parallelization of partitioning Clustering algorithms in the MapReduce Framework.

The results obtained during the evaluation of our contribution show the importance of the data distribution and the results fusion phases for Clustering in a distributed environment and their influence on Clustering results. These results also show the effectiveness of our proposed parallel scheme by improving the quality of the Clustering.

Keywords: Data Clustering, Parallel processing, MapReduce, Results merging, Data distribution, Genetic algorithm.

Remerciement

L’achèvement de tout travail mené sur plusieurs années procure une grande satisfaction. Il est l’occasion de se remémorer les étapes passées et les personnes qui y ont contribué.

Mes plus sincère remerciement et sentiment de respect vont à ma directrice de thèse le Professeur Nadjet Kamel qui a dirigé ce travail et m’a orienté afin d’atteindre mon objectif. Je la remercie pour ses conseils, sa patience durant tout ce cursus de doctorat et pour la chance qu’elle m’a accordé pour terminer ce travail sans laquelle je n’y arriverais pas.

Je tiens également à remercier Madame Habiba DRIAS, Professeur à l’USTHB, Monsieur Riadh BABA ALI, Professeur à l’USTHB, Monsieur Abdelmadjid BOUKRA, Professeur à l’USTHB et Monsieur Lakhdar SAÏS, Professeur à l’université D’Artois, pour l'intérêt qu'ils ont porté à mes travaux en examinant cette thèse et pour l'honneur qu'ils m’ont fait en participant à ce jury.

Je remercie mes parents et ma grand-mère, qui, plus que quiconque ont souhaité me voir réussir et arriver à ce moment. Merci pour votre encouragement, votre soutien et vos sincères prières, je suis heureux que vos inquiétudes n’étaient pas en vains.

Je remercie ma famille, et spécialement ma sœur et mon frère qui étaient fiers de moi dès le commencement de ce parcours.

Je tiens à remercier mon ami Islam Bendjeddou pour son aide précieuse et sa disponibilité.

Enfin, j’adresse à travers ces mots mon remerciement à toute personne ayant participé à l’achèvement de ce travail, à toute personne m’ayant aidé à terminer ce parcours, que ce soit avec une action, une prière ou un mot d’encouragement.

Sommaire

Table des figures

Liste des tableaux

Introduction générale

1 Contexte et problématique

Le développement technologique rapide connu dans les dernières décennies, a conduit à une explosion dans les quantités de données disponibles. Ce qui a conduit à l'impossibilité de l'interprétation directe de ces dernières. Les techniques d'extraction de connaissances à partir des données (ECD) ont ainsi émerges [44].

L'extraction de connaissance à partir des données ou Knowledge Discovery from Data (KDD) implique plusieurs étapes différentes: la sélection, le nettoyage et la transformation des données, ces trois étapes sont incluses dans le processus du pré-traitement des données. Par la suite intervient l'étape du Data Mining où des modèles de données sont extraits à partir de ces dernières. Par la suite, une étapes d'évaluation et d'interprétation est appliquée sur les résultats obtenus lors de l'étape de Data Mining, afin que ces derniers peuvent être interprétés par l'utilisateur selon ses besoins.

Ainsi, on peut constater que l'étape ou la tâche du Data Mining représente le cœurs de l'ECD, c'est pourquoi dans certains cas ou contextes, les termes Data Mining et ECD sont confondus ou inter-changés.

Le Data Mining est une des branches de l'intelligence artificielle. Le Data Mining comporte plusieurs disciplines incluant: classification supervisée des données, classification non-supervisée ou Clustering de données et l'extraction des règles d'association..etc.

Le contexte de cette thèse est centré sur la classification non-supervisée des données.

L'objectif des algorithmes de Clustering est de grouper un ensemble de données en des sous-ensembles en se basant sur le concept de similarité ou de distance. Ainsi, les données ayant des caractéristiques similaires (proches) seront dans un même groupe et les données dissimilaires (éloignées) seront dans des groupes différents [63].

Plusieurs classes d'algorithmes de Clustering existent. Dans cette thèse, nous nous intéressons spécialement aux algorithmes de Clustering par partitionnement.

Un des principaux objectifs des travaux réalisés sur le Clustering, est de trouver un compromis entre la qualité des résultats et le temps d'exécution. Avec l'augmentation

significative du volume de données, le processus de Clustering devient une tâche coûteuse en terme de calcul.

Par conséquent, plusieurs solutions ont été proposées pour surmonter ce problème en utilisant le parallélisme. Les solutions proposées visent à optimiser le temps d'exécution tout en maintenant une qualité de Clustering proche ou identique à celle obtenue par une exécution séquentielle.

Le parallélisme pour le Clustering peut être appliqué en utilisant deux méthodes principales. La première méthode consiste en l'utilisation d'un réseau de machines connectées, où l'algorithme de Clustering est exécuté en utilisant un Cluster d'ordinateurs. La seconde méthode consiste en l'utilisation de la mémoire partagée ou l'utilisation des threads sur un seul processeur multi-cœur [78].

Outre la classification des méthodes de Clustering parallèles selon l'environnement ou l'infrastructure utilisée, on peut classifier les algorithmes de Clustering parallèles selon la manière dont les données sont traitées et acheminées durant ce processus (architecture) [82].

Cette classification consiste aussi en deux méthodes principales distinctes. La première consiste à exécuter en parallèle les différentes tâches de l'algorithme en question. Pour appliquer cette stratégie, les tâches indépendantes sont identifiées et exécutées simultanément et indépendamment par différents processus (ou threads) ou machines. Par conséquent, le schéma de cet algorithme de Clustering doit être modifié pour s'adapter à cette nouvelle forme parallèle. Par exemple, pour l'algorithme K-means, le calcul de la distance et la mise à jour des centroïdes sont effectués par différents processus ou machines. Comme cette stratégie suit le même comportement que l'algorithme séquentiel, la qualité du Clustering résultant est identique à celle de l'exécution séquentielle. La seule différence sera donc dans le temps d'exécution.

Dans la seconde stratégie, les données traitées sont divisées en plusieurs partitions de tailles plus petites que l'ensemble initial de données. Chacun de ces sous-ensembles est traité indépendamment sur un seul processeur ou machine en utilisant l'algorithme de Clustering séquentiel en question sans avoir à le modifier. Avec cette stratégie, nous obtenons des résultats intermédiaires provenant de chacune des partitions de données. Ces derniers seront fusionnés pour obtenir un résultat global final. Cette stratégie peut être appliquée avec presque tous les algorithmes de Clustering par partitionnement. Le schéma d'exécution parallèle dans cette stratégie diffère du séquentiel. Par conséquent, les Clusters générés seront différents de ceux obtenus à partir d'une exécution séquentielle. Les Clusters générés avec cette stratégie sont considérées dans la plupart des cas comme étant moins précis que ceux générés par une exécution séquentielle. Ceci est du au fait que chacun de ces Clusters intermédiaires représentent seulement le résultat d'une seule partie des données initiales.

La différence dans la qualité du Clustering est influencée par les méthodes utilisées dans les phases de distribution des données et de fusion des résultats intermédiaires. Pour les raisons citées ci-dessus, la plupart des travaux récents sur le Clustering parallèle sont basés sur la première stratégie de parallélisme pour éviter cet inconvénient.

L'implémentation d'une solution parallèle utilisant un Cluster de machines peut être simplifiée par l'utilisation du Framework MapReduce. MapReduce est un Framework permettant la parallélisation de certains types de tâches sans requérir des connaissances avancées sur la programmation distribuée [31]. Chaque instance exécutée sur le Framework MapReduce est appelée «Jobs» MapReduce.

Cependant, lors de l'application de la première stratégie avec des algorithmes itératifs tels que l'algorithme K-means, le processus nécessitera autant de « Jobs »(Instance d'un programme MapReduce) MapReduce que d'itérations nécessaires pour la version séquentielle de l'algorithme, ce qui ne représente pas une solution optimale en terme de temps d'exécution. Car l'initialisation d'un job MapReduce à chaque itération ralentira le temps d'exécution.

En effet, le modèle MapReduce n'est pas adapté aux algorithmes itératifs car ils nécessitent de nombreux Jobs MapReduce, augmentant ainsi le flux de données à travers le réseau, ce qui augmente à la fois le coût de communications entre les nœuds du réseau et le temps d'exécution. La seconde stratégie, en revanche, ne nécessite qu'un seul job MapReduce, ce qui lui évite les inconvénients de la première stratégie. Pour ces raisons, nous proposons une solution basée sur la seconde stratégie afin d'éviter les Jobs MapReduce itératifs. Mais, comme cité ci-dessus, cette seconde stratégie ne peut pas fournir une qualité de Clustering égale à celle des algorithmes séquentiels.

2 Contribution

Nous nous sommes intéressés dans cette thèse, dans un premier temps, à la proposition d'une parallélisation pour un algorithme de Clustering déjà existant (Sampling PSO K-means algorithm)[49]. Nous avons proposé une solution basée sur le Framework Mapreduce en utilisant un réseau de machines, et afin d'exploiter la puissance de calcul de chacune des machines, on a utilisé une parallélisation locale de l'algorithme sur chacune de ces machines [16].

Suite aux évaluations effectuées sur la première proposition, nous avons constaté que les méthodes standards utilisées pour la fusion des résultats intermédiaires du Clustering n'exploitaient pas efficacement ces derniers.

Pour cela, dans notre deuxième contribution, nous avons proposé une solution à base de Meta-heuristiques (algorithme génétique) pour sélectionner les centroïdes les plus pertinents pour la solution finale [17].

Dans le but d'améliorer les propositions précédentes, nous avons proposé dans une dernière contribution une solution pour la phase de distribution des données sur les machines du Cluster. Généralement les méthodes utilisées consistent à distribuer les données aléatoirement. Dans notre solution nous avons proposé l'utilisation d'une distribution non-aléatoire des données. Dans cette solution, la distribution des données est effectuée de manière à maximiser la distance entre les données présentes sur chaque machine dans le but d'obtenir des Clusters intermédiaires les plus dissimilaires possible.

Après tests et expérimentations, nous avons présenté nos différentes propositions sous forme d'un Framework adapté à la parallélisation des différents algorithmes de Clustering par partitionnement.

Ainsi, notre contribution globale consiste à proposer un nouveau modèle de flux de données dans un environnement MapReduce en utilisant des stratégies non conventionnelles pour la distribution de données et la fusion des résultats [18]. Ceci assurera un bon temps d'exécution et une qualité de Clusters supérieure à celle obtenue avec les stratégies parallèles déjà proposées ou avec une exécution séquentielle. Ainsi, l'amélioration apportée par notre approche réside dans les méthodes utilisées pour la distribution de données et la fusion des résultats.

3 Plan de la thèse

Cette thèse est organisée en six chapitres comme suit :

- Le chapitre 1 présente une introduction au Clustering de données. Les différentes formes de Clustering et les différents algorithmes de Clustering séquentiels sont présentés. Nous discutons et analysons les caractéristiques et la complexité des algorithmes de Clustering les plus utilisés.
- Dans le chapitre 2 nous introduirons le concept du Clustering parallèle. On présentera les différentes méthodes utilisées pour la parallélisassion des algorithmes de Clustering, les différentes classifications de ces derniers selon les méthodes utilisées.
- Le chapitre 3 présente notre première contribution qui consiste en une parallélisation locale dans un environnement distribué de l'algorithme « sampling pso kmeans algorithm ».

- Dans le chapitre 4 nous présenterons le concept de la fusion des résultats de Clustering dans un environnement parallèle, et par la suite notre contribution pour la fusion des résultats avec l'algorithme génétique.
- Le chapitre 5 présente notre contribution principale qui consiste en la proposition d'un schéma pour le Clustering parallèle des données dans l'environnement MapReduce.
- Le chapitre 6 présentera une conclusion générale.

Chapitre 1

Le Clustering de données

1 Introduction

Le Data Clustering est l'une des plus importantes méthodes pour l'analyse des données. Le Clustering permet de découvrir des modèles de données cachés (Clusters) dans un ensemble de données non structurés [63]. Donc on peut voir le Clustering comme une méthode de modélisation de données. D'une perspective d'apprentissage automatique, le Clustering peut être décrit comme une méthode d'apprentissage (classification) non supervisé [7]. Dans ce chapitre nous présentons le concept général du Clustering de données, la définition du Clustering et les différents types et algorithmes de Clustering. Nous présenterons aussi les méthodes utilisées pour l'évaluation d'une solution de Clustering.

Comme l'objectif principal de cette thèse est de proposer une solution parallèle pour un problème de Clustering dans le but d'optimiser ses performances, nous traitons dans ce chapitre le Clustering d'une perspective d'optimisation de performance en terme de puissance de calcul. Ainsi nous discuterons les avantages et les inconvénients des différents algorithmes du coté performance (complexité).

2 Définition du Clustering de données

La Clustering ou la classification non-supervisée de données est une des branche du Data Mining. Le Clustering consiste à partitionner un ensemble de données en des groupes qu'on appelle Clusters, de telle sorte qu'il existe une forte similarité entre les données d'un même Cluster et une faible similarité entre les données appartenant à des Clusters différents. Cette méthode permet de constituer des groupes homogènes d'objets, sans se baser sur des modèles préétablis, d'où l'appellation de classification non-supervisée [7]. Les algorithmes

de Clustering sont utilisés afin de déterminer et d'identifier ces groupes. Le Clustering de données a des applications dans de divers domaines tels que le marketing, la médecine, la sécurité, et les réseaux sociaux...etc [13].

Dans le cas le plus basique (hard Clustering), le Clustering d'un ensemble de données X contenant n points de données sur K Clusters doit vérifier les 3 propriétés suivantes:

1. Chaque Cluster doit contenir au moins un point de données:

$$C_i \neq \emptyset, \forall i \in \{1,2,\ldots,K\}$$

2. Deux Clusters ne doivent pas avoir des points de données en commun:

$$C_i \bigcap C_j = \emptyset, \forall i \neq jeti, j \in \{1,2,\ldots,K\}$$

3. Chaque point de données doit appartenir à un des Clusters:

$$\bigcup_{i=1}^{K} C_i = X$$

Il existe plusieurs types d'algorithmes de Clustering. La classification des algorithmes de Clustering peut se baser sur plusieurs critères donnant ainsi différentes classifications possibles [12].

La plus connue des méthodes est celle qui classifie les algorithmes selon la méthode adoptée pour définir les Clusters. Ce qui nous donne les catégories suivantes:

- Clustering par partitionnement (Partitional Clustering)
  - Clustering par partitionnement avec relocalisation.
  - Clustering par densité.
- Clustering hiérarchique (Hierarchical Clustering)
  - Clustering hiérarchique divisif.
  - Clustering hiérarchique agglomératif.

On présentera en détail dans les sections suivantes ces catégories de Clustering avec des exemples d'algorithmes de chaque catégorie. Outre ces catégories principales, on présentera d'autres catégories de Clustering comme le grid-Clustering et le Clustering par contraintes.

Les algorithmes appartenant à ces différentes classes de Clustering sont appliqués à un problème donné en fonction du type des données, de la complexité de l'algorithme et de sa pertinence avec le problème traité ainsi qu'à la nature des résultats visés.

Comme notre contribution est basée sur les algorithmes de Clustering par partitionnement, ces derniers seront plus détaillés que les autres types de Clustering.

3 Types de Clustering (classes)

La classification des algorithmes de Clustering s'avère être une tâche ambiguë et complexe. En effet, les différentes classes d'algorithmes se chevauchent, donnant ainsi des cas où des algorithmes de Clustering peuvent appartenir à différentes classes. Dans une première analyse, les algorithmes sont divisés en deux catégories distinctes :

– Clustering par partitionnements.

– Clustering hiérarchiques.

Les algorithmes de Clustering hiérarchiques assemblent (agglomératifs) ou désassemblent (divisifs) progressivement l'ensemble de données en des Clusters. Tandis que les algorithmes par partitionnement cherchent à découvrir les Clusters entièrement et directement. Les méthodes de Clustering par partitionnement résultent des partitions d'objets à un seul niveau. Tandis que les méthodes hiérarchiques construisent une partition imbriquée des objets résultant en un arbre de Clusters.

Dans ce qui suit nous présenterons les principaux types de Clustering avec des exemples d'algorithmes de chaque catégorie.

3.1 Clustering par partitionnement

Les algorithmes de Clustering par partitionnement visent à découvrir les Clusters soit, en relocalisant itérativement les points de données entre les Clusters jusqu'à atteindre un état stable, soit en découvrant des zones avec une densité élevée de données. La première catégorie se nomme: Clustering par partitionnement avec relocalisation (communément appelé Clustering par partitionnement) et la deuxième se nomme Clustering par densité [90].

3.1.1 Clustering par partitionnement avec relocalisation

Les méthodes par partitionnement ont pour but de fournir un partitionnement des données en K classes, où le nombre des classes (Clusters) à générer doit être fixé au préalable. Le principe des méthodes de Clustering par partitionnement est qu'à partir d'un ensemble de données E de n individus, on cherche à construire un partitionnement contenant K classes homogènes et distinctes, tout en respectant un critère basé sur une distance entre les individus. Les algorithmes appartenant à cette catégorie peuvent encore être classifiés en 3 sous catégories : **(1)** Clustering probabiliste. **(2)** les algorithmes basés sur les centroïdes (basés sur K-moyennes). **(3)** les algorithmes basés sur les médoïdes (basés sur K-médoïdes) [47].

Trouver un partitionnement des données revient à vérifier toutes les combinaisons possibles sur la répartition des objets entre les Clusters, ce qui est irréalisable. Pour remédier à ce problème, des heuristiques gloutonnes sont utilisées itérativement pour obtenir une solution optimale locale. Contrairement aux méthodes hiérarchiques, où les Clusters une fois créés ne seront jamais mis à jour, les méthodes de Clustering par partitionnement permettent progressivement d'améliorer les Clusters. Ce qui garantira avec des données adéquates une meilleure qualité de Clustering.

Parmi les algorithmes de Clustering par partitionnement on peut citer: K-moyenne (K-means), K-médoïdes (PAM) ou le Clustering avec l'Algorithme Génétique ...etc. Dans ce qui suit nous présentons différents algorithmes de Clustering par partitionnement avec relocalisation selon les 3 sous catégories citées auparavant.

3.1.1.1 L'algorithme K-means (Les algorithmes basés sur K-means)

L'algorithme K-moyenne ou K-means, est la plus basique forme des algorithmes de partitionnement basé sur les centroïdes. Il est considéré comme la plus simple et la plus populaire méthode de Clustering classique pour sa facilité d'implémentation et son efficacité. K-means segmente les données en K groupes ou Clusters de façon à ce que les données similaires soient dans le même groupe et elles sont représentées par leurs centres ou moyennes. Le centre de chaque groupe est calculé comme moyenne de toutes les données appartenant à ce groupe. Le succès de K-means est dû au fait que cet algorithme présente un rapport coût/efficacité avantageux.

L'algorithme fonctionne selon les étapes suivantes :

- Choisir k objets (centres) formant ainsi k Clusters.
- Affecter chaque objet au Cluster dont le centre est le plus proche.
- Recalculer le centre de chaque Cluster et répéter les étapes précédentes jusqu'à convergence ou le nombre maximal d'itérations est atteint.

Le principal inconvénient de l'algorithme K-means est que la partition finale dépend du choix de la partition initiale. L'algorithme est sensible à la sélection initiale des centroïdes. Trouver la configuration optimale pour les centroïdes de départ est un problème d'optimisation combinatoire.

Un autre inconvénient de l'algorithme K-means, est qu'il nécessite à l'utilisateur de lui fournir le nombre K de classes (une information souvent peu disponible). Comme les centres des Clusters représentent la moyenne des objets appartenant à chacun de ces Clusters,

K-means est mieux adapté pour la classification des données numériques continues. Avoir un «outliner» dans l'ensemble des données peut affecter grandement la précision du résultat.

Complexité

La complexité spatiale de K-means est de $O(m*n)$, où m est le nombre de points de données et n est le nombre d'attributs, tandis que sa complexité temporelle est de $O(I*k*m*n)$ où I est le nombre d'itérations exigées pour la convergence. Cette complexité linéaire est l'une des raisons de la popularité de K-means contrairement aux autres algorithmes dont leur complexité est non linéaire. Mais avec l'augmentation de la taille des données traitées et le nombre d'attributs, K-means devient un algorithme très gourmand en terme de ressources de calcul.

Les algorithmes basés sur K-means:

Comme cité dans la section précédente, l'algorithme K-means présente plusieurs inconvénients que ce soit en terme de performances de calcule ou en qualité de Clustering. Plusieurs solutions basées sur l'algorithme K-means ont été proposées afin de remédier à ces problèmes.

Dans [19], Bradley et Fayyad ont proposé une solution basée sur l'échantillonnage de données pour remédier au problème de la sensibilité de K-means au centroïdes de départ. Dans cette solution, K-means est exécuté sur plusieurs petits échantillons de données. Chacun des résultats obtenus des échantillons est considéré comme une solution potentielle pour les centroïdes de départ. La meilleure configuration de Clusters est utilisée ainsi comme centroïdes initiaux pour la classification de tout l'ensemble des données initiales.

Dans [66], les auteurs ont proposé d'utiliser un Clustering hiérarchique pour la sélection des centres initiaux, dans un premier temps un Clustering hiérarchique est appliqué sur les données, les centroïdes résultants de cet algorithme sont utilisés avec K-means pour améliorer ses résultats.

L'algorithme K-means++ [6], est une amélioration de l'algorithme K-means classique. K-means++ propose une sélection non aléatoire des centroïdes initiaux. Les k centroïdes sont choisis d'une manière progressive. Le premier centroïde est choisi aléatoirement, et les $K-1$ prochains centroïdes sont choisis d'une manière à maximiser la distance entre ce centroïde et les centroïdes déjà sélectionnés.

Une des solutions les plus connues pour le problème d'initialisation de K-means, est son hybridation avec les différentes méta-heuristiques.

Dans [56], l'algorithme génétique est utilisé pour générer une solution pour les centroïdes initiaux. Cette solution vise à trouver une solution optimale en évitant à K-means de converger vers un minimum local.

Du même principe, plusieurs autres solutions à base d'autres méta-heuristiques ont été proposées. Dans [24][2], des solutions basées sur l'algorithme PSO ont été proposées pour pallier au problème des centroïdes initiaux. De la même manière, l'algorithme colonie de fourmis est utilisé dans [73][69] pour le raffinement des centroïdes initiaux.

Outre ces solutions visant à améliorer la qualité des résultats de K-means, d'autre approches ont été proposées afin d'améliorer principalement son temps d'exécution.

Dans [27], les auteurs ont proposé une solution pour réduire le temps d'exécution de K-means lorsque la taille des données est importante. Les auteurs ont proposé de diviser aléatoirement les données en de plus petits sous-ensembles et d'appliquer K-means sur chacun. Ensuite, les solutions générées sur chacun des sous-ensembles sont utilisées pour classifier les données entières. La solution avec les meilleurs résultats est considérée comme la solution finale. Avec cette approche le temps d'exécution est légèrement réduit. En effet, appliquer K-means sur des ensembles de données moins volumineux nécessitera un nombre d'itérations moins important que celui nécessaire pour traiter les données à part entière.

Dans une autre solution [74], on a proposé dans un premier temps d'appliquer l'algorithme K-means sur une petite partition de données, en suite, les résultats obtenus sont utilisés sur l'ensemble entier des données. Avec cette solution l'algorithme K-means convergera en moins d'itérations ce qui réduira le temps d'exécution global.

Dans [37], Fahim et al, ont présenté l'algorithme « An efficient enhanced K-means Clustering algorithm » dans le but de réduire le temps d'exécution de K-means. L'amélioration apportée se passe lors de l'étape de calcule des distances entre les points de données et les nouveaux centroïdes générés. Les auteurs proposent qu'à chaque itération on sauvegarde les distances entre les points de données et leurs centroïdes respectives. Dans la prochaine itération, pour chaque point de données, on commence par calculer sa distance par rapport au centroïdes du Cluster auquel il appartient. Si cette distance est inférieure ou égale à la distance entre ce point et son centroïde précédent, on garde alors ce point dans son actuel Cluster et on ne vérifie pas sa distance/similarité avec les autres nouveaux centroïdes. On évitera ainsi de calculer la distance avec les autres $K-1$ centroïdes. Cette solution optimisera le temps d'exécution spécialement pour les cas où le nombre de Clusters K est élevé.

Dans [57], les auteurs ont amélioré la solution proposée par Fahim et al [37]. La solution vise aussi à améliorer l'étape de calcul des distances pour les points de données. Les auteurs ont proposé de trier les centroïdes de chaque itération de manière ascendante selon leurs normes. Ces valeurs sont utilisées pour estimer le prochain centroïde avec lequel chaque

point de données et comparé. Cette solution permet de réduire le nombre d'opérations de calcul des distances, ce qui optimisera le temps d'exécution global.

3.1.1.2 L'algorithme K-médoïdes (Les algorithmes basés sur k-médoïdes)

L'algorithme K-médoïdes, également appelé l'algorithme PAM (Partitionnement Autour de Médoïdes), est un algorithme de Clustering basé sur le partitionnement. PAM partage la même structure générale que l'algorithme K-means. A la différence de K-means, les centres des Clusters sont représentés par des médoïdes. Les centres des Clusters représentent les médianes des éléments de chacun des Clusters. Ainsi, chaque centre de Cluster sera un des individus appartenant à l'ensemble des données [52]. Le fonctionnement de PAM consiste à choisir initialement K éléments de l'ensemble de données qui sont considérés comme les médoïdes des K Clusters.

Ensuite, à chaque itération, tous les éléments de l'ensemble de données qui ne sont pas des médoïdes sont examinés pour vérifier s'ils peuvent être considérés comme des médoïdes dans leurs Clusters respectifs. La qualité du Clustering est mesurée par la somme de toutes les distances des objets non-médoïdes au médoïde du Cluster auquel ils appartiennent. L'algorithme PAM est considéré plus robuste que K-means en présence de données et valeurs aberrantes (outliners); parce qu'un médoïde est moins influencé par des valeurs aberrantes ou d'autres valeurs extrêmes qu'une moyenne. De plus de son insensibilité aux outliners, l'algorithme PAM est mieux adapté à l'analyse des données catégoriques et numériques discrètes.

Compléxité:

En dépit des avantage rapportés par l'algorithme PAM par rapport à l'algorithme K-means, la tâche de calcul des nouveaux médoïdes s'avère être très coûteuse contrairement à la mise à jour des centroïdes pour l'algorithme K-means. La complexité temporelle de PAM est de l'ordre de $O(i*k*(n-k)^2)$, ce qui nous donne une complexité quadratique.

Les algorithmes basés sur PAM

L'avantage principal de l'algorithme PAM est l'utilisation des médoïdes au lieu des centroïdes, ce qui lui donne une robustesse vis-à-vis des 'outliners'. Par contre, comme déjà mentionné, l'algorithme est très couteux, spécialement lors de l'étape de calcul des nouveaux médoïdes.

Pour remédier à ce problème, plusieurs variations et optimisation ont été proposées pour l'algorithme PAM. Dans ce qui suit nous discuterons les algorithmes basés sur PAM présentés dans la littérature.

Kaufmann and Rousseeuw [51] ont proposé l'algorithme CLARA qui est une amélioration de algorithme PAM. Étant donné que l'algorithme PAM n'est pas adapté au traitement de larges ensembles de données, CLARA vise à résoudre ce problème en proposant d'exécuter PAM sur plusieurs échantillons de données au lieu d'analyser l'ensemble entier des données. Selon les expériences reportées, plusieurs échantillons de la taille de $40+2k$ donneront des résultats satisfaisants. A chaque itération, on applique l'algorithme PAM sur l'un des échantillons de données. Les k médoïdes générés sont utilisés pour classifier l'intégralité des données. Si les résultats montrent que cette solution de k médoïdes donne les meilleurs résultats comparée à l'itération précédente, elle sera retenue comme meilleure solution. CLARA réduira la complexité quadratique. Mais les résultats ne sont pas aussi précis que ceux générer avec PAM vu que CLARA n'analyse pas l'intégralité des données.

L'algorithme CLARANS a été proposé par Ng et Han [68] afin d'améliorer l'efficacité de l'algorithme CLARA. L'algorithme peut être vu comme une recherche dans un graphe. Chaque nœud de ce graphe représente K médoïdes potentiels. Deux nœuds sont reliés par une arête (voisins) s'ils différent d'un seul médoïde. Donc chaque nœud aura $k(n-k)$ voisins possibles. Les paramètres principaux de l'algorithme sont: la valeur «maxVoisins» qui correspond au nombre maximal de nœuds voisins à évaluer pour chaque nœud et la valeur «localMin» qui correspond à la meilleure solution locale trouvée jusqu'à l'itération courante. Avec cette solution, CLARANS parcourt plusieurs solutions possibles en utilisant toutes les données contrairement à CLARA qui parcourt toutes les solutions possibles dans un espace de données réduit. Ce qui sera plus couteux et dans un ensemble de donnée partiel, ce qui réduira la précision et ralentira le temps d'exécution.

3.1.1.3 Clustering flou et probabiliste

Dans les algorithmes classiques de Clustering, chaque objet appartient à seulement un Cluster selon une mesure de distance (le plus proche). Dans certains cas un objet peut être difficile à classifier étant donné que la différence de distance par rapport à deux ou plusieurs centres de Cluster est très proche. Avec les solutions classiques l'objet est assigné au plus proche Cluster sans prendre en compte son rapprochement des autres Clusters, c'est ce qu'on appelle Hard-Clustering. Tandis qu'avec le Clustering basé sur la théorie des ensembles flous et celui basé sur la théorie des probabilités, un objet peut appartenir à un ou plusieurs Clusters à la fois selon un degré d'appartenance ou selon une distribution de

probabilité[41]. Ce type de Clustering est appelé aussi Soft-Clustering. Cette description peut nous permettre de classifier les algorithmes de Clustering en deux catégories: Soft et Hard Clustering. Dans ce qui suit nous présenterons deux algorithmes de Clustering appartenant à la classe Soft-Clustering qui sont: l'algorithme K-means flou et l'algorithme EM(Expectation Maximization).

K-means flou

L'algorithme K-means flou (Fuzzy K-means) ou C-means, est le plus connu des algorithmes de Clustering flous. Le principe de C-means est le même que celui de K-means, à savoir, générer des Clusters en minimisant la distance entre les éléments d'un même Cluster et en la maximisant entre les éléments des différents Clusters. En effet, les résultats de C-means ne représentent pas une distribution des données sur des Clusters, mais des informations concernant une potentielle distribution de ces éléments [14]. L'algorithme C-means fonctionne selon les étapes suivantes :

- Choisir k objets aléatoirement formant ainsi k Clusters.
- Calculer le degré d'appartenance de chaque objet.
- Recalculer le centre de chaque Cluster et répéter les étapes précédentes jusqu'à convergence ou le nombre maximal d'itérations est atteint.

C-means vise à minimiser la fonction objective présentée par la formule (1.1) :

$$\min_{c,u} Q_{\mathrm{FKM}}(c,u) = \min_{c,u} \sum_{k=1}^{n_k} \sum_{x_i \in \mathscr{X}} u_{ik}^{\beta} \|x_i - c_k\|_2^2 \tag{1.1}$$

Avec:

$$\sum_{k=1}^{n_k} u_{ik} = 1 \quad \forall x_i \in \mathscr{X}$$

et

$$u_{ik} \geq 0 \quad \forall x_i \in \mathscr{X}, \forall k \in [1 \ldots n_k]$$

Où le degré d'appartenance u_{ik} est calculé selon la formule (1.2) :

$$u_{ik}^{*} = \frac{\|x_i - c_k\|_2^{2/(1-\beta)}}{\sum_{j=1}^{n_k} \|x_i - c_j\|_2^{2/(1-\beta)}} \forall x_i \in \mathscr{X}, \forall k \in [1.n_k] \tag{1.2}$$

Les centroïdes sont mis à jour avec la formule (1.3) :

$$c_k^{*} = \arg\min_{c \in \mathbb{R}^p} \sum_{x_i \in \mathscr{X}} u_{ik}^{\beta} \|x_i - c\|_2^2 \tag{1.3}$$

On note que l'algorithme K-means peut être considéré comme un cas spécial de l'algorithme C-means où tous les degrés d'appartenances aux Clusters les plus proches sont égaux à 1.

Complexité: La complexité temporelle de C-means est: $O\left(n*d*k^2*i\right)$, où n est le nombre de points de données, d la dimension des données, k le nombre de Clusters et i le nombre d'itérations.

L'algorithme EM (Expectation Maximization)

Contrairement à l'algorithme K-means, EM (Expectation Maximization) repose sur l'hypothèse que les objets de l'ensemble de données ont des attributs dont les valeurs sont distribuées selon une combinaison linéaire (inconnue) de distributions de probabilités. L'algorithme EM est un algorithme de raffinement itératif populaire qui peut être utilisé pour trouver les estimations des paramètres. Il peut être vu comme une extension pour l'algorithme K-means. Au lieu d'affecter chaque objet à un Cluster précis, EM affecte chaque objet à un Cluster en fonction d'un poids représentant la probabilité d'appartenance. Donc, il n'y a pas des limites strictes entre les Clusters. En pratique, il converge plus rapidement que K-means mais il peut ne pas atteindre une solution optimale globale [67].

Complexité: La complexité temporelle est linéaire en d (le nombre d'attribut en entrée), n (le nombre d'objets) et t (le nombre d'itérations).

3.1.2 Clustering par densité

Le Clustering par densité fait partie des méthodes de Clustering par partitionnement. A la fin du processus de Clustering on aura comme résultats des partitions de données distinctes. Les Clusters sont définis en découvrant des zones avec une densité élevée de données [55]. Le Clustering par densité propose une flexibilité au niveau des formes des Clusters générés. Ils peuvent donc découvrir des Clusters avec des formes irrégulières contrairement aux algorithmes basés sur la relocalisation. Vue leur stratégie de procéder, ces algorithmes sont plus robustes avec les outliners. Le Clustering par densité est plus adapté aux données spatiales avec un nombre de dimension réduit (attributs). Les résultats peuvent être interprétés comme une segmentation d'un espace de données (Distribution de données).

Pour bien expliquer le Clustering par densité nous présenterons dans ce qui suit l'algorithme DBSCAN proposé par Ester, et al [36].

L'algorithme DBSCAN dépend de deux paramètres principaux : ε et $MinPts$. Ces deux paramètres déterminent la manière dont l'algorithme DBSCAN procède pour découvrir

l'ensemble des zones de données denses (Clusters). Chaque objet des données peut être soit un individu dit « cœur » soit un individu dit « frontière ».

Soit x_i un objet appartenant à l'ensemble de données X. x_i est considéré comme un point « cœur » s'il a dans son voisinage au moins $MinPts$ points ne dépassant pas la distance ε. Sinon x_i et considéré comme un point frontière.

Le voisinage de chaque individu est défini par la formule (1.4) :

$$\mathscr{N}_\varepsilon(x_i) = \left\{x_j \in \mathscr{X} | d\left(x_i, x_j\right) \leq \varepsilon\right\} \tag{1.4}$$

Ainsi, si la valeur $\mathscr{N}_\varepsilon(x_i)$ du voisinage est supérieure ou égale à $MinPts$, le point x_i est un cœur, sinon c'est un point frontière.

Suivant ce principe, DBSCAN parcourt récursivement l'ensemble des points jusqu'à trouver un point cœur x_i, qui devient un générateur de groupe ou source du groupe.

Les voisins de x_i qui n'appartient pas déjà à un groupe seront affectés au même groupe que x_i. Les nouveaux voisins de x_i qui sont des points cœurs se chargeront de propager le groupe jusqu'à plus aucun point cœur n'est disponible. Dans ce cas, le groupe (Cluster) est considéré défini. Cette opération est réitérée pour tous les points de données. À la fin du processus, si des points frontières ne sont affectés à aucun groupe, ils seront considérés comme des outliners. D'après ces définitions on peut introduire les notions suivantes :

- Un point x_j est dit directement atteignable en densité par x_i si x_j appartient au voisinage de x_i , et que x_i est un point cœur.
- Un point x_j est dit atteignable en densité par x_i si: il est directement atteignable en densité par x_i, ou : il existe un point x_k qui est un individu cœur, et que x_j est directement atteignable par x_k et x_k est atteignable par x_i.

Avec cette stratégie DBSCAN a l'avantage de trouver le nombre de groupes sans qu'il soit défini par l'utilisateur. Par contre la définition des paramètres ε et $MinPts$ reste difficile et ambiguë. DBSCAN est présenté dans l'algorithme ci-dessous. L'ensemble R représente les points frontières n'appartenant à aucun groupe.

Algorithme DBSCAN

ENTRÉES : $\mathcal{X}, MinPts, \epsilon$
SORTIES : $C = \{C_1, ..., C_{n_k}\}, \mathcal{R}$
1 : $i = 1, k = 1$ et $\mathcal{R} = \emptyset$
2 : $C_k = \emptyset$
3 : Tant que $|\mathcal{N}_\epsilon(x_i)| < MinPts$ et $x_i \notin \bigcup_{1 \leq g \leq k} C_g$ Faire i++, $\mathcal{R} = \mathcal{R} \cup \{x_i\}$
4 : $C_k = C_k \cup \{x_i\} \cup \mathcal{A}(x_i)$
5 : Si $\exists x_j \in \mathcal{X}$ tel que $x_j \notin \bigcup_{1 \leq g \leq k} C_g \cup \mathcal{R}$ alors k++ et aller en **2**.

3.1.2.1 Compléxité: La complexité théorique temporelle de DBSCAN dans le cas des données spatiales avec des dimensions réduites est de l'ordre de $O(Nlog(N))$.

3.2 Clustering hiérarchique

Les méthodes de Clustering hiérarchiques construisent une partition imbriquée des objets résultant en une structure hiérarchique de Clusters appelée dendrogramme [48]. Un dendrogramme est un arbre dans lequel chaque niveau correspond à une partition de l'ensemble de données. Chaque nœud de l'arbre correspondant à un groupe d'individu (Cluster) appelé *amas*, et l'ensemble des fils de ce nœud correspondant à leur tour au partitionnement de ce nœud. Il existe deux méthodes possibles pour construire un dendrogramme, les méthodes agglomératives et les méthodes divisives. Les méthodes agglomératives (ascendantes) commencent à construire le dendrogramme par la base où chaque objet est dans un amas (Cluster) individuel. A chaque étape, l'algorithme regroupe les amas les plus similaires en de plus grand amas. Les méthodes divisives (descendantes) commencent à construire le dendrogramme par le sommet. Dans chaque étape un amas d'individu est divisé en des plus petits sous-amas.

Contrairement aux autres méthodes de Clustering, les méthodes hiérarchiques peuvent ne pas prendre aucun paramètre en entrée. Dans ce qui suit, nous présenterons les deux principaux algorithmes de Clustering hiérarchiques: la méthode agglomérative et la méthode divisive.

3.2.1 Méthode divisive

L'approche divisive crée un dendrogramme de manière descendante. L'amas initial A contient l'ensemble de tous les individus. Un amas est caractérisé par un diamètre qui représente la distance entre les deux individus les plus dissimilaires [43]. Le diamètre d'un amas est calculé avec la formule (1.5):

$$\text{Diam}(A) = \max_{x_i \in A, x_j \in A} d\left(x_i, x_j\right) \tag{1.5}$$

L'algorithme partitionne l'amas initial itérativement en deux amas A' et A''. Au début de l'algorithme, l'amas A' contient tous les individus initiaux (A) et A'' est vide.

L'algorithme procède à déplacer les individus estimés les plus dissimilaires possible de l'ensemble des individus de A' vers A''. À chaque itération, l'individu qui maximise la formule (1.6) ci-dessous sera transféré vers A'':

$$D\left(x_i,A'\setminus\{x_i\}\right)=\frac{1}{|A'|-1}\sum_{\substack{x_j\in A'\\ x_j\neq x_i}} d\left(x_i,x_j\right) \tag{1.6}$$

Cette formule calcule la distance moyenne d'un individu x_i par rapport aux autres individus de son amas actuel. Ce processus de transfert d'individus est exécuté itérativement jusqu'à ce que la valeur définie par la formule (1.7) suivante devienne négative:

$$\Delta\left(A',\overline{A'},x_i\right)=D\left(x_i,A'\setminus\{x_i\}\right)-D\left(x_i,\overline{A'}\right) \tag{1.7}$$

Ce processus crée un équilibre entre les deux sous amas créés à partir de A.

La subdivision des amas est ré-exécutée récursivement en partant par l'amas avec le plus grand diamètre entre A' et A''.

L'algorithme ci-dessous résume le fonctionnement de la méthode du Clustering hiérarchique divisive.

Algorithme

ENTRÉES : $\mathcal{X}$, $d(.,.)$
SORTIES : $\mathcal{D}$
1 : $A=\mathcal{X}$, $\mathcal{D}_0=\{A\}$ et $i=1$
2 : $A'=\underset{A\in\mathcal{D}_{i-1}}{arg\,max}\, Diam(A)$, $\overline{A'}=\emptyset$ et $\mathcal{D}_i=\mathcal{D}_{i-1}\setminus A'$
3 : choisir $x_i^*=\underset{x_i\in A'}{arg\,max}\, D(x_i,A'\setminus\{x_i\})$
4 : si $\Delta(A',\overline{A'},x_i^*)\geq 0$ alors $A'=A'\setminus\{x_i^*\}$, $\overline{A'}=\overline{A'}\cup\{x_i^*\}$ et aller en **3**
5 : si $i<|\mathcal{X}|$ alors $i=i+1$, $\mathcal{D}_i=\mathcal{D}_i\cup A'\cup\overline{A}$ et aller en **2**

3.2.1.1 Inconvénient de la méthode divisive L'un des inconvénients des méthodes divisives est leur sensibilité aux outliners. Un outliner peut maximiser la valeur du diamètre d'un amas ce qui influencera négativement sur le processus de division. Outre ce problème, les méthodes divisives requièrent un temps d'exécution important, pour ce fait les méthodes agglomératives qu'on présentera dans la section suivante; restent les plus utilisées.

3.2.2 Méthode agglomérative

Contrairement aux méthodes divisives, les méthodes agglomératives commencent par créer un ensemble d'amas à partir des individus initiaux, où chaque individu représente un amas particulier. Ensuite, suivant un processus itératif, les amas les plus similaires sont fusionnés pour obtenir de nouveaux amas [30]. Le calcul de similarité entres amas ce fait selon plusieurs mesures : *SLINK*, *ALINK* et *CLINK*. Le résultat de l'algorithme dépendra de la méthode utilisée pour le calcul de la similarité.

3.2.2.1 SLINK (simple link) La distance entre deux amas représente la distance minimale entre deux individus des deux amas comparés. SLINK est calculé avec la formule (1.8) :

$$D\left(A_i,A_j\right) = \min_{x_i \in A_i, x_j \in A_j} d\left(x_i,x_j\right) \tag{1.8}$$

3.2.2.2 CLINK (complet link) La distance entre deux amas représente la distance maximale entre deux individus des deux amas comparés. CLINK est calculé avec la formule (1.9) :

$$D\left(A_i,A_j\right) = \max_{x_i \in A_i, x_j \in A_j} d\left(x_i,x_j\right) \tag{1.9}$$

3.2.2.3 ALINK (average link) La distance entre deux amas représente la distance entre les centres de ces deux amas suivant la formule suivante (1.10):

$$D\left(A_i,A_j\right) = d\left(c_i,c_j\right) \tag{1.10}$$

Les centres des amas A_i et A_j qui sont c_i et c_j respectivement, sont calculés selon la formule (1.11) suivante :

$$c_i = \frac{1}{|A_i|} \sum_{x_i \in A_i} x_i \tag{1.11}$$

L'algorithme ci-dessous décrit le fonctionnement de la méthode agglomérative.

Algorithme

ENTRÉES : $\mathcal{X}, d(.,.)$
SORTIES : $\mathcal{D}$
1 : $\forall x_i \in \mathcal{X}, A_i = \{x_i\}, \mathcal{D}_0 = \{A_i\}_{1 \leq i \leq |\mathcal{X}|}$ et $i' = 1$
2 : $(A_1^*, A_2^*) = \underset{(A_1,A_2) \in \mathcal{D}_{i'-1}^2}{arg\,min} \; D(A_1, A_2)$
3 : $\mathcal{D}_{i'} = \mathcal{D}_{i'-1} \setminus \{A_1^*\} \cup \{A_2^*\}$ et $\mathcal{D}_{i'} = \mathcal{D}_{i'-1} \cup \{A_1^* \cup A_2^*\}$
4 : si $i' < |\mathcal{X}|$ alors $i' = i' + 1$ et aller en **2**

3.3 Clustering basé sur les grilles

Les méthodes de Clustering par grille consistent à explorer l'espace des données pour extraire des Clusters. L'espace de données est partitionné en un nombre fini de cellules qui forment une structure de grille. Chaque objet de données est assigné à une cellule. Les cellules sont analysées pour détecter des zones de données denses qui représentent des Clusters [71]. Si la densité d'une cellule est inférieure à un seuil donné, elle sera ignorée est éliminée de la grille. Ces méthodes sont utilisées principalement pour le Clustering des données spatiale. Parmi les algorithmes basés sur les grilles on peut citer : STING (Statistical Information Grid) [86], Wave Cluster (Clustering Using Wavelet Transformation) [3].

3.4 Clustering basé sur les contraintes

Le Clustering basée sur les contraintes consiste à trouver des Clusters répondant aux préférences ou aux contraintes spécifiées par l'utilisateur [84]. Les contraintes utilisées peuvent concerner la sélection des paramètres de Clustering ou les fonctions de calcul de distance ou de similarité utilisées. Les contraintes peuvent être utilisées pour exiger la présence dans un même Cluster ou la séparation de certains individus qui vérifient certaines contraintes. Cette catégorie de Clustering ne définit pas un type spécifique de Clustering, ces contraintes peuvent être appliquées aux différents algorithmes de Clustering existants. Les contraintes établies servent comme un guide à l'algorithme de Clustering utilisé pour construire l'ensemble des Clusters. Pour cette raison, ce type de Clustering est appelé aussi classification semi-supervisé. Les algorithmes de Clustering avec contraintes visent à trouver un partitionnement des données en respectant les contraintes existantes, sinon, ils proposent des solutions approximatives en minimisant le nombre de contraintes non satisfaites.

4 Mesures pour le clustering

Dans ce chapitre, nous avons présenté plusieurs algorithmes de Clustering, dont la majorité restent sensibles aux paramètres utilisés, ce qui influence sur le résultat du Clustering. Pour cette raison, il est important d'évaluer les résultats obtenus afin de juger l'efficacité de l'algorithme utilisé.

Comme on ignore généralement la structure des données analysées, décider si un résultat de Clustering est bon ou non n'est pas une tâche évidente. Pour pallier à ce problème, plusieurs approches d'évaluation et de validation des résultats de Clustering ont été proposées [91].

Les méthodes d'évaluations sont utilisées pour : valider les résultats d'un algorithme de Clustering, comparer des algorithmes de Clustering différents ou comparer les résultats d'un même algorithme avec des paramètres d'entrée différents. Les techniques d'évaluations peuvent alors être utilisées pour déterminer le nombre adéquat de Clusters en observant le changement dans la qualité du Clustering en variant le nombre de Clusters.

Ces techniques apportent une évaluation statistique aux algorithmes utilisés, mais en l'absence d'une classification prédéfinie des données analysées, ces techniques restent approximatives.

Les techniques de validation peuvent être classifiées en deux catégories : validation interne et validation externe. Dans cette section nous présenterons les différentes techniques pour l'évaluation des résultats de Clustering, ensuite nous présenterons les mesures de calcul de similarité entre les données. Ces dernières sont utilisées par les différentes techniques de validation des résultats de Clustering.

4.1 Mesures de validation des résultats de Clustering

4.1.1 Validation interne

Les algorithmes de Clustering visent à générer des Clusters en minimisant la distance entre les individus d'un même Cluster et maximiser la distance entre les individus des Clusters différents. Ainsi les techniques de validation interne consistent à évaluer un résultat de Clustering en se basant sur ces valeurs de distances inter et intra Clusters. Ces techniques sont dites internes car seulement les informations contenues dans le résultat du Clustering sont utilisées et aucune information externe concernant les données analysées n'est utilisée [61].

Plusieurs techniques dans cette catégorie ont été proposées.

4.1.1.1 La mesure SSW La mesure SSW ou (Sum-of-Squares Within Clusters) est la plus connue des mesures d'évaluation. Elle calcule la distance moyenne entre les objets de chaque Cluster par rapport aux centroïdes des Clusters auxquels ils appartiennent. Elle se base sur la distance entre un objet est le centre de son Cluster seulement (distance intra Cluster). SSW est calculée avec la formule (1.12) suivante :

$$SSW = \frac{\sum_{i=1}^{k} \left\{ \frac{\sum_{j=1}^{n_i} d\left(o_i, p_{ij}\right)}{n_i} \right\}}{K} \tag{1.12}$$

Où: p_{ij} est le j^{eme} point de données dans le i^{me} Cluster; o_i est le centroïde du i^{eme} Cluster; $d(o_i,p_{ij})$ est la distance entre le point de données p_{ij} et le centroïde o_i; n_i est le nombre d'individus dans le Cluster C_i et K est le nombre de Clusters.

4.1.1.2 La mesure SSB Contrairement à SSW, la mesure SSB ou (sum-of-squares between Clusters), est la distance moyenne entre chaque centroïde et la moyenne des autres centroïdes (distance inter Clusters). SSB est calculé par la formule (1.13) suivante :

$$SSB = \sum_{i=1}^{i=K} |C_i| \left(\text{dist}\left(c_i,c\right)\right)^2 \tag{1.13}$$

Où : $|C_i|$ est le nombre d'individus dans le Cluster c_i, et c : est la moyenne des centroïdes de la configuration des Clusters à l'exception de c_i.

4.1.1.3 La mesure DBI la métrique DBI ou (Davies-Bouldin index) représente un ratio entre la distance intra Clusters et la distance inter Clusters [28]. La mesure DBI est calculée avec la formule (1.14) suivante :

$$DBI = \frac{1}{K} \sum_{i=1}^{K} R_i \tag{1.14}$$

Où R_i est égale au ratio de distance maximale entre le Cluster c_i et les $k-1$ autres Clusters. La valeur de R_{ij} est calculée avec la formule (1.15) ci-dessous :

$$R_{ij} = \frac{S_i - S_j}{d_{ij}} \tag{1.15}$$

Où : S_i (et S_j) représente la distance moyenne entre les individus du Cluster c_i (et c_j) et leur centroïde.

4.1.2 Validation externe

Les méthodes de validation externes sont dites externes car elles utilisent des informations concernant les données analysées qui sont externes aux informations obtenues grâce au processus de Clustering, et qui étaient disponibles au préalable [32]. Les méthodes de validation externe sont préférables par rapport aux méthodes internes quand des classes (labels) réelles prédéfinies sont disponibles. Les classes prédéfinies consistent en des labels assignés à chaque individu de l'ensemble de données pour définir sa classe.

Ainsi, la validation des résultats consiste à vérifier la correspondance entre la classification générée par l'algorithme de Clustering et la classification prédéfinie sur les données réelles.

Parmi ces méthodes on peut citer : le Rappel, la Précision et F-mesure...etc. Ces techniques sont utilisées généralement pour l'évaluation des algorithmes de classification supervisée où les informations sur les données analysées sont disponibles mais aussi pour les algorithmes de Clustering dans le cas où les données analysées contiennent ces informations.

Dans ce qui suit nous présentons des exemples pour les techniques de validation externe.

4.1.2.1 Le Rappel Dans un système de recherche d'information, le rappel désigne le nombre de résultats pertinents trouvés par rapport à tous les résultats pertinents possibles [83]. Dans un problème de Clustering de données, le rappel correspond au nombre d'individus assignés correctement à un Cluster par rapport au nombre d'individus appartenant réellement à ce Cluster. Le rappel est calculé avec la formule (1.16) :

$$r(i,j) = \frac{n_{ij}}{n_i} \tag{1.16}$$

Où : n_{ij} est le nombre des individus de la classe i assignés au Cluster j et n_i est le nombre d'individus de la classe i.

4.1.2.2 La Précision La précision désigne le nombre de résultats pertinents trouvés par rapport à tous les résultats trouvés. Plus cette valeur est élevée, plus le système est dit précis [83]. Dans un problème de Clustering de données, la précision correspond au nombre d'individus assignés correctement à un Cluster par rapport au nombre de tous les individus contenus dans ce Cluster. La précision est calculée avec la formule (1.17):

$$p(i,j) = \frac{n_{ij}}{n_j} \tag{1.17}$$

Où : n_{ij} est le nombre des individus de la classe i assignés au Cluster j et n_j est le nombre d'individus du Cluster j.

4.1.2.3 F-mesure La métrique F-mesure, spécialement utilisée pour l'évaluation des systèmes de recherche d'information, vise à évaluer la précision d'un système. F-mesure utilise deux mesures différentes : le rappel et la précision [45]. La valeur totale de F-mesure est calculée avec la formule (1.18) :

$$F = \sum_i \frac{n_i}{n} \max_j (F(i,j)) \tag{1.18}$$

Où $F(i,j)$ représente la valeur de F-mesure de la classe i avec le Cluster j. Elle est calculée par la formule (1.19) suivante :

$$F(i,j) = \frac{2 \cdot r(i,j) \cdot p(i,j)}{r(i,j) + p(i,j)} \tag{1.19}$$

La formule F a une valeur maximale de 1 qui est atteinte dans le cas où, pour la classe i, tous ses éléments sont assignés au Cluster j, et le Cluster j ne contient que les éléments de la classe i. Ainsi, plus la valeur de la formule F est proche de 1, plus le résultat du Clustering est considéré de bonne qualité.

4.2 Les mesures de similarités des données (distance)

Dans la section précédente, nous avons présenté les méthodes d'évaluation des algorithmes de Clustering. Plusieurs métriques ont été proposées. Toutes ces métriques à l'exception des méthodes d'évaluation externes, reposent sur le concept de distance entre les points de données. La distance entre deux individus représente le niveau de similarité entre eux. Une large distance représente une dissimilarité entre ces individus, et une distance réduite représente une forte similarité entre eux [21].

Il existe plusieurs méthodes pour calculer la distance entre deux points de données. Chaque méthode est utilisée selon la nature des données, l'algorithme utilisé et le type de résultats attendus. Dans ce qui suit, nous présenterons les méthodes de calcul de distance communément utilisées par les algorithmes de Clustering.

4.2.1 Distance euclidienne

La distance euclidienne est la métrique standard pour le calcul des distances. Elle est la plus communément utilisée par les algorithmes de Clustering. La distance euclidienne calcule la racine carrée de la somme des différences au carré entre les coordonnées d'une paire d'objets [26]. La formule (1.20) est utilisée pour calculer la distance euclidienne entre deux points de données.

$$Dist_{XY} = \sqrt{\sum_{k=1}^{m} \left(X_{ik} - X_{jk}\right)^2} \tag{1.20}$$

4.2.2 Distance de Manhattan

La distance de Manhattan calcule la somme des valeurs absolues des différences entre les coordonnées d'une paire d'objets [79]. La formule (1.21) est utilisée pour calculer la distance de Manhattan.

$$Dist_{XY} = \sum_{k=1}^{d} \left| X_{ik} - X_{jk} \right| \tag{1.21}$$

4.2.3 Distance de Tchebychev

La distance de Tchebychev entre deux points de données représente la différence absolue maximale entre ces deux points sur une seule dimension [76]. Cette distance est calculée par la formule (1.22) suivante :

$$\text{Dist}_{XY} = \max_{k} \left| X_{ik} - X_{jk} \right| \tag{1.22}$$

4.2.4 Distance de Minkowski

La distance de Minkowski est une mesure de distance généralisée [65], elle est calculée par la formule (1.23) suivante :

$$Dist_{XY} = \left(\sum_{k=1}^{d} \left| X_{ik} - X_{jk} \right|^{\frac{1}{p}} \right)^{p} \tag{1.23}$$

Quand la valeur de p est égale à 2, la formule représentera la distance euclidienne ; et si p est égal à 1, la formule représentera la distance de Manhattan.

5 Conclusion

Dans ce chapitre nous avons présenté plusieurs algorithmes de Clustering appartenant à différentes catégories. Certains de ces algorithmes proposent des optimisations pour des algorithmes déjà existants, visant à améliorer leur qualité de Clustering ou leur complexité temporelle. Ces solutions ont montré leur efficacité mais avec l'élargissement des masses de données, ces optimisations deviennent insuffisantes. De plus, certains de ces algorithmes tendent à utiliser les techniques d'échantillonnage pour réduire la taille des données analysées dans le but d'optimiser le temps d'exécution, mais ceci affectera la précision des résultats du Clustering. Pour ces raisons l'utilisation du parallélisme devient une solution incontournable. Dans le chapitre suivant nous introduirons le Clustering parallèle de données.

Chapitre 2

Clustering parallèle

1 Introduction

Avec l'augmentation significative dans le volume des données analysées, le processus de Clustering devient une tâche coûteuse. Comme mentionné dans le chapitre précédent, plusieurs travaux ont tenté de réduire la complexité temporelle des algorithmes de Clustering en utilisant différentes techniques, de l'échantillonnage des données à l'optimisation des schémas des algorithmes. Avec l'explosion dans la taille des données dans les différents domaines, ces solutions deviennent insuffisantes. Par conséquent, plusieurs travaux ont été proposés pour surmonter ce problème en utilisant le parallélisme.

Dans un contexte général, la parallélisation des algorithmes vise à exploiter les processeurs multi-cœurs ou des Clusters de machines pour optimiser le temps d'exécution d'une tâche donnée [78].

Dans le domaine du Clustering de données, les algorithmes parallèles peuvent être classifiés en plusieurs catégories selon plusieurs critères.

La première des méthodes possible, est de classifier ces algorithmes selon la plateforme ou le système utilisé pour l'implémentation du parallélisme. Ainsi, la parallélisation des algorithmes peut être classifiée en deux catégories:

1. Parallélisme avec mémoire partagée qui inclue l'utilisation des CPU multi-cœurs ou les processeurs graphiques (GPU).
2. Parallélisme avec mémoire distribuée qui consiste à l'utilisation des réseaux de machines.

Dans un contexte plus général, un programme parallèle utilisant des CPU ou GPU peut utiliser une mémoire distribuée locale, où à chaque processus est alloué une mémoire privée. De la même manière, un Cluster de machines peut utiliser une mémoire partagée qui, dans

ce cas, est appelée mémoire partagée distribuée. La mémoire partagée distribuée est une forme de mémoire où des mémoires séparées sur des différentes machines sont adressées comme une seule mémoire logique. Dans le contexte de cette thèse la mémoire partagée est utilisée pour désigner les approches implémentant un parallélisme local qui utilise un CPU multi-coeur ou un GPU, et la mémoire distribuée désigne seulement les approches implémentant un parallélisme distribué utilisant des Clusters de machines.

Ces deux types de parallélisme peuvent utiliser soit une architecture peer-to-peer ou une architecture master-slave, selon les besoins de la solution implémentée. Dans l'architecture peer-to-peer, les machines ou processus communiquent directement entre eux pour achever leur objectif [77]. Avec l'architecture Master/Slave, une machine (ou un processus) dit Master, se charge de diriger et d'assigner des tâches aux autres machines(ou processus) appelées Slaves (ou workers) [33].

Outre cette classification, dans [82], l'auteur a présenté une classification des algorithmes parallèles dans le domaine du Data Mining selon les schémas suivis pour distribuer les tâches et les données afin d'implémenter le parallélisme. L'auteur a proposé trois classifications possibles:

1. Parallélisme indépendant.
2. Parallélisme par tâches.
3. Parallélisme SPMD (Single Program, Multiple Data).

Ces trois architectures peuvent être implémentées en utilisant un parallélisme avec une mémoire partagée ou distribuée.

Dans ce qui suit, ces deux différentes classifications sont présentées.

2 Classification du parallélisme selon la mémoire utilisée

2.1 Parallélisme avec mémoire partagée

Le parallélisme avec mémoire partagée consiste à implémenter du parallélisme en utilisant un processeur multi-cœurs ou un processeur graphique. Dans ce cas, les tâches indépendantes de l'algorithme implémenté sont exécutées simultanément par les différents cœurs physiques du CPU ou du GPU (processus ou threads). Les données analysées sont présentes entièrement sur la mémoire locale de la machine utilisée. Tous les processus impliqués partagent la même mémoire avec des accès concurrents à cette dernière d'où l'appellation de mémoire partagée [50].

2.1.1 Avantages

Les avantages de la mémoire partagée sont liés à la disponibilité entière des données traitées sur la même machine. Ce qui garantira :

- Rapidité d'accès aux données en évitant les communications réseaux comme le cas des mémoires distribuées.
- L'accès entier à tout l'ensemble de données, ce qui favorisera l'obtention de résultats précis pour le cas des algorithmes de Clustering.

2.1.2 Inconvénients

Parmi les inconvénients de la mémoire partagée on peut citer:

- L'accès simultané aux données par plusieurs processeurs (processus/thread) peut engendrer des cas de blocage (deadlock) pour certains processus.
- Le nombre de cœurs physiques dans les processeurs multi-cœurs communément utilisés n'est pas très important ce qui limitera les performances dans le cas où la taille des données traitées est très importante.
- Dans les cas de traitement des données massives, la mémoire locale est insuffisante pour les contenir.

2.2 Parallélisme avec mémoire distribuée

Avec l'explosion de la taille des données à analyser, le parallélisme en utilisant la mémoire partagée devient inefficace due au nombre limité de cœurs au sein d'un seul processeur. Ainsi, le parallélisme basé sur les Clusters de machines devient une solution nécessaire. Ce parallélisme consiste à utiliser un ensemble de machines connectées pour effectuer une seule opération principale. Chaque machine utilise ses propres ressources pour participer au traitement d'une tâche donnée. Ainsi, chaque machine peut accéder seulement aux données présentes localement sur sa propre mémoire d'où l'appellation de mémoire distribuée [72].

2.2.1 Avantages

L'avantage de l'utilisation d'un Cluster de machines peut se résumer dans les points suivants :

- La distribution des données et de la mémoire évitera les problèmes des accès concurrents.

– L'utilisation d'un réseau de plusieurs machines garantira des ressources de calcules importantes.
– L'utilisation de plusieurs machines, chacune avec son propre espace de stockage, permettra le traitement d'une quantité de données très importante.

2.2.2 Inconvénients

Hormis les avantages de la mémoire distribuée, cette dernière présente certains inconvénients:

– La distribution des données sur les machines d'un réseau, force l'utilisateur à adopter une méthode pour la distribution des données qui peut ne pas être adéquate pour les problèmes traités.
– La communication et le transfert des données entre les nœuds du réseau ralentiront le temps d'exécution.
– L'implémentation d'une solution parallèle sur un réseau de machine requière des compétences avancées en programmation parallèle ce qui a encouragé plusieurs solutions récentes à utiliser le Framework MapReduce.

Le Framework MapReduce sera présenté en détail dans la section 4 du prochain chapitre.

3 Classification selon la distribution des données et des tâches

3.1 Parallélisme indépendant

Avec le parallélisme indépendant, chaque processus (sur un cœur de CPU ou sur une machine) est exécuté indépendamment des autres processus. Les processus ne sont pas synchronisés et ne communiquent pas entre eux. Chaque processus a accès à tout l'ensemble des données et se chargera d'exécuter tout l'algorithme en question. Dans le cas des algorithmes de Clustering, ce type de parallélisme consiste à exécuter plusieurs instances de l'algorithme de Clustering en utilisant tout l'ensemble de données, mais avec des paramètres différents, comme le nombre de Clusters k. L'exécution avec le meilleur résultat sera retenue comme solution pour l'algorithme.

3.2 Parallélisme par tâches

Selon ce type de parallélisme, chaque processus, exécute une tâche différente des autres processus en utilisant une partition ou l'ensemble complet des données. Pour appliquer cette stratégie, les tâches indépendantes de l'algorithme utilisé doivent être identifiées et

exécutées simultanément (sur un cœur de CPU ou sur une machine). Dans le cas des algorithmes de Clustering, chaque processus exécute une opération indépendante appartenant à l'algorithme utilisé. Les processus communiquent entre eux pour échanger les résultats partiels. Par exemple, pour l'algorithme K-means, les opérations de calcul des distances minimales peuvent être exécutées simultanément ainsi que les opérations des mises à jour des centroïdes. Avec cette stratégie, le déroulement de l'algorithme original n'est pas modifié et tout l'ensemble de données est traité comme une seule entité. Ainsi cette stratégie garantira des résultats identiques à ceux d'une exécution séquentielle, ce qui peut être considéré comme un avantage. Par contre, l'inconvénient de cette approche est que ce n'est pas tous les algorithmes qui peuvent être répartis en des sous-tâches indépendantes, ainsi, cette dernière ne peut pas être appliquée.

3.3 Parallélisme « SPMD » (Single Program, Multiple Data)

Avec cette méthode, l'algorithme utilisé est exécuté simultanément par plusieurs processus (ou machine) en utilisant des partitions de données différentes. Chaque processus génère des résultats de Clustering partiels. Ces résultats seront combinés pour générer un résultat global. L'avantage de la méthode SPMD est qu'aucune modification n'est requise sur le schéma principal de l'algorithme utilisé, ce dernier est seulement exécuté simultanément avec des données différentes.

Par contre, cette stratégie exige que les données soient distribuées sur les processus, et ensuite, les résultats partiels doivent être fusionnés. Ce point peut être considéré comme un inconvénient étant donné que les méthodes utilisées pour la distribution des données ou la fusion des résultats peuvent ne pas être adéquates. Plusieurs méthodes peuvent être utilisées pour ces deux étapes que nous détaillerons dans les prochains chapitres.

Les trois stratégies de parallélisme citées dans cette section peuvent être combinées dans le but d'optimiser le temps d'exécution et la précision des résultats du Clustering.

4 Evaluations des algorithmes de Clustering parallèles

Comme pour les algorithmes de Clustering séquentiels, l'objectif d'un algorithme de Clustering parallèle est de découvrir un modèle de distribution de données de sorte à maximiser la similarité entre les données d'un même groupe, ce qui est vérifiable avec les métriques citées dans le premier chapitre. Outre l'évaluation de la qualité de Clustering, une solution parallèle doit aussi satisfaire plusieurs autres métriques qui déterminent le rapport de performance apporté par le schéma parallèle utilisé [39].

Une solution parallèle vise à améliorer la complexité temporelle d'un algorithme séquentiel. Cette amélioration peut ne pas être optimale et n'exploitera pas les ressources allouées à cause des techniques utilisées pour l'implémentation du parallélisme.

Ainsi, pour vérifier l'efficacité d'un algorithme parallèle, plusieurs métriques sont utilisées : Speed Up, Efficiency (Efficacité), Size Up et Scale Up.

4.1 Speed Up

Le Speed Up ou l'accélération est une mesure qui détermine le facteur de rapidité d'un algorithme parallèle en utilisant x machines (ou cœurs de processeur) par rapport à sa version séquentielle, ou d'un algorithme parallèle utilisant x machines par rapport à son exécution en utilisant y machines ; où $x > y$ [4].

Le Speed Up est calculé avec la formule (2.1) suivante :

$$SpeedUp = \frac{T_y}{T_x} \tag{2.1}$$

Où : T_y est le temps d'exécution de l'algorithme séquentiel ou de l'algorithme parallèle en utilisant y machine ; T_x est le temps d'exécution de l'algorithme parallèle en utilisant x machines; avec $x > y$.

Ainsi, le Speed Up d'un algorithme parallèle donné est analysé en fixant tous les paramètres et les données utilisées tout en variant le nombre de machines (ou processeurs) utilisées.

4.2 Efficacité

L'Efficacité ou Efficiency d'un algorithme parallèle est une mesure qui détermine le taux d'exploitation des machines utilisées par un algorithme parallèle [42]. L'Efficacité est le ratio du Speed Up par rapport au nombre de machines utilisées. L'Efficacité est calculée par la formule (2.2) suivante :

$$\text{Efficacite } = \frac{\text{SpeedUp}}{\text{Nb machines}} \tag{2.2}$$

L'Efficacité a une valeur théorique maximale de 1. Plus sa valeur est proche du 1 plus l'algorithme est considéré efficace.

4.3 Size Up

La mesure Size Up évalue la performance d'un algorithme parallèle à gérer la croissance dans la taille des données analysées. Pour évaluer l'algorithme, le nombre de machines et les autres paramètres sont fixés tout en dupliquant la taille des données utilisées [89]. Les données sont dupliquées par un facteur fixe m. Le Size Up est calculé avec la formule (2.3) suivante :

$$\text{SizeUp} = \frac{T_y}{T_x} \tag{2.3}$$

Où : T_y est le temps d'exécution de l'algorithme parallèle avec des données de la taille de y ; T_x est le temps d'exécution de l'algorithme parallèle avec des données de la taille de x. Avec $y = m * x$ et $m > 1$.

Le Size Up est analysé en observant les variations dans la taille des données et leur impact sur le temps d'exécution de l'algorithme parallèle.

4.4 Scale Up

La mesure Scale Up mesure la capacité d'un algorithme parallèle à traiter des données x fois plus importantes que les données de base, en utilisant x machines dans le même temps d'exécution nécessaire pour sa version séquentielle [5]. La formule (2.4) suivante est utilisée pour calculer le Scale Up d'un algorithme parallèle :

$$\text{Scale U } p = \frac{T}{T_x} \tag{2.4}$$

Où : T est le temps d'exécution de la version séquentielle de l'algorithme parallèle utilisé pour classifier un ensemble de données D ; T_x est le temps d'exécution de l'algorithme parallèle analysé en utilisant x machines pour classifier un ensemble de données de la taille de $D * x$.

5 Algorithmes parallèles pour le Clustering de données

Dans cette section nous présentons plusieurs algorithmes de Clustering parallèles suivant les différentes catégories et architectures présentées dans cette section. Les algorithmes présentés seront classifiés selon le type de parallélisme utilisé :

– Parallélisme avec mémoire partagé.

– Parallélisme avec mémoire distribuée.

Nous précisons aussi pour les algorithmes présentés, leur catégorie selon la classification présentée par [82].

5.1 Algorithmes parallèle avec mémoire partagée

Plusieurs travaux utilisant différentes approches ont été proposés pour paralléliser les algorithmes de Clustering en utilisant la mémoire partagée.

Dans [53], une parallélisation pour l'algorithme K-means utilisant la mémoire partagée a été proposée. La solution est basée sur l'architecture Master/Slave et le concept de message entre processus. La solution consiste à générer un ensemble de centroïdes à partir des données globales qui seront envoyées aux processus Slaves par le processus Master. Ce dernier distribue les données sur tous les processus Slaves. Chaque processus Slave assigne ses points de données aux Clusters les plus proches. Les résultats de distribution sont renvoyés au processus Master qui procède à la mise à jour des centroïdes. Ces étapes sont répétées jusqu'à convergence. Cet algorithme utilise l'architecture « parallélisme par tâches ».

Dans [20], les auteurs ont utilisé les systèmes multi-agents pour paralléliser est optimiser le processus de Clustering. La solution utilise plusieurs catégories d'agents. Chaque catégorie d'agents est responsable d'une tâche particulière. Ainsi, les types d'agents sont :

- Agents des données: où chaque agent est responsable d'une partition de données.
- Agents de Clustering: où chaque agent est responsable d'un Cluster donné.
- Agents de validation: où chaque agent mesure la pertinence d'un Cluster donné.

Les différents types d'agents communiquent entre eux dans le but d'atteindre une solution de Clusters optimale. L'approche basée sur les agents a été appliquée sur l'algorithme K-means. Comme chaque agent est responsable d'une tâche particulière, cet algorithme suit ainsi l'architecture « parallélisme par tâches ».

Une solution parallèle pour l'algorithme de Clustering hiérarchique agglomérative a été proposée dans [70]. La solution est basée sur l'architecture « parallélisme par tâches », ainsi, chaque Cluster est géré par un processus indépendant. Une fois deux Clusters sont fusionnés, un des processus prendra en charge le nouveau Cluster et l'autre processus sera arrêté.

Dans [22], une solution parallèle en utilisant un processeur graphique (GPU) a été proposée. Une des solutions possible pour le problème de Clustering de données consiste à imiter le comportement de flocage des essaims d'oiseaux [23]. Cette solution a une complexité temporelle de $O(n^2)$, ce qui nécessitera un temps d'exécution très important dans le cas où les données analysées ont une taille importante. Pour cette raison, les auteurs ont proposé une solution parallèle en utilisant un processeur graphique. Les résultats ont montré que l'utilisant d'un GPU comparé à l'utilisation d'un CPU a apporté un temps d'exécution 30 fois plus rapide. Ceci est du au nombre important de cœurs de calcul d'un GPU par rapport à un CPU.

De la même manière que dans [22], une solution basée sur l'utilisation des GPU a été proposée dans [60] pour paralléliser la technique de Clustering par recherche d'affinités

[10]. Les résultats ont monté que par l'utilisation d'un GPU de basse performance, le temps d'exécution et 21% plus rapide que l'algorithme séquentiel.

Dans [88], un algorithme parallèle pour le Clustering de documents a été proposé. La solution consiste à utiliser un algorithme hybride qui utilise une implémentation parallèle de l'algorithme K-means avec une implémentation parallèle de l'algorithme « Principal Direction Divisive Partitioning» (PDDP) [15]. L'algorithme PDDP a été utilisé pour l'initialisation des centroïdes de départ pour l'algorithme K-means. Le parallélisme est réalisé en utilisant un CPU multi-cœurs en suivant l'architecture « parallélisme par tâches».

Une autre approche basée sur les CPU multi-cœurs a été proposée dans [54] pour paralléliser l'algorithme K-means. Les auteurs ont opté pour l'utilisation de la mémoire partagée pour éviter les communications réseaux. L'algorithme adopte une architecture Master/slave. L'utilisation de la mémoire partagée peut causer dans certains cas des problèmes de blocage (deadlocks). Les auteurs ont proposé l'algorithme McK-means (Multi-cores K-means), qui, en plus d'optimiser le temps d'exécution, il évite les problèmes de blocages entre les processus. Avec l'algorithme McK-means, l'algorithme K-means est parallélisé en traitant simultanément le calcul des distances minimales et la mise à jour des centroïdes. Cette approche commence par partitionner l'ensemble de données en plusieurs partitions. Chaque partition est gérée par un thread pour calculer la distance minimale entre chaque point de données de cette partition et les K centroïdes. Chaque centroïde est mis à jour par un thread indépendant à la fin de chaque itération. Ainsi, les opérations de calcul de distances minimales est la mise à jour de centroïdes sont exécutées simultanément. Cette solution utilise l'architecture «parallélisme par tâches ».

5.2 Algorithmes parallèles avec mémoire distribuée

Dans cette section nous présentons plusieurs approches parallèles pour les algorithmes de Clustering en utilisant des Clusters de machines. Les différentes solutions utilisent: (1)soit un Framework de parallélisation tel MapReduce, comme sur les travaux présentés dans [38] [80], ce qui leur permettra d'éviter les problèmes de gestion des communications et la synchronisation entre les nœuds du réseau; (2) soit ils proposent leurs propres modèles et schémas pour la gestion des communications entre les machines du Cluster, comme sur les travaux [8] [81].

Les solutions présentées dans cette section seront plus basées sur l'utilisation du Framework MapReduce.

Dans [38], les auteurs ont proposé une solution parallèle pour le Clustering de données en utilisant MapReduce. La solution consiste à utiliser un Cluster de machines gérées par le Framework MapReduce pour optimiser le temps d'exécution du processus de Clustering.

Dans cette solution, les données sont distribuées aléatoirement sur les différentes machines. Chaque machine applique un algorithme de Clustering séquentiel sur son ensemble de données. Les configurations de Clusters générés sur les différentes partitions (Clusters intermédiaires) seront fusionnées pour obtenir le résultat final.

Le schéma de cet algorithme se résume en ces trois étapes :

- Distribuer les données aléatoirement.
- Appliquer un Clustering local.
- Fusionner les résultats qui se chevauchent dans l'espace de données.

Les auteurs ont mentionné que différents algorithmes de Clustering séquentiels peuvent être utilisés pour le Clustering local au niveau de chaque machine. A partir de la description de cette solution, on peut remarquer que cet algorithme suit l'architecture SPMD.

De la même manière que la solution précédente, dans [80], une solution pour paralléliser l'algorithme K-means basée sur MapReduce a été proposée. Dans cette solution les données sont divisées en des sous-ensembles où chacun est classifié indépendamment.

Dans [35], un algorithme basé sur l'architecture SPMD en utilisant le Framework MapReduce a été présenté. La solution proposée a été testée pour paralléliser l'algorithme K-means et l'algorithme k-médoïdes. Similairement à la solution proposée dans [38], les données sont distribuées aléatoirement sur les nœuds du réseau, ensuite, sur chacun des nœuds, un Clustering local est appliqué. Dans ce cas, l'algorithme K-means et l'algorithme k-médoïdes sont utilisés pour le Clustering local. Dans le cas où les données sont classifiées par K-means, ce dernier est utilisé pour classifier les centroïdes intermédiaire afin de définir les centroïdes finaux. De la même manière, K-médoïdes est utilisé dans cette étape dans le cas où ce dernier est utilisé pour la classification des partitions de données.

Dans [8], un algorithme K-means parallèle basé sur une architecture peer-to-peer a été présenté. La solution consiste à distribuer les données sur l'ensemble des machines, ensuite, chaque itération de l'algorithme K-means est exécutée par toutes les machines du réseau en utilisant les mêmes centroïdes de départ. Chaque machine met à jour ces centroïdes selon les données qu'elle détient. Ces centroïdes seront encore mis à jour en calculant leurs moyennes avec les centroïdes voisins des autres machines. Ces étapes sont répétées jusqu'à ce qu'un état stable soit atteint. Cette solution utilise l'architecture SPMD.

Une autre solution pour la parallélisation de K-means avec le Framework MapReduce a été proposée dans [92]. La solution adapte l'algorithme K-means au paradigme de MapReduce. Dans cette solution, l'ensemble des partitions de données distribuées sur les Mappers ne sont pas classifiées indépendamment comme dans [38] [35]. Les auteurs ont proposé de garder le même schéma principal de l'algorithme K-means. Ainsi, l'algorithme commence par choisir

K centroïdes aléatoires qui seront distribués sur les Mappers. Ces derniers se chargent de calculer les distances minimales entre chaque point de données et les K centroïdes. Les Mappers assignent chaque point de données au Cluster le plus proche en générant un résultat de type paire $(Cl, Valeur)$, où « Valeur » correspond à un point de données ; et « Clé » correspond au Cluster auquel le point de données contenu dans « Valeur » appartient. En suivant ce modèle de données généré par les Mappers, chaque Reducer recevra tous les points de données appartenant à un même Cluster (Clé). Les Reducers procèderont à la mise à jour des centroïdes en utilisant les points de données reçues (Valeurs). Les nouveaux centroïdes seront communiqués à tous les Mappers pour procéder à une nouvelle itération. Ce processus est répété jusqu'à convergence ou un nombre maximal d'itérations est atteint. Cet algorithme utilise l'architecture parallélisme par tâches, où chaque Mapper ou Reducer exécute une des tâches de l'algorithme principale. On note que pour chaque itération une opération Map/Reduce est initialisée (MapReduce Job), ce qui ralentira sur le temps d'exécution global.

Dans [93], le même algorithme proposé dans [92] a été adapté et utilisé pour la classification de documents.

De la même manière, dans [62], l'algorithme présenté dans [92] a été implémenté pour la classification des images.

Dans [81], les auteurs ont proposé une solution parallèle utilisant un réseau de machines avec une architecture peer-to-peer pour paralléliser l'algorithme K-means. La solution est similaire à celle présentée dans [92] mais sans l'utilisation du Framework MapReduce.

Dans [25], les auteurs ont proposé une parallélisation basée sur MapReduce pour l'algorithme K-means. Cette solution permet d'éviter le traitement itératif de l'algorithme K-means. Dans cette solution les données sont échantillonnées dans un premier processus (Job) MapReduce afin de réduire leur taille. Dans un deuxième processus MapReduce les données sont classifiées en utilisant K-means. Enfin les données sont fusionnées pour avoir un résultat global.

Dans [85], les auteurs ont proposé une solution parallèle pour l'algorithme hybride PSO-K-means [2]. L'algorithme PSO est exécuté séquentiellement pour le raffinement des centroïdes de départ. Les résultats obtenus sont utilisés pour exécuter l'algorithme K-means qui est parallélisé en utilisant le Framework MapReduce suivant l'architecture proposée dans [92]. Dans cette solution l'algorithme PSO n'est pas parallélisé, ce qui n'optimisera pas le temps d'exécution de la phase de raffinement des centroïdes.

Une autre solution utilisant les méta-heuristiques a été proposée dans [9] pour classifier les données. Dans ce travail l'algorithme colonie d'abeilles a été parallélisé en utilisant MapReduce suivant l'architecture parallélisme par tâches. Le parallélisme a été appliqué

pour la phase de calcul des valeurs fitness des solutions de l'algorithme qui requerra le plus de temps d'exécution.

6 Conclusion

Dans ce chapitre nous avons présenté le concept du Clustering parallèle. Plusieurs architectures de parallélisme dans le domaine du Data Clustering ont été présentées. Chacune de ces architectures est choisie selon l'infrastructure utilisée, selon la structure et l'objectif de l'algorithme implémenté.

Nous avons présenté plusieurs travaux utilisant le Framework MapReduce suivant différentes architectures. Nous avons constaté que les travaux récemment proposés utilisent le Framework MapReduce car il présente certains avantages comme la facilité de l'implémentation d'une solution parallèle. Ceci nous a encouragé à l'utiliser dans la solution qui sera présentée dans le prochain chapitre.

En effet l'architecture utilisée a un impact sur les résultats de Clustering en termes de précision et de temps d'exécution. Les algorithmes présentés adoptaient deux architectures différentes : parallélisme par tâches ou parallélisme SPMD.

Sachant que MapReduce n'est pas adapté aux processus itératifs, l'utilisation de l'architecture parallélisme par tâches avec des algorithmes qui nécessitent plusieurs itérations engendrera un temps d'exécution plus important que lors de l'utilisation de l'architecture SPMD, qui, selon les travaux présentés nécessitera un nombre limité d'instances MapReduce.

D'un autre côté, les algorithmes de Clustering utilisant MapReduce avec l'architecture parallélisme par tâches suivent le même schéma et déroulement que leurs versions séquentielles, ce qui garantira des résultats avec une précision similaire à celle générée par ces derniers. Cependant, avec l'architecture SPMD, la précision des résultats variera des versions séquentielles selon les techniques utilisées pour le partitionnement des données et la fusions des résultats.

D'après les travaux présentés dans ce chapitre, on a remarqué que les solutions parallèles proposées visent à réduire seulement le temps d'exécution par rapport à leurs versions séquentielles en exploitant les différentes techniques de parallélisme, tout en maintenant une qualité de Clustering proche ou identique à celle assurée avec une exécution séquentielle.

Chapitre 3

Parallélisation locale dans un environnement distribué (MapReduce)

1 Introduction

Dans le chapitre précédant, nous avons présenté les techniques de Clustering parallèles de données. Plusieurs catégories de Clustering parallèle ont été discutées. Chacune avec ses avantages et ses inconvénients. Parmi ces catégories, nous avons présenté les techniques de parallélisme avec mémoire partagée et les techniques de parallélisme avec mémoire distribuée.

Dans ce chapitre, nous présenterons notre première contribution qui consiste en une solution hybride pour la parallélisation d'un algorithme de Clustering basé sur la mémoire distribuée et la mémoire partagée. Cette solution a été présentée dans [16].

La solution consiste à paralléliser l'algorithme Sampling-PSO-K-means [49], qui est un algorithme hybride visant à améliorer les résultats de l'algorithme K-means en utilisant l'échantillonnage de données et la méta-heuristique Particule Swarm Optimisation (PSO). Cet algorithme est utilisé par la suite dans les chapitres suivants comme algorithme principal avec lequel les différentes approches présentées dans cette thèse sont illustrées et validées.

Dans la solution présentée dans ce chapitre, l'algorithme Sampling-PSO-K-means est parallélisé en utilisant la mémoire distribuée qui est implémentée par le Framework MapReduce. De plus, sur chacun des nœuds appartenant au réseau MapReduce, un parallélisme local en utilisant la mémoire partagée est appliqué afin d'exploiter les ressources de calcul disponibles sur chacun de ces nœuds, ce qui garantira une optimisation du temps d'exécution.

Ce chapitre sera présenté comme suit: Dans la section 2 nous présenterons l'algorithme Sampling-PSO-Kmeans. Nous introduirons le Framework MapReduce dans la section 3.

Dans la section 4 nous présenterons les détails de notre approche proposée. La section 5 présente la partie expérimentation et évaluation des résultats obtenus. La section 6 conclut le chapitre.

2 L'optimisation par essaims particulaires (PSO)

PSO est une métaheuristique basée sur le comportement social au sein d'un organisme tel que un essaim d'oiseaux ou de poissons (particules) [34].

L'algorithme PSO est basé sur une population de particules appelée 'essaim' se déplaçant dans un espace de recherche afin d'atteindre une solution pour un problème d'optimisation. La position de chaque particule représente une solution potentielle pour le problème d'optimisation traité. La recherche de solutions dans l'espace de recherche se fait par la mise à jour des positions des particules de l'essaim.

Dans un état initial, les particules sont positionnées aléatoirement dans l'espace de recherche, chaque particule essaye d'atteindre la solution objectif.

Chaque particule se déplace dans l'espace de recherche en suivant sa propre expérience qui représente la meilleure position atteinte par elle même, ou, en suivant l'expérience des particules voisines.

Ainsi, pour déterminer sa prochaine position, une particule considère 3 possibilités:

- Suivre sa propre vitesse en avançant dans l'espace de recherche.
- Suivre sa propre meilleure solution atteinte.
- Suivre la meilleure solution atteinte par les particules voisines.

L'algorithme PSO vise à combiner ces trois comportements afin d'atteindre une solution globale.

Chaque particule a une vélocité avec laquelle elle se déplace dans l'espace de recherche. Dans chaque itération de l'algorithme, une particule change sa position en ajustant sa vélocité. La vélocité est ajustée en ajoutant la vélocité courante + une portion de la vélocité en direction de sa meilleure position + une portion de la vélocité en direction de la meilleure solution voisine.

Chaque particule gardera alors en mémoire les informations concernant sa meilleure position atteinte, ce qui est considéré comme mémoire propre à la particule. La particule peut aussi communiquer avec les particules voisines dans le but de suivre leurs comportements ce qui est considéré comme mémoire partagée. Ces données décideront du prochain état de chaque particule. Ainsi les particules coopèrent en échangeant les informations sur les espaces qu'elles ont exploré. En suivant ce principe l'essaim convergera vers une solution au problème.

2.1 Le voisinage d'une particule

Le voisinage d'une particule représente les particules avec lesquelles une particule coopère. Pour chaque particule on définit un voisinage géographique ou social. Le voisinage géographique est basé sur la distance entre les particules, ainsi les particules les plus proches de la particule en question sont considérées comme ses voisins. Avec le voisinage social, chaque particule a un voisinage défini au début de l'algorithme qui ne changera pas au cours du développement de l'algorithme, même si par la suite ces derniers sont loin de la particule en question. L'avantage du voisinage social est qu'on a pas à calculer la distance entre les particules pour définir le voisinage de chacune. On note que si l'algorithme converge vers une solution, tous les voisins sociaux d'une particule deviennent des voisins géographiques puisque dans le cas de la convergence la distance entre les particules sera réduite. Avec les deux voisinages, le nombre des voisins doit être défini, mais PSO n'est pas sensible à ce paramètre. Dans un cas spécial de PSO, chaque particule considère tout l'essaim de particules comme son voisinage.

3 L'algorithme Sampling-PSO–K-means (SPKM)

L'algorithme SPKM [49] a été présenté comme une solution pour le problème d'initialisation des centroïdes de l'algorithme K-means. La solution consiste à utiliser l'échantillonnage de données et l'algorithme PSO pour la phase de raffinement des centroïdes. Dans cette solution, l'algorithme PSO a été utilisé pour une sélection optimale des centroïdes initiaux. Or, dans le cas où les données analysées sont volumineuses, l'algorithme PSO requière un temps d'exécution très important comme ce dernier n'est pas adapté aux données volumineuses. Ainsi, pour réduire le temps d'exécution, les auteurs ont proposé d'utiliser l'échantillonnage de données afin de réduire la taille des données initiales avant d'être traitées par l'algorithme PSO. La phase d'échantillonnage consiste à diviser l'ensemble des données en des sous-ensembles et d'appliquer l'algorithme K-means sur chacun de ces derniers.

Les centroïdes générés par l'algorithme K-means provenant des différents sous-ensembles représenteront l'ensemble de données initiales et seront utilisés avec l'algorithme PSO comme données de départ. Cette étape réduira la taille des données analysées ce qui réduira le temps d'exécution et améliorera les performances de l'algorithme PSO qui est sensible aux données volumineuses.

L'algorithme SPKM peut être résumé alors dans les étapes suivantes :

– Diviser les données en S sous-ensembles.

– Appliquer l'algorithme K-means sur chacun des S sous-ensembles.

– Utiliser les centroïdes générés dans l'étape précédente avec l'algorithme PSO en les considérant comme les particules de l'essaim (S particules).
– Appliquer l'algorithme K-means sur l'intégralité des données en utilisant comme centroïdes initiaux ceux générés par l'algorithme PSO.

4 Le Framework MapReduce

4.1 Introduction

MapReduce est un Framework introduit par Google permettant de paralléliser certains types de problèmes, spécialement ceux traitant des quantités importantes de données, en utilisant des Clusters de machines [31]. Les problèmes pouvant être traités par MapReduce doivent être faciles à répartir en des sous-tâches indépendantes qui seront traitées parallèlement et simultanément par les machines du Cluster.

MapReduce utilise un ensemble de machines (nœuds du réseau) qui sont réparties en : un nœud « master » et plusieurs nœud « slaves » ou « workers ».

Le nœud master se charge de la gestion du fonctionnement du réseau et des nœuds workers. Il détient les informations essentielles pour le fonctionnement du réseau, comme les adresses des workers et leurs états (libre, en progression, tâche terminée, ou déconnecté). Le nœud master se charge d'attribuer des tâches aux différents workers (Map ou Reduce) selon leurs états de disponibilité, de suivre leurs progressions et de gérer les pannes au niveau des nœuds du réseau. Si un nœud se déconnecte ou ne répond plus, sa tâche sera attribuée à un autre nœud libre, ou elle sera ajoutée à une file d'attente, où par la suite sera attribuée au premier nœud ayant un état libre.

Les nœuds workers ne communiquent pas directement entre eux ainsi, le Framework MapReduce utilise une architecture Master/Slave. Dans sa forme la plus simple, MapReduce est un processus à deux phases : la phase Map et la phase Reduce.

Les nœuds workers se chargent d'exécuter les différentes tâches des phases Map et Reduce. Le nœud master peut aussi servir comme un nœud worker si nécessaire en exécutant des tâches Map et/ou Reduce.

L'avantage du Framework MapReduce réside dans le fait que les tâches Map et Reduce peuvent être distribuées sur les différents nœuds constituant le réseau, et chaque nœud peut exécuter plusieurs tâches Map ou Reduce. MapReduce permet aux programmeurs d'implémenter des solutions distribuées sans avoir à maitriser la programmation distribuée.

4.2 Model de programmation MapReduce

Les parties principales du modèle MapReduce sont les fonctions Map et Reduce qui sont programmées par l'utilisateur. Nous présentons dans ce qui suit les détails de fonctionnement de ces deux fonctions.

4.2.1 Map

Dans la phase Map, le nœud master divise le problème traité en un nombre de parties indépendantes qui sont assignées aux Mappers (map tasks). Chacun des Mappers traite sa partie du problème et génère des résultats sous la forme de paires *Clé-Valeur*.

La partie Map est détaillée dans les étapes suivantes:

- **Étape 1:** Le Framework MapReduce divise les données traitées en plusieurs parties. Chaque point de données de ces parties est formaté en paire (Clé, Valeur), où Clé correspond à un identifiant du mapper auquel cette paire est destinée, et Valeur correspond au point de donnée lui même. Les points avec la même clé constituent une « partie » des données et ils sont destinés au même Mapper. Le Master instancie un Mapper pour chaque partie de données.

- **Étape 2:** Le Mapper exécute sa tâche (Map), et traite les paires (Clé, Valeur) reçues. En traitant les données (Valeur), le Mapper crée de nouvelles paires (Clé, Valeur) de sorte à ce que les données corrélées (selon le problème traité) auront une même Clé.

- **Étape 3:** Les données de sortie des Mappers sont triées de sorte à rassembler les paires avec les mêmes clés. Le nombre de tâches Reduce (Reducers) correspond au nombre de clés distinctes collectées à cette étape.

4.2.2 Reduce

La phase Reduce reçoit les résultats de sortie de la phase Map, où chaque Reducer recevra les résultats depuis les différent Mappers ayant des Clé identiques. Ces Valeurs constitueront une nouvelle partie de données propre au Reducer auquel elle a été destinée. Les Reducers procéderont alors au traitement des données reçues et généreront à leurs tours des résultats sous la forme de Clé-Valeur. Les phases Map et Reduce peuvent être réitérées si nécessaire selon le problème traité.

La partie Reduce est détaillée avec les étapes suivantes :

- **Étape 1:** (Ou étape « Shuffle »). Dans cette étape MapReduce transfère les données de sorties des Mappers vers les Reducers. Cette étape peut commencer avant même que les Mappers aient terminé leurs tâches à 100%.

- **Étape 2:** (ou étape « Sort»). Cette étape est reliée à l'étape Shuffle. L'étape « sort » trie les paires (Clé, Valeur) intermédiaires selon leurs clés. Cette étape aide MapReduce à prendre la décision de créer un nouveau Reducer, comme chaque ensemble de valeurs ayant une même clé doivent être traités par un Reducer distinct.

- **Étape 3:** Après avoir reçu toutes les données qui lui étaient destinées, le Reducer exécute la tâche qui lui est définie par le programmeur. Le Reducer traite les données reçues est génère de nouvelles paires (Clé, valeur) qui correspondent à des résultats intermédiaires dans le cas où d'autres itérations Map/Reduce sont prévues, ou à des résultats finaux pour le problème traité, dans le cas ou le programme s'arrêtera dans cette étape.

D'après ces détails sur les différentes étapes d'un processus MapReduce, on peut noter que MapReduce est une plateforme de tri distribué.

Les phases Map et Reduce sont des parties complémentaires. Comme les Mappers sont indépendants, ils peuvent être exécutés simultanément, ce qui est le cas aussi pour les Reducers. Mais l'étape Map et Reduce sont synchronisés, un Reducer ne peut pas commencer son traitement avant que tous les Mappers aient terminé leurs traitements. Ceci est garanti par le Framework MapReduce sans l'intervention du programmeur.

La figure (3.1) illustre un schéma basique d'un programme MapReduce.

4.3 Avantages de l'utilisation de MapReduce/Hadoop

L'utilisation du Framework MapReduce garantira à l'utilisateur plusieurs avantages sur plusieurs aspects différents. Dans ce qui suit nous citons les points les plus importants.

4.3.1 Calcul Parallèle

Bien évidemment l'avantage principal de MapReduce est la possibilité du traitement parallèle d'une tâche donnée. Ce point est réalisé en divisant cette tâche en plusieurs sous-tâches qui sont traitées parallèlement et simultanément.

On cite par l'occasion que ce point représente aussi un des rares inconvénients de MapReduce car, ce n'est pas tous les programmes qui sont divisibles en des sous-tâches

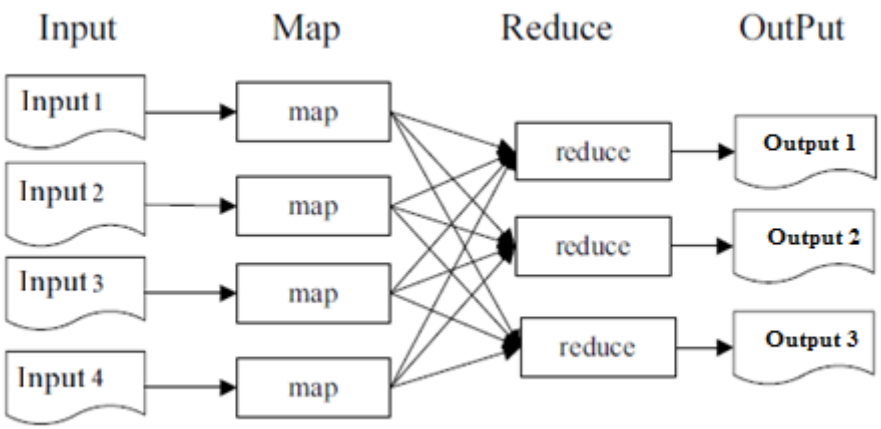


FIG. 3.1 – *Schéma d'un Job MapReduce basique*

indépendantes, ou bien les données analysées nécessitent d'être traitées intégralement. Ainsi, certains types de programmes ne peuvent pas être adaptés au modèle MapReduce.

4.3.2 Evolutivité (Scalabilité)

MapReduce est un Framework hautement évolutif. Ce qui est due à sa faculté de stocker et distribuer une large quantité de données sur un large ensemble de machines, ainsi qu'à sa faculté de distribuer des tâches parallèles sur l'ensemble de ces machines. L'extension d'un Cluster géré par MapReduce requière seulement l'enregistrement d'une nouvelle machine sur ce dernier. Cette dernière aura automatiquement des tâches (Map ou Reduce) assignées si nécessaire sans avoir à modifier le programme exécuté sur le réseau MapReduce. L'espace de stockage de cette machine sera aussi inclus automatiquement dans le système de fichier MapReduce (HDFS).

4.3.3 Rapport Coût/Performances

L'évolutivité du Framework MapReduce implique qu'il est une solution avec un rapport coût/performances avantageux pour les projets nécessitant un large espace de stockage et d'importantes ressources de calcul. L'augmentation des performances d'un Cluster MapReduce requière seulement l'ajout d'une nouvelle machine au Cluster où son espace de stockage et ses ressources de calcul seront ajoutées aux ressources du Cluster.

4.3.4 Disponibilité des données et gestion des erreurs

Quand une machine du Cluster MapReduce recevra un ensemble de données pour le traitement, les autres machines du Cluster recevront aussi ces données. Ainsi, si la machine concernée par le traitement de ces données échouera en raison d'une quelconque erreur, sa tâche sera attribuée à une autre machine disponible et les données seront déjà disponibles. Ceci assurera toujours la disponibilité des données. Ce paramètre est flexible où seulement un nombre réduit de machines recevront les données dupliquées selon la disponibilité de l'espace de stockage.

Un des avantages les plus importants aussi de MapReduce est sa tolérance aux erreurs. MapReduce a la capacité de rapidement reconnaitre les erreurs au niveau de ces nœuds et d'appliquer un processus de restauration si l'un d'eux ne répond pas.

Lors de la gestion d'un large Cluster, ce point devient un avantage très important pour MapReduce car la gestion manuelle devient une tâche très complexe.

4.3.5 Modèle de programmation simple

Parmi les avantages les plus importants de l'utilisation du Framework MapReduce est la simplicité de son modèle de programmation. Ceci permettra aux utilisateurs de développer des programmes MapReduce qui peuvent gérer les tâches Map et Reduce avec facilité et efficacité. De plus ce modèle de programmation épargne aux développeurs la tâche de gestion des communications entre les machines. MapReduce (Hadoop) offre la possibilité de développer ses programmes avec plusieurs langages de programmation comme *Java*,*Python*,*C* etc... ce qui facilitera la courbe d'apprentissage.

5 Notre contribution : l'algorithme «Parallel Sampling-PSO–Multi-core-K-means using MapReduce» (SPKmMR):

Notre objectif est de proposer une parallélisation pour l'algorithme SPKM en utilisant un Cluster de machines géré par MapReduce en se basant sur les techniques et les architectures présentées dans le Chapitre 2. Ces différentes architectures peuvent être appliquées pour la parallélisation de SPKM, chacune apportant ses avantages et ses inconvénients.

5.1 Choix de l'architecture parallèle

Une des solutions possibles pour la parallélisation de SPKM est l'utilisation de l'architecture « parallélisme par tâches ». Avec cette solution les données seront traitées comme un

seul ensemble et une seule instance de l'algorithme SPKM est exécutée. Cela nécessitera un nombre d'itérations pour l'algorithme PSO similaire à une exécution séquentielle. De plus, chacune de ces itérations sera exécutée lors d'un « Job » MapReduce indépendant, ce qui conduira à un processus MapReduce itératif. Ceci ralentira le temps d'exécution de la solution du fait que le Framework MapReduce n'est pas adapté aux processus itératifs.

L'autre solution possible, est l'utilisation de l'architecture SPMD « Single Program, Multiple Data ».

Avec cette architecture, les données seront distribuées sur l'ensemble des machines, où sur chacune une instance de l'algorithme séquentiel SPKM est exécutée.

D'après [49], l'algorithme PSO est sensible aux données volumineuses. Plus les données analysées sont importantes, plus PSO requerra un nombre élevé de particules et d'itérations. De plus PSO converge vers de meilleurs résultats sur des données moins volumineuses. Ce qui a encouragé les auteurs à utiliser l'échantillonnage afin de réduire la taille des données. Ainsi, avec l'architecture SPMD, en plus de la réduction du temps d'exécution grâce à l'utilisation de plusieurs machines, cette architecture nous permettra de réduire la taille des données traitées par l'algorithme SPKM sur chaque machine. Ceci permettra à l'algorithme SPKM d'atteindre de meilleurs résultats en un nombre réduit d'itérations et par conséquent en un temps réduit.

Pour les raisons citées précédemment, notre choix porte sur l'utilisation de l'architecture SPMD.

5.2 Module de parallélisme distribué: Adaptation de l'algorithme SPKM au Framework MapReduce

Dans cette section nous présenterons le module de parallélisation distribué de l'algorithme SPKM en utilisant le Framework MapReduce. Généralement, l'adaptation d'un algorithme de Clustering par partitionnement à l'architecture SPMD dans l'environnement MapReduce consiste en trois phases principales :

- Phase de distribution des données.
- Phase de Clustering local.
- Phase de fusion des résultats.

5.3 Phase de distribution

Dans cette phase, les données sont divisées en des sous-ensembles indépendants, où chacun est assigné à un des nœuds du Framework MapReduce. Les données sont distribuées

aléatoirement comme présenté par les différents travaux dans la littérature [38] [35]. Nous obtenons ainsi des partitions de tailles équilibrées et identiques.

5.4 Phase de Clustering

Cette phase consiste à appliquer l'algorithme de Clustering choisi sur les différents sous-ensembles de données au niveau de chaque nœud du réseau. Dans notre cas nous appliquons l'algorithme SPKM. Ceci engendrera plusieurs résultats de Clustering dits intermédiaires pour chaque sous-ensemble de données.

5.5 Phase de fusion

Enfin, dans cette dernière phase, les résultats obtenus au niveau de chaque nœud sont fusionnés afin d'obtenir une solution globale qui représente le résultat du Clustering de tout l'ensemble de données. Pour cette phase nous avons utilisé la stratégie de fusion simple présentée dans [58][46], qui consiste à fusionner les centroïdes de façon à grouper ceux les plus proches à partir de chaque partition. Chaque centroïde final représentera la moyenne euclidienne des centroïdes qui le constituent.

Ainsi, l'implémentation parallèle de l'algorithme SPKM se résume par les 3 phases principales suivantes :

- Distribution aléatoire des données sur les nœuds du réseau MapReduce.
- Application de l'algorithme SPKM sur chaque partition de données sur chaque machine du réseau MapReduce.
- Fusion des résultats intermédiaires en utilisant la stratégie « fusion simple».

On note que pour les étapes de distribution et de fusion de données, nous avons opté pour des stratégies déjà existantes et utilisées dans plusieurs travaux.

Avec cette solution, sur chaque nœud un algorithme séquentiel est exécuté. Actuellement, la majorité des machines disponibles sont équipées d'un processeur à deux cœurs au minimum. Ainsi, dans le but d'optimiser la solution déjà proposée et afin d'exploiter les ressources disponibles sur toutes les machines utilisées dans le réseau MapReduce, on a proposé d'implémenter un parallélisme local au niveau de chaque nœud en utilisant un parallélisme avec mémoire partagée.

5.6 Module de parallélisme local avec mémoire partagée

Dans cette section nous proposons une parallélisation locale de l'algorithme SPKM au niveau de chaque machine du réseau MapReduce en utilisant un parallélisme avec mémoire partagée.

La parallélisation de l'algorithme SPKM peut avoir plusieurs formes en utilisant différentes techniques et architectures, et peut être réalisée sur différentes parties de cet algorithme.

On remarque que dans l'algorithme SPKM, l'algorithme K-means est utilisé sur deux étapes principales: l'étape d'échantillonnage et l'étape de Clustering final. De plus, lors de la phase d'échantillonnage, K-means est appliqué sur chacun des S sous-ensembles crées. Pour cette raison, l'utilisation d'une version parallèle de K-means réduira considérablement le temps d'exécution sur chaque machine.

Une autre optimisation possible est de paralléliser la partie de raffinement des centroïdes en utilisant l'algorithme PSO, mais cette solution n'est pas prise en considération car les performances de l'algorithme PSO sont déjà optimisées en utilisant l'échantillonnage de données.

Pour l'implémentation parallèle de l'algorithme K-means, nous avons opté pour la solution la plus adaptée et la plus optimale pour les conditions présentes dans notre cas, à savoir l'utilisation d'un seul processeur multi-cœur ainsi que la disponibilité intégrale des données à analyser sur la même machine (au niveau de chaque nœud). Ainsi nous utilisons à la place de l'algorithme K-means séquentiel, l'algorithme multi-core-K-means (McK-means) présenté dans [54].

Cette solution est basée sur l'architecture « parallélisme par tâches », où les phases de calcul des distances minimales et la mise à jour des centroïdes sont parallélisées.

Ainsi dans cette solution, l'ensemble de données de départ est divisé en P partitions, et pour chaque partition l'étape de calcul des distances minimales pour l'affectation des points de données aux Clusters est assurée par un thread indépendant et simultanément.

Du même principe, K threads se chargeront de la mise à jour des K centroïdes simultanément.

L'algorithme Multi-core-K-means est présenté par les étapes suivantes :

1. L'ensemble de données est réparti en P partitions selon le nombre de cœurs physiques disponibles dans le CPU utilisé.
2. P threads sont créés pour chaque partition de données, où chacun est chargé du calcul des distances minimales pour les données de sa partition et de les affecter aux Clusters les plus proches.

3. *K* threads sont créés pour chaque Cluster, où chacun est chargé de mettre à jour le centroïde de son Cluster.

Cette solution permet d'éviter les problèmes de blocage (deadlock) qui est assuré par la synchronisation entre l'étape de calcul des distances minimales et l'étape de mise à jour des centroïdes.

La solution finale proposée nommée « Sampling PSO-Mc-K-means using MapReduce » (SPKmMR) peut être définie alors par les étapes suivantes :

1. Distribution aléatoire des données sur les Mapper de MapReduce.

2. Sur chaque nœud les étapes suivantes sont exécutées :

   (a) Diviser les données en *S* sous-ensembles.

   (b) Un processus d'échantillonnage est appliqué sur les *S* sous-ensembles en utilisant l'algorithme McK-means.

   (c) Utiliser les centroïdes générés dans l'étape précédente avec l'algorithme PSO en les considérant comme les particules de l'essaim (*S* particules).

   (d) Appliquer l'algorithme McK-means sur l'intégralité des données en utilisant comme centroïdes initiaux ceux générés par l'algorithme PSO.

3. Les résultats intermédiaires des différentes partitions sont collectées sur une même machine, qui par la suite seront fusionnés en utilisant la stratégie « fusion simple » pour obtenir une configuration finale de Clusters.

6 Implémentation

Pour implémenter notre approche, nous avons utilisé un Cluster de 3 machines qui sont gérées par le Framework Hadoop. Hadoop est une implémentation open source du Framework MapReduce [87].

Notre approche a été implémentée et testée avec deux Data set numériques multidimensionnels et synthétiques [40] qui sont présentées dans la table (3.1).

Avant l'utilisation des données, ces dernières sont soumises à un pré-traitement qui est important et nécessaire pour des résultats précis et pertinents. Le pré-traitement –dans notre cas- consiste en : (1) Le nettoyage des données où les points de données incomplets sont éliminés ; (2) La normalisation des données si nécessaire selon la nature des données traitées. (3) Adaptation des données au modèle MapReduce « Clé/Valeur ».

TAB. 3.1 – *Data sets*

Data set	Size
DataSet1	1400 vecteurs, 8 dimensions
DataSet2	10125 vecteurs, 15 dimensions

6.1 Codification des données

Selon notre algorithme on dispose de 3 types différents de structures de données :

- Les instances (points) de données principales
- Les Clusters de données pour l'algorithme K-means.
- Les particules de l'essaim pour l'algorithme PSO.

Une **instance de données** appartenant à l'ensemble de données analysées est représentée par le vecteur multidimensionnel suivant :

$$I_i = (a_{i1}, a_{i2}, a_{i3} \ldots, a_{im})$$

où I_i est la i^{eme} instance de données et m est la dimension du Data set.

Un **Cluster** est un ensemble d'instances de données avec une taille variable. Il est représenté par un vecteur de plusieurs instances :

$$C_i = (I_{i1}, I_{i2}, I_{i3}, \ldots)$$

Avec $i \in \{1,2,3,\ldots,K\}$; où K est le nombre de Clusters.

Une **particule** est représentée par un vecteur de K centroïdes où K est le nombre de Clusters définis ; et un centroïde est une instance de données du Data set:

$$P_i = (C_{i1}, C_{i2}, C_{i3}, \ldots, C_{ik}$$

Avec $i \in \{1,2,3,\ldots,l\}$; où l est la taille de l'essaim.

Un **essaim** (Swarm) qui est un ensemble de particule est représenté par un vecteur de l particules :

$$S = (P_1, P_2, P_3, \ldots, P_l)$$

Dans l'étape d'échantillonnage le Data Set est partitionné en plusieurs sous-ensembles où chacun est représenté par un vecteur comme suit :

$$E_i = (a_{i1}, a_{i2}, a_{i3} \ldots, a_{im})$$

où m est la taille du sous-ensemble.

La codification présentée est générale. Elle sera utilisée par tous les algorithmes implémentés et expérimentés. Pour notre approche proposée, cette codification doit être adaptée au modèle MapReduce où chaque instance et modèle de données doit être formaté en une paire $Cl/Valeur$.

Dans notre approche les données sont distribuées aléatoirement sur les machines du Clusters. Ainsi, lors de la phase Map, chaque instance de données lui est attribué une valeur aléatoire qui représentera sa « Clé » et l'instance elle-même représentera la « Valeur » de la paire $Cl/Valeur$.

Ce modèle de codification sera aussi adopté dans les prochains chapitres.

7 Evaluation et expérimentation

L'évaluation des résultats de Clustering dans cette partie consiste à évaluer la qualité des Clusters générés ainsi que la performance en temps d'exécution en comparaison avec d'autre algorithmes. L'évaluation de la qualité de Clustering consiste à vérifier la compacité des Clusters en utilisant la formule calculant la valeur SSW définie dans le premier chapitre.

Pour évaluer l'amélioration apportée par notre approche parallèle hybride Sampling PSO Multi-core K-means using MapReduce (SPKMcKmMR), nous avons implémenté les algorithmes suivants :

- L'algorithme K-means.
- L'algorithme SPKM séquentiel.
- L'algorithme SPKM en utilisant seulement un parallélisme avec mémoire partagé qu'on nomme SPMcKM.
- L'algorithme SPKM en utilisant seulement un parallélisme distribué avec MapReduce qu'on nomme Sampling PSO K-means using MapReduce (SPKmMR).

Pour l'évaluation de la qualité du Clustering nous comparons seulement les algorithmes **K-means**, **SPKM** et **SPKmMR** étant donné que les algorithmes **SPMcKM** et **SPKM** auront les mêmes résultats de Clustering de même pour **SPKmMR** et **SPKMcKmMR**.

Ce dernier algorithme sera testé seulement lors de l'évaluation du temps d'exécution afin de montrer l'importance de l'utilisation d'un parallélisme à deux niveaux (distribué et local).

TAB. 3.2 – *Paramètres des algorithmes*

Paramètres	Valeurs
Facteur d'inertie (PSO)	0.3
Coefficient de confiance position actuelle (PSO)	0.72
Coefficient de confiance au voisinage (PSO)	1.49
Nombre de Cluster (K-means)	3 (DataSet1) - 5 (DataSet2)
Nombre d'itérations (K-means)	5 (DataSet1) - 25 (DataSet2)
Nombre de threads	2 (dynamique selon le CPU utilisé)

7.1 Environnement de test

Pour les tests, on a utilisé un Cluster composé de 3 machines, chacune équipé d'un processeur Intel Dual-core et 2 Go de mémoire. Les machines utilisent le système d'exploitation linux Ubuntu 10.04. Le Cluster est géré par le Framework Hadoop 1.21 qui est une implémentation open source du Framework MapReduce.

7.2 Paramètres des algorithmes

Plusieurs algorithmes implémentés présentent des paramètres communs et des paramètres spécifiques. Les paramètres les plus importants pour l'algorithme PSO sont : le facteur d'inertie et les coefficients de confidences. Pour l'algorithme Multi-core K-means, on cite les paramètres : le nombre de Clusters, le nombre d'itérations et le nombre de threads utilisés qui est fixé à 2 selon le nombre de cœurs physiques disponibles dans les CPU Dual-Core utilisés. Le nombre de Clusters et d'itérations sont définis après plusieurs testes conduits sur les différents Data Sets utilisés.

Les valeurs de ces paramètres sont résumées dans la table (3.2) :

7.3 Résultat

7.3.1 Qualité de Clustering

Pour évaluer la qualité de Clustering nous avons utilisé la formule (1.12) pour calculer la fitness de chaque algorithme implémenté.

TAB. 3.3 – *Performances des algorithmes*

	K-means	SPKM \ SPMcKM	SPKmMR\SPMcKmMR
DataSet1	0.3537	0.2193	0.2101
DataSet2	0.4592	0.2691	0.2451

La table (3.3) contient les résultats obtenus sur les deux différents Data Set.

D'après les résultats présentés dans la table (3.3), et comme mentionné auparavant, les algorithmes SPKM et SPMcKM proposent les mêmes résultats car la seule différence entre ces algorithmes réside dans l'utilisation d'un parallélisme avec mémoire partagée, ce qui n'affecte pas la qualité des résultats. Ceci est le cas aussi pour les algorithmes SPKmMR et SPMcKmMR où leur seule différence est l'implémentation d'un parallélisme local pour l'algorithme SPMcKmMR au niveau de chaque nœud.

On remarque aussi que l'algorithme SPMcKmMR propose les meilleurs résultats avec les plus petites valeurs de fitness sur les deux différents Data Sets. Les deux algorithmes SPKM et SPMcKmMR utilisent l'échantillonnage et l'algorithme PSO pour le raffinement des centroïdes, mais l'algorithme SPMcKmMR apporte de meilleurs résultats. L'utilisation du parallélisme avec MapReduce par l'algorithme SPMcKmMR, où les données sont partitionnées sur l'ensemble des nœuds, réduit la taille des données analysées par l'algorithme PSO. Ceci lui permettra d'atteindre de meilleurs résultats selon [49].

Dans [49], les auteurs ont mentionné qu'un ensemble de données de taille importante requerra un nombre d'itérations plus important pour converger vers un résultat par rapport à un ensemble de données de taille moins importante. Ceci est confirmé par nos résultats où on peut remarquer que l'amélioration apportée sur le Data Set DataSet2 est plus important que celle apportée sur le Data Set DataSet1 en comparant les algorithmes SPKM et SPMcKmMR. Ceci est expliqué par le fait que le Data Set DataSet2 est plus large que le Data Set DataSet1 ce qui lui permettra de mieux bénéficier de notre approche.

Dans ce qui suit, nous discutons la convergence de notre algorithme SPMcKmMR comparé à l'algorithme SPMcKM. Les résultats analysés concernent seulement le Data Set DataSet2. Les résultats de convergence sont présentés dans la figure (3.2).

D'après la figure (3.2), on remarque que l'algorithme SPMcKM apporte les meilleurs résultats après 5 itérations. Avant d'atteindre la 10^{eme} itération, l'algorithme SPMcKmMR propose une meilleure valeur de fitness que l'algorithme SPMcKM.

Apres la 10^{eme} itération, l'algorithme SPMcKM garde la même valeur de fitness et converge à une valeur finale dans la 15^{eme} itération. Contrairement à ça, l'algorithme

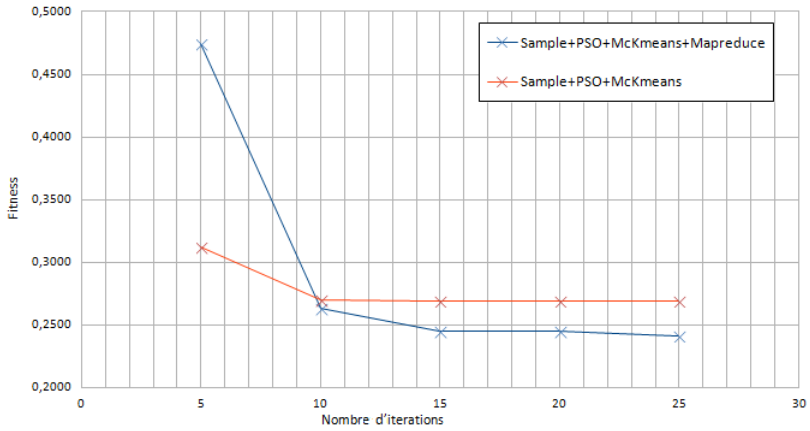


FIG. 3.2 – *Convergence sur le Data Set DataSet2*

SPMcKmMR, continue sa convergence après la 10^{eme} itération jusqu'à atteindre un état stable à la 25^{eme} itération en proposant de meilleurs valeurs que l'algorithme SPMcKM.

D'après ces remarques on note que l'algorithme SPMcKmMR converge vers de meilleurs résultats comparé à l'algorithme SPMcKM dans un même intervalle d'itérations.

On note que les itérations dans cette expérimentation concernent la phase finale du Clustering en utilisant Mc-K-means après l'exécution de l'algorithme PSO, et non pas les itérations de ce dernier (PSO).

7.3.2 Temps d'exécution

Dans ce qui suit nous évaluons le temps d'exécution de notre approche en la comparant avec les autres algorithmes implémentés afin de montrer son efficacité.

Les algorithmes testés sont : SPKM, SPMcKM, SPKmMR et SPMcKmMR.

Ces algorithmes sont testés en utilisant les paramètres présentés dans la table (3.2).

Les résultats obtenus sont présentés dans la table (3.4).

D'après les résultats présentés dans la table (3.4), on remarque que l'algorithme SPKM a le plus long temps d'exécution. En lui appliquant l'algorithme multi-core K-means (SPMcKM) on obtient une amélioration importante au niveau du temps d'exécution, qui est due à l'utilisation d'une version parallèle de K-means (Mc-K-means) à la place de la version séquentielle.

Avec l'algorithme SPKmMR, on remarque que le temps d'exécution est réduit par plus de 50% par rapport à l'algorithme SPMcKM grâce à l'utilisation d'un Cluster de machines, ce qui nous a permis de réduire la taille des données traitées sur chaque machine. Ainsi les machines du Cluster analyseront simultanément des quantités réduites de données ce qui

TAB. 3.4 – *Temps d'exécution des algorithmes implémentés*

Algorithmes	Temps d'exécution (secondes)
SPKM	70,87
SPMcKM	52,94
SPKmMR	24,16
SPMcKmMR	21,98

réduira la complexité de l'algorithme PSO en réduisant son nombre d'itérations nécessaires, ce qui réduira encore plus le temps d'exécution. Ceci montre l'importance de l'utilisant d'un Cluster de machines avec une architecture SPMD dans le cas de traitement des données volumineuses.

Enfin, nous soulignons que notre approche principale SPMcKmMR propose le meilleur résultat en terme de temps d'exécution grâce à l'exploitation des ressources disponibles sur chaque machine en utilisant un parallélisme local, en plus de l'utilisation d'un Cluster de machine géré par MapReduce.

8 Conclusion

Dans ce chapitre, nous avons présenté une approche parallèle hybride pour l'algorithme Sampling-PSO-K-means présenté dans [49]. Dans un premier temps, nous avons proposé d'utiliser un Cluster de machines géré par le Framework MapReduce pour paralléliser l'algorithme étudié. Le choix de l'architecture parallèle s'est fait en prenant en considération la nature de l'algorithme en question et celle du Framework MapReduce. Ensuite, dans le but d'exploiter les ressources disponibles sur les machines utilisées dans le Cluster, nous avons proposé d'implémenter un parallélisme local avec mémoire partagée au niveau de chaque machine du Cluster.

Les résultats ont montré que l'approche utilisée a amélioré les performances de l'algorithme SPKM. La réduction de la taille des données, en les répartissant sur l'ensemble des machines, a permis à l'algorithme PSO d'atteindre de meilleurs résultats en un nombre moins important d'itérations. En plus de l'amélioration de la qualité du Clustering, le temps d'exécution a été considérablement réduit, ce qui représentait l'objectif principal de l'utilisation du parallélisme.

La solution parallèle proposée utilise l'architecture SPMD qui a été choisie pour les avantages qu'elle apporte à l'algorithme SPKM. Comme mentionné dans le chapitre 2, l'architecture SPMD nécessite l'utilisation d'une stratégie pour la distribution des données et une stratégie pour la fusion des résultats. Ainsi, cette architecture permet la possibilité d'améliorer les résultats de l'algorithme de Clustering utilisé en améliorant les stratégies utilisées dans la phase de distribution des données et dans la phase de fusion des résultats. Dans ce chapitre, nous avons utilisé pour ces deux étapes des techniques déjà existantes proposées dans la littérature. Dans le but d'améliorer les résultats obtenus, nous visons dans le prochain chapitre à proposer une nouvelle technique pour la fusion des résultats afin de mieux exploiter les résultats intermédiaires obtenus au niveau de chaque machine.

Chapitre 4

La fusion des résultats intermédiaires du Clustering dans un environnement parallèle (MapReduce) en utilisant les métaheuristiques

1 Introduction

Dans le chapitre précèdent, nous avons présenté une solution parallèle pour l'algorithme SPKM. La solution est basée sur l'environnement MapReduce en utilisant l'architecture SPMD, qui consiste à répartir les données sur un ensemble de machines et d'appliquer l'algorithme SPKM sur chaque partie des données indépendamment. Les résultats obtenus sur chaque machine sont fusionnés afin d'obtenir une configuration globale de Clusters.

Plusieurs techniques sont utilisées pour la fusion des résultats de Clustering, ces techniques présentent des avantages et des inconvénients selon l'algorithme implémenté. Les techniques utilisées pour la fusion des résultats ont un impact direct sur la qualité du Clustering final. Utiliser une stratégie de fusion inappropriée aura un impact négatif sur les résultats finaux. Nous détaillons par la suite dans ce chapitre les techniques communément utilisées pour la fusion des résultats.

Une de ces techniques qui est la « fusion simple » a été utilisée dans notre approche de Clustering parallèle « Sampling PSO-Multi-core-K-means using MapReduce » (SPMcKmMR) présentée dans le chapitre 3. Dans le but d'améliorer cette solution, nous proposons d'optimiser l'étape de fusion des résultats afin d'exploiter les résultats intermédiaires obtenus au niveau de chaque machine du réseau.

La fusion simple comme nous le détaillerons plus tard, utilise tous les résultats intermédiaires. Nous avons supposé que l'utilisation de tous les centroïdes intermédiaires dans la génération de la configuration des Clusters finaux n'est pas une solution optimale. Nous avons ainsi proposé de choisir les K meilleurs centroïdes à partir des centroïdes intermédiaires afin d'obtenir une solution finale optimale.

Cette tâche est considérée comme un problème d'optimisation combinatoire qui nécessite l'utilisation d'une méta-heuristique.

Comme l'utilisation des méta heuristiques sur ce problème n'a pas été proposée auparavant, notre choix porte sur l'utilisation de l'algorithme génétique, qui est considéré comme la méta-heuristique la plus utilisée. Et ceci pour son efficacité et sa simple implémentation.

Cette approche a été présentée dans [17].

Nous avons implémenté l'algorithme SPMcKmMR en utilisant notre stratégie proposée pour la fusion des résultats. On a comparé notre solution avec d'autres techniques de fusion en utilisant le même algorithme de Clustering (SPMcKmMR). Les résultats obtenus ont montré l'efficacité de notre approche. Pour des raisons de simplicité, nous notons l'algorithme SPMcKmMR dans ce chapitre et les chapitres restants avec SPKmMR.

La suite de ce chapitre sera présentée comme suit: Dans la section 2, nous discutons les techniques communément utilisées pour la fusion des résultats intermédiaires des algorithmes parallèles utilisant l'architecture SPMD. Dans la section 3 nous présenterons le principe de fonctionnement de l'algorithme génétique. Avec la section 4 nous détaillerons notre approche proposée pour la fusion des résultats. Dans la section 5 les expérimentations réalisées sont présentées et discutées. Enfin, la section 6 présentera une conclusion pour ce chapitre.

2 Les techniques de fusion des résultats

L'implémentation d'un algorithme de Clustering parallèle, suivant l'architecture SPMD, résultera des ensembles de résultats intermédiaires au nombre des partitions de données analysées parallèlement. Une stratégie doit être établie pour obtenir un résultat final et global à partir de ces derniers.

Dans ce qui suit nous présenterons les solutions communément utilisées pour la fusion des résultats (configuration de Clusters) intermédiaires dans le domaine du Clustering parallèle de données.

2.1 Résultats partiels

Avec cette stratégie les résultats ne sont pas fusionnés. Les différents résultats intermédiaires sont utilisés pour classifier l'ensemble total des données. La configuration de Clusters proposant le meilleur résultat sera choisi comme solution globale. Avec cette stratégie, aucun traitement supplémentaire n'est requis pour la détermination de la solution finale ce qui peut être bénéfique en termes de temps d'exécution. Par ailleurs, une seule configuration de Clusters intermédiaires est prise en considération et toutes les autres sont ignorées, sachant qu'ils peuvent contenir des centroïdes pertinents proposant une meilleure qualité de Clusters. Dans [27], une stratégie pareille a été utilisée afin d'accélérer l'exécution de l'algorithme K-means, où les données sont réparties en des sous-ensembles moins volumineux qui sont analysés séparément. Les résultats partiels apportant les meilleurs résultats sont pris comme des résultats finaux.

2.2 Fusion simple

La fusion simple consiste à fusionner chaque centroïde des différentes partitions (N partitions) avec les $N-1$ plus proches centroïdes de chacune des autres différentes partitions.

La fusion dans ce cas représente la moyenne des centroïdes. Ainsi, chacun des centroïdes finaux représentera une moyenne de N centroïdes. Cette solution est simple à réaliser et ne requière pas un temps d'exécution important. Mais la fusion de deux centroïdes, en calculant leur moyenne, peut produire un centroïdes non précis. Si on veut calculer la moyenne de deux centroïdes A et B et qui ont respectivement M et N points de donnée dans leur Clusters, sachant que $M >> N$, la moyenne réelle de ces deux Clusters devrait être plus proche du centroïde A que du centroïde B. Mais avec cette approche le centroïde résultant sera juste au milieu de A et B. Cette solution a été adoptée dans les travaux [16] [58] [46].

2.3 Fusion avec Clustering

La stratégie de fusion avec Clustering consiste à appliquer un algorithme de Clustering sur les centroïdes intermédiaires. Chaque ensemble de centroïdes intermédiaires peut représenter sa propre partition de données (échantillonnage). Ainsi, l'ensemble de tous les centroïdes intermédiaires peut représenter l'ensemble des données globales. Les centroïdes obtenus du Clustering de l'ensemble des centroïdes intermédiaires seront utilisés pour classifier l'ensemble des données initiales.

Différents algorithmes de Clustering peuvent être utilisés pour cette tâche. Le choix le plus pertinent est d'utiliser un algorithme identique ou proche de celui utilisé lors de la phase principale de Clustering.

En utilisant le Clustering, chaque centroïde final sera présenté par la moyenne des centroïdes les plus proches possible entre eux, contrairement à l'utilisation de la fusion simple qui impose à chaque centroïde d'être généré à partir de N (nombre de partitions) centroïdes, ce qui peut dégrader sa précision. Cette stratégie présente ainsi de meilleurs résultats comparée à la fusion simple. Cette stratégie a été utilisée dans [35] où les algorithmes K-means et K-medoïdes ont été utilisés pour la phase de fusion.

2.4 Fusion avec chevauchement (overlapping)

Cette méthode consiste à fusionner les Clusters qui se chevauchent dans l'espace de données selon un seuil défini. Deux Clusters A et B seront fusionnés s'il existe des points de données appartenant au Cluster A, et qui peuvent aussi appartenir au Cluster B, alors ces deux Clusters seront fusionnées pour constituer un seul nouveau Cluster. Cette stratégie ressemble à la fusion par Clustering du fait que ce processus de fusion par chevauchement se rapproche du principe des algorithmes de Clustering par densité. Avec cette stratégie, le nombre final de Clusters générés peut être différent de la valeur K définie par l'utilisateur. Dans le cas où la valeur K a été bien définie et ne devrait pas être changée ceci présentera un inconvénient pour cette stratégie. Dans [38] [80] la fusion par chevauchement a été adoptée.

Dans la section suivante nous détaillerons le fonctionnement de l'algorithme génétique qui sera utilisé dans la phase de fusion de notre approche.

3 L'algorithme génétique (AG)

L'algorithme génétique (AG) est une méta-heuristique bio-inspirée utilisée pour résoudre les problèmes d'optimisation combinatoire. L'algorithme génétique reproduit le comportement du système d'évolution naturelle dans le but de trouver des solutions pour un problème d'optimisation donné. En utilisant les opérateurs de croisement et de mutation, l'algorithme génétique génère à partir d'une population de nouveaux individus qui pourront être des solutions potentielles pour le problème traité [29].

L'AG a été utilisé dans plusieurs problèmes de Clustering de données. L'AG est connu pour son efficacité dans le problème de sélection des centroïdes initiaux pour l'algorithme K-means. Dans [56], une solution basée sur l'AG a été proposée pour la sélection des centroïdes initiaux pour l'algorithme K-means, l'algorithme sélectionne les K meilleurs centroïdes à

partir de l'ensemble initial des données. Dans [64], une solution pour le clutering de données basée seulement sur l'algorithme génétique a été proposée.

On peut résumer le fonctionnement principal de l'algorithme génétique avec les étapes suivantes:

1. Initialisation de la population.
2. Évaluation de la population en utilisant une fonction fitness.
3. La génération de nouveaux individus pour la population en utilisant les opérations de croisement et de mutation.
4. Évaluation des nouveaux individus en utilisant la fonction fitness.
5. Les meilleurs individus parmi les nouveaux générés et ceux de la population initiale sont sélectionnés pour représenter la population de la génération suivante.
6. Si le nombre de génération maximal n'est pas atteint, aller à l'étape 2 pour une nouvelle génération.
7. A la fin du processus, l'individu avec la meilleure valeur de fitness est désigné comme solution pour le problème.

Dans ce qui suit, nous détaillons les étapes et les notions principales de l'algorithme génétique.

3.1 Initialisation de la population

L'initialisation de la population consiste à générer aléatoirement un ensemble N d'individus ; où N est la taille de la population initiale définie par l'utilisateur. Un individu dit chromosome, représente une solution potentielle pour le problème traité. Un chromosome est représenté généralement par un vecteur de paramètres (gènes). Ainsi, un ensemble de gènes constitue un chromosome, et un ensemble de chromosomes représente une population.

3.2 La fonction fitness

La fonction fitness détermine la qualité de la solution représentée par un individu. Elle calcule une valeur fitness pour chaque individu de la population. C'est à base de la valeur calculée par la fonction fitness que les individus sont comparés. Un individu avec une meilleure valeur de fitness représente une meilleure solution au problème. La fonction fitness utilisée varie selon le problème traité. Elle doit calculer une valeur significative pour le problème.

3.3 La sélection

La phase de sélection consiste à choisir parmi les individus de la population une paire d'individus afin qu'ils passent leurs gènes à un nouveau individu de la prochaine génération (croisement). Le choix de cette paire est basé sur la valeur de fitness ; les deux individus avec les meilleures valeurs de fitness ont plus de chance à être sélectionnés dans le but de produire un individu avec une meilleure solution. Deux individus avec de meilleures valeurs de fitness ne produiront pas forcément un bon individu. Le croisement entre deux individus dont l'un ou les deux avec une mauvaise valeur de fitness peut générer un individu avec une bonne solution. Pour cette raison la sélection basée seulement sur l'aléatoire reste efficace et utilisée.

3.4 Le croisement

Le croisement est l'étape la plus importante de l'algorithme génétique. Cette opération consiste à générer à partir de deux individus parents, un ou plusieurs nouveaux individus. Ceci nous permettra de générer de nouvelles solutions potentielles pour le problème traité.

Pour chaque deux individus choisis pour être croisés, un point de croisement est choisi aléatoirement au niveau duquel les paramètres des solutions contenus dans ces individus sont échangés. Le croisement peut être simple, où les paramètres des solutions sont échangés au niveau d'un seul point de croisement, ou multiple où les paramètres sont échangés au niveau de plusieurs points de croisement. L'opération de croisement est effectuée selon une valeur de probabilité entre 0 et 1 définie par l'utilisateur. Le choix d'une valeur égale à 1 assurera un croisement à tous les coups, ce qui génèrera beaucoup de solutions potentielles, mais il pourra ralentir le temps d'exécution de l'algorithme. Les nouveaux individus générés par l'opération de croisement peuvent avoir ou non de meilleures valeurs de fitness que les individus parents. La figure (4.1) suivante représente un cas de croisement simple entre deux individus résultant deux nouveaux chromosomes.

3.5 La mutation

L'opération de mutation se produit généralement après l'opération de croisement. La mutation consiste à altérer une ou plusieurs des valeurs des paramètres d'un individu donné avec de nouvelles valeurs aléatoires.

Ce changement génère une nouvelle solution qui peut avoir ou non une meilleure valeur de fitness que la solution originale. Comme pour le croisement, la mutation est appliquée sur les nouveaux individus générés selon une valeur de probabilité définie par l'utilisateur.

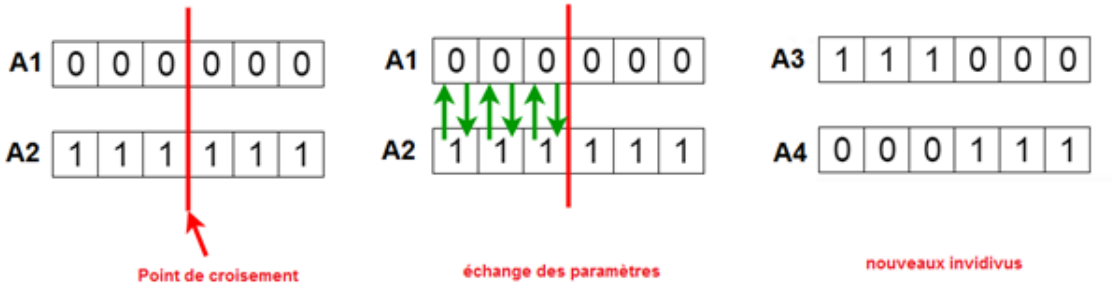


FIG. 4.1 – *Opération de croisement*

FIG. 4.2 – *Opération de mutation*

La figure suivante (4.2) représente un cas de mutation sur un individu au niveau de 3 de ses paramètres (gènes).

3.6 La sélection d'une nouvelle population

La phase de sélection de la nouvelle population consiste à trier et sélectionner, parmi les individus de la population courante et les nouveaux individus générés, les meilleurs éléments selon leurs valeurs de fitness. Ces individus représenteront la nouvelle population de la génération suivante. Il faut noter que certains individus avec une mauvaise fitness peuvent contribuer à la génération de meilleures solutions dans des prochaines générations. Ainsi, inclure ces individus dans une nouvelle population peut contribuer positivement à la diversification de l'espace de recherche ce qui peut conduire à la génération d'une solution optimale.

3.7 Condition d'arrêt

Le processus de l'algorithme génétique s'arrêtera si la population converge i.e. aucun des nouveaux individus n'apporte une amélioration signifiante par rapport aux solutions

courantes après plusieurs générations, ou lorsqu'une solution approximative visée est atteinte. L'arrêt du processus peut aussi être déterminé par le paramètre « nombre de générations » (itérations) qui est défini par l'utilisateur. Dans ce dernier cas la meilleure solution obtenue lors de la dernière génération est considérée comme la solution au problème traité.

4 Notre contribution

Dans cette section, nous présenterons notre contribution pour la fusion des résultats intermédiaires. Dans la section précédente, nous avons présenté différentes stratégies utilisées dans la littérature pour la fusion des résultats. Certaines de ces stratégie présentaient différents inconvénients. Cependant, le point commun entre elles est que tous les centroïdes intermédiaires sont impliqués dans la génération de la solution finale (à l'exception de la méthode « Résultats partiels »).

Nous supposons que l'utilisation de tous les centroïdes intermédiaires conduira à des solutions impertinentes à cause de la qualité de ces centroïdes.

En effet, certains centroïdes peuvent être non-valides et malformés à cause d'une mauvaise initialisation ou à cause de la distribution aléatoire des données lors de leur répartition sur les machines. Nous proposons ainsi de sélectionner parmi ces centroïdes intermédiaires les K centroïdes optimaux afin de représenter la solution finale sans avoir à créer de nouveaux centroïdes. Dans ce qui suit, nous détaillerons les différentes étapes de l'algorithme génétique adapté au problème de fusion des résultats.

4.1 Problème de sélection des K centroïdes finaux

En utilisant l'architecture SPMD, les données sont réparties en N partitions qui correspondent au nombre de machines dans notre cas de figure. A partir de chaque partition, K centroïdes intermédiaires sont générés. Notre objectif est de sélectionner parmi ces $N * K$ centroïdes une combinaison de K centroïdes qui optimisera la qualité du Clustering final.

Ce problème peut être formulé comme un problème d'optimisation combinatoire, qui cherche à trouver la meilleur combinaison de K centroïdes qui minimisent une fonction objectif et ceci parmi toutes les combinaisons possibles qu'on peut obtenir à partir des centroïdes intermédiaire. Le nombre de toutes les combinaisons possibles est : $\frac{n!}{k!(n-k)!}$, avec n le nombre de centroïdes intermédiaires et k le nombre de centroïdes dans une solution potentielle. Avec une augmentation dans la taille de n et dans le pire scénario, nous auront un cas d'explosion combinatoire et ce problème ne pourra pas être résolu dans un temps

FIG. 4.3 – *Chromosome (solution)*

raisonnable. Pour pallier à ce problème, nous proposons d'utiliser l'algorithme génétique pour la sélection des K centroïdes optimaux.

4.2 La fusion des résultats basée sur l'algorithme génétique

Notre approche pour la fusion des résultats en utilisant l'AG se produit après la collection de l'ensemble des centroïdes à partir de toutes les partitions de données. Dans cette solution l'AG prend cet ensemble de centroïdes comme données d'entrée pour la génération de la population de solutions.

Dans ce qui suit, nous présenterons les détails de l'application de l'algorithme génétique au processus de fusion des résultats intermédiaires.

4.3 Population et chromosomes

L'algorithme génétique prend comme paramètre d'entrée l'ensemble de centroïdes $\bar{C}$ où:

$$\overline{c} = \bigcup_{i}^{n} c_i \tag{4.1}$$

C_i est l'ensemble des centroïdes générés avec la i^{eme} partition de données. La population initiale de l'algorithme génétique est générée aléatoirement à partir de cet ensemble $\bar{C}$. N individus (solutions) sont générés, où N représente la taille de la population.

Chaque individu de la population est un vecteur de taille K constitué de K centroïdes choisis aléatoirement à partir de l'ensemble $\bar{C}$. Un individu représente une solution potentielle finale pour la classification de toutes les données analysées.

Durant chaque génération (itération) les meilleurs N individus sont sélectionnés pour passer à la génération suivante.

La figure (4.3), représente une solution potentielle (chromosome) dans le cas où le nombre de Cluster k est égal à 6. Le premier élément de la solution qui est $C2{,}5$ représente le 5^{eme} centroïde provenant de la 2^{eme} partition de données, l'élément suivant $C8{,}6$, représente le 8^{eme} centroïde provenant de la 6^{eme} partition est ainsi de suite.

A partir de la figure (4.3), on peut constater que les centroïdes constituants le chromosome sont issus de différentes partitions, mais ce n'est pas toutes les partitions qui ont des centroïdes

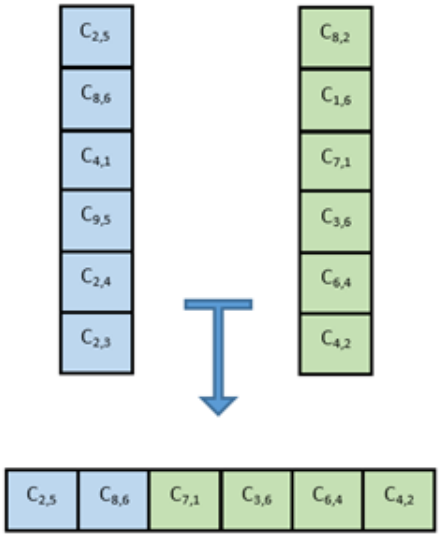


FIG. 4.4 – *Croisement de deux configurations de Clustering*

présents dans la solution finale. Plusieurs centroïdes issus de la même partition peuvent aussi être présents dans la même solution selon la situation.

4.4 Les opérations de croisement et de mutation

A chaque génération, plusieurs nouvelles solutions sont générées grâce aux opérateurs de croisement et de mutation. Dans notre approche, le croisement consiste à combiner deux configurations de Clusters choisies aléatoirement de la population afin de générer une nouvelle solution qui peut être éligible pour représenter la configuration finale pour tout l'ensemble de données. La figure (4.4) représente un exemple de croisements de deux configurations de Clusters générant une nouvelle configuration.

On remarque que les éléments du nouveau chromosome généré dans la figure (4.4) sont uniques i.e. aucun centroïde n'est présent plus d'une fois dans le même chromosome. Après plusieurs générations, il se peut qu'une solution issue d'un croisement contienne un ou plusieurs centroïdes redondants. Une solution pareille est considérée comme une solution non valide. Pour pallier à ce problème, une étape de vérification s'impose après chaque opération de croisement. Les redondances sont échangées avec d'autre centroïdes choisis aléatoirement à partir de l'ensemble des centroïdes $\bar{C}$. Ce processus de vérification est répété jusqu'à l'obtention d'une solution composée d'un ensemble de centroïdes uniques.

Comme pour le croisement, l'opération de mutation contribue à la génération de nouvelles configurations de Clusters. Dans notre solution, l'application de la mutation à un chromosome

consiste à changer un des K centroïdes qui le constituent avec un autre centroïde choisi aléatoirement à partir de l'ensemble des centroïdes $\bar{C}$. Afin de garantir la validité de la nouvelle solution, le même processus de validation utilisé à l'étape de croisement est appliqué sur la solution issue de la mutation.

4.5 L'évaluation des chromosomes (la fonction fitness)

L'évaluation des individus de la population dans cette approche consiste à déterminer la qualité des Clusters contenus dans ces derniers.

Nous avons choisi d'utiliser la formule SSW (1.12). La valeur de fitness sera alors égale à la valeur calculée par la formule SSW.

Comme déjà mentionné, la formule SSW calcule la distance moyenne entre les données de chaque Cluster et leur centroïde. Ce calcul est réalisé pour chaque individu de la population et pour tous les nouveaux individus générés à chaque génération, où les données de départ prévues au processus de Clustering sont classifiées en utilisant les centroïdes de ces derniers. L'utilisation de tout l'ensemble de données de départ avec chaque individu requerra un temps d'exécution très important.

Afin d'optimiser ce processus, seul un échantillon des données est utilisé dans l'évaluation afin de réduire le temps d'exécution. Dans une étape finale, la meilleure solution générée est réévaluée avec l'ensemble complet de données pour obtenir la valeur de fitness finale représentant la solution.

5 Implémentation et expérimentation

5.1 Implémentation

Pour l'analyse de notre proposition, nous avons implémenté notre algorithme SPKmMR avec trois différentes techniques de fusion de résultats y compris notre stratégie proposée :

1. Fusion simple.
2. Fusion par Clustering.
3. Fusion par algorithme génétique.

Avant l'application du processus de Clustering, un pré-traitement est appliqué aux données. Les étapes du pré-traitement sont les mêmes cités dans le chapitre 3.

5.2 Évaluation

L'évaluation d'une solution consiste à déterminer à quel point les instances de données d'un même Cluster sont proches. On utilise pour cela la formule (1.12) qui calcule la valeur SSW comme dans le chapitre précédent. Cette formule est utilisée par les différents algorithmes impliqués dans les différentes approches implémentées, à savoir : l'algorithme K-means, l'algorithme PSO ainsi que l'algorithme génétique comme cité auparavant.

5.3 Experimentation

Pour démontrer l'importance de l'étape de fusion des résultats et pour souligner l'efficacité et l'amélioration apportée par notre approche, nous avons implémenté les algorithmes suivants : l'algorithme K-means, l'algorithme Sampling-PSO-K-means (SPKM), l'algorithme « K-means using MapReduce » (KmMR) et l'algorithme « Sampling-PSO-Mc-K-means using MapReduce » (SPKmMR).

L'algorithme KmMR est une implémentation parallèle de l'algorithme K-means en utilisant MapReduce. Cet algorithme est identique à l'algorithme SPKmMR sauf qu'à la place d'utiliser l'algorithme SPMcKM sur chaque machine on utilise l'algorithme K-means.

Notre méthode de fusion de résultats proposée, ainsi que les deux autres méthodes implémentées sont testées avec les algorithmes KmMR et SPKmMR.

Pour évaluer notre proposition, nous avons utilisé un Cluster de 5 machines, chaque machine est équipée d'un CPU Dual-Core et de 2 Go de mémoire. Toutes les machines utilisent le système d'exploitation Ubuntu 10.04. Pour la gestion du Cluster, nous avons utilisé le Framework Hadoop 1.2.1.

Les expérimentations ont été conduites sur un Data Set de données synthétique [40] (DataSet1) et sur le Data Set de données réel 'The Individual household electric power consumption' [59] (DataSet2), qui représente les données de consommation d'électricité dans un ensemble de ménages.

Plusieurs algorithmes ont été implémentés pour évaluer notre approche. Chaque algorithme présente un ensemble de paramètres spécifique. Pour l'algorithme PSO, la table (4.1) résume les paramètres utilisés pour les Data Sets DataSet1 et DataSet2 (selon [49]).

La table (4.2) résume les paramètres utilisés par l'algorithme K-means utilisé lors du Clustering principale ainsi que dans l'étape de fusion.

Pour le nombre de Cluster on a utilisé deux valeurs différentes pour le Data Set DataSet2: 50 et 100.

La table (4.3) résume les paramètres utilisés par l'algorithme génétique.

TAB. 4.1 – *Paramètres pour l'algorithme PSO*

Paramètres	Valeurs	
	DataSet1	DataSet2
Inertia factor	0.5	0.3
Coefficient de confiance (Position courante)	0.1	0.72
Coefficient de confiance (Voisinage)	3.0	1.49

TAB. 4.2 – *Paramètres K-means*

Paramètres	Valeurs
Nombre de partitions (Pour MapReduce)	16 (Nombre de machines)
Nombre de Clusters	11 (DataSet 1) / 50 –100 (DataSet 2)
Numbre d'iterations pour K-means	15 (DataSet 1) / 25 (DataSet 2)
Nombre de threads (pour Multi-core K-means)	2 (dynamique selon le CPU utilisé)

TAB. 4.3 – *Paramètres pour l'algorithme génétique*

Paramètres	Valeurs
Taille de la Population	50
Nombre de générations	100
Croisement %	100 %
Mutation %	80 %

TAB. 4.4 – *Résultats des algorithmes séquentiels(SSW)*

Algorithmes	Data Sets		
	DataSet1	DataSet2 (k=50)	DataSet2 (k=100)
K-means	0.079604	0.079233	0.067021
SPKM	0.077303	0.073251	0.063955

TAB. 4.5 – *Qualité du Clustering avec les différentes stratégies de fusion des résultats*

Techniques	KmeansMR			SPKmMR		
de fusion	DataSet1	DataSet2 (k=50)	DataSet2 (k=100)	DataSet1	DataSet2 (k=50)	DataSet2 (k=100)
Fusion simple	0.079888	0.077204	0.064242	0.0775832	0.071459	0.063574
Fusion par K-means	0.078829	0.071763	0.061751	0.077153	0.070317	0.059569
Fusion par AG	0.077211	0.057541	0.051913	0.0767487	0.054363	0.049195

Dans ce qui suit nous discutons les résultats obtenus par chaque algorithme implémenté. Les résultats sont évalués en utilisant la formule (1.12).

La table (4.4) représente les résultats obtenus par les algorithmes séquentiels K-means et SPKM sur les Data Sets DataSet1 et DataSet2.

La table (4.5) présente les résultats obtenus par les algorithmes parallèles implémentés avec les différentes techniques de fusion de résultats sur les Data Sets DataSet1 et DataSet2.

Les résultats présentés sur la Table (4.5) montrent la différence dans les résultats obtenus en utilisant de différentes techniques pour la fusion des résultats intermédiaires.

De la table (4.5), avec la fusion simple et pour les deux Data Sets, on remarque que les résultats obtenus avec l'algorithme KmMR sont légèrement meilleurs que ceux obtenus avec l'algorithme K-means séquentiel présentés sur la table (4.4).

De plus, pour le Data Set DataSet1, on remarque que l'algorithme K-means séquentiel a des résultats légèrement meilleurs que ceux obtenus avec l'algorithme KmMR avec la fusion simple.

Pour l'algorithme SPKmMR (table 4.5), dans le cas où $K = 100$ les résultats sont presque identiques à ceux de l'algorithme SPKM séquentiel (table (4.4)). Dans le cas où $K = 50$ les résultats se sont légèrement améliorés par rapport à l'algorithme SPKM.

De la même manière que pour l'algorithme KmMR, l'algorithme SPKmMR, en utilisant la fusion simple, ne réussit pas à obtenir sur le Data Set DataSet1 des résultats identiques à sa version séquentielle.

Ces résultats non satisfaisants sont dus à l'incapacité de la technique de fusion simple à exploiter les résultats intermédiaires.

Avec la fusion par Clustering, on remarque qu'avec les deux algorithmes KmMR et SPKmMR et avec les différents Data Set, les résultats obtenus sont meilleurs que ceux obtenus en utilisant les algorithmes séquentiels et ceux parallèles en utilisant la fusion simple sur tous les Data Sets.

Les meilleurs résultats de Clustering des deux algorithmes implémentés avec les différents Data Set et les différentes valeurs de K sont obtenus en utilisant notre approche de fusion de résultats en utilisant l'algorithme génétique.

D'après ces résultats on remarque aussi que les résultats de l'algorithme K-means parallèle (KmMR), en utilisant la fusion avec Clustering et la fusion avec AG, sont meilleurs dans la plupart des cas que les résultats de l'algorithme SPKM séquentiel ou parallèle avec une fusion simple. Ceci montre l'importance de l'étape de fusion pour une meilleure exploitation des résultats intermédiaires.

La figure (4.5), illustre les résultats obtenus par les algorithmes KmMR et SPKmMR en utilisant les 3 techniques de fusion de résultats utilisées dans le cas où $K = 100$.

La figure (4.5) nous permet de mieux visualiser et distinguer une des importantes améliorations apportées par notre méthode de fusion avec AG.

On note alors que l'amélioration apportée par l'algorithme SPKmMR par rapport à l'algorithme KmMR, en utilisant une sélection non aléatoire des centroïdes initiaux (PSO), est plus importante et plus visible lors de l'utilisation de la fusion avec Clustering que lors de l'utilisation d'une fusion simple. De même, la différence en qualité de Clustering est encore plus importante en utilisant la fusion avec algorithme génétique.

En utilisant cette dernière méthode, les résultats intermédiaires obtenus par l'algorithme SPKM au niveau de chaque machine, et qui sont meilleurs que ceux générés par l'algorithme K-means (avec l'algorithme KmMR), sont mieux exploités, et seulement les K meilleurs centroïdes sont retenus pour la solution finale.

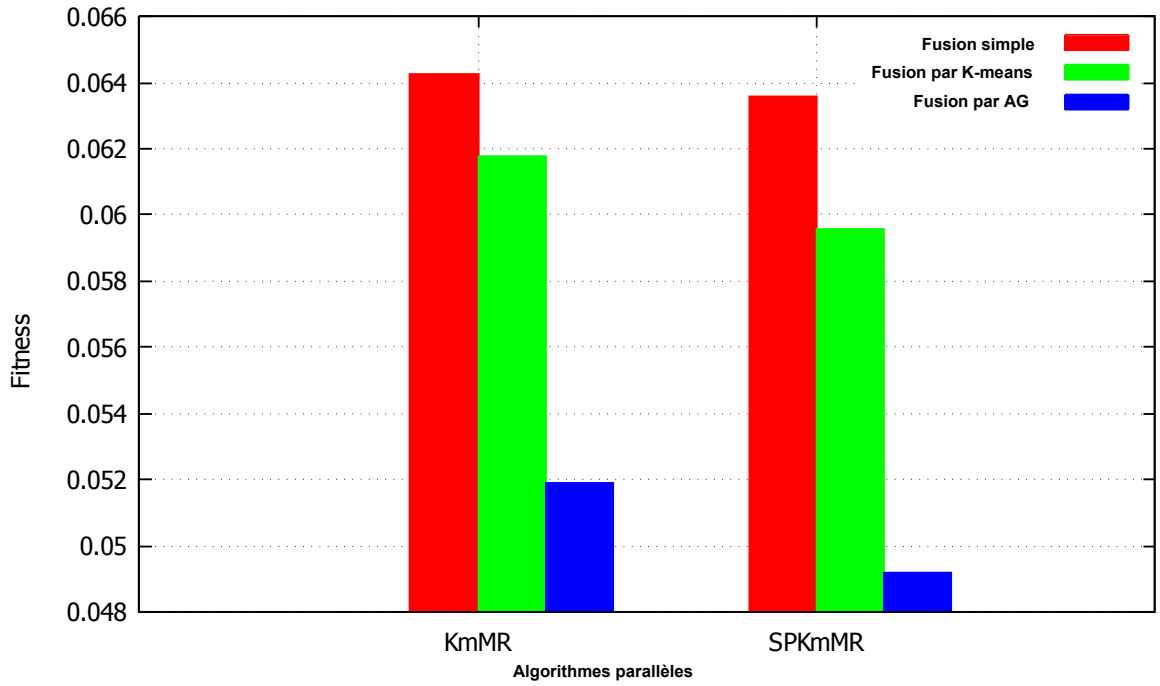


FIG. 4.5 – *Performances des techniques de fusion*

6 Conclusion

Dans ce chapitre nous avons présenté une solution pour le problème de fusion des résultats intermédiaires pour un algorithme de Clustering parallèle dans l'environnement MapReduce.

La solution a été proposée afin d'améliorer les résultats de l'algorithme SPKmMR [16] présenté dans le chapitre 3.

Dans cette solution l'algorithme génétique a été utilisé pour sélectionner les centroïdes les plus pertinents afin de représenter la solution finale du Clustering.

L'approche a été testée sur un Data Set réel et comparée avec d'autres méthodes de fusion de résultats (fusion simple, fusion par Clustering/K-means). L'approche proposée a été aussi testée avec l'algorithme KmMR afin de prouver son efficacité avec d'autres algorithmes.

Les résultats des expérimentations ont montré l'efficacité de notre solution en améliorant considérablement la qualité des Clusters générés par les algorithmes SPKmMR et KmMR.

Dans le but d'améliorer d'avantage la qualité des Clusters, nous proposons dans le prochain chapitre une solution non-aléatoire pour la répartition des données sur les machines du Framework MapReduce.

Chapitre 5

Distribution des données et fusion des résultats pour le Clustering parallèle dans l'environnement MapReduce : Présentation d'un nouveau schéma parallèle

1 Introduction

Dans les chapitres précédents nous avons présenté deux travaux différents concernant le Clustering parallèle utilisant le Framework MapReduce. Une des solutions consistait en la proposition d'une parallélisation d'un algorithme de Clustering en utilisant principalement le Framework MapReduce avec l'architecture SPMD tout en implémentant un parallélisme local au niveau de chaque machine. Par la suite et dans le but d'améliorer cette dernière, nous avons proposé une nouvelle solution pour la fusion des résultats intermédiaires en utilisant l'algorithme génétique. Après l'analyse des résultats obtenus précédemment, nous avons constaté la possibilité d'améliorer les solutions proposées en utilisant une méthode non-aléatoire lors de la phase de distribution des données au niveau des Mappers.

La solution vise à distribuer les données de façon à maximiser la distance entre les données présentes sur chaque machine du Cluster. En suivant cette méthodologie, nous supposons que la distance entre les centroïdes des Clusters générés sur chaque machine sera maximisée à son tour, ce qui est l'objectif principal des algorithmes de Clustering.

Cette méthode de distribution de données est supposée apporter des résultats pertinents dans le cas où la méthode de fusion de résultats utilisée n'est pas conventionnelle, comme la méthode de fusion par algorithme génétique que nous avons proposé dans le chapitre précédent. Ce point sera justifié plus tard dans la partie expérimentale. Pour cette raison, notre solution pour la distribution des données est liée à notre solution de fusion des résultats par algorithme génétique.

Nous avons défini alors un nouveau modèle pour l'acheminement des données dans le Framework MapReduce. Ce modèle commence par une distribution non aléatoire des données et se termine par la fusion des résultats en utilisant l'algorithme génétique.

Après expérimentations, nous avons remarqué que cette solution apporte des améliorations non seulement pour notre algorithme étudié dans cette thèse qui est SPKmMR, mais pour d'autres algorithmes de Clustering par partitionnement.

Ainsi, dans ce chapitre nous proposons un schéma général pour la parallélisation des algorithmes de Clustering par partitionnement dans le Framework MapReduce. Les améliorations apportées par ce schéma résident dans les méthodes utilisées pour la distribution des données et pour la fusion des résultats.

Notre approche a été testée est comparée à des algorithmes récents de Clustering parallèle. Les résultats des tests ont montré l'efficacité et l'évolutivité de notre solution en terme de temps de calcul. En termes de qualité de Clustering, les résultats de notre approche sont supérieurs à ceux des autres algorithmes testés.

Dans une étape finale, et dans le but de prouver la pertinence et la validité des résultats apportés par notre approche, nous l'avons appliquée sur un problème de Clustering réel, qui consiste en la détection des communautés d'utilisateurs dans les réseaux sociaux en utilisant les algorithmes de Clustering. Encore une fois, les résultats ont montré l'efficacité de notre solution.

La suite de ce chapitre est organisée comme suit : Dans la section 2 nous présentons les différentes méthodes utilisées pour la répartition des données par les algorithmes de Clustering parallèle. Dans la section 3 nous détaillons notre approche qui est un schéma parallèle pour les algorithmes de Clustering par partitionnement. La section 4 sera la partie expérimentale du chapitre où nous testerons les différents aspects de notre approche et on la comparera avec d'autres algorithmes. Dans la section 5, notre approche sera appliquée au problème de détection des communautés d'utilisateurs dans le réseau social Bibsonomy. La section 6 sera une conclusion pour le chapitre.

2 Les méthodes de distribution de données pour le Clustering parallèle

L'utilisation de l'architecture SPMD nécessite non seulement une stratégie pour la fusion des résultats comme mentionné dans le chapitre précédant, mais aussi une méthode pour la distribution des données. Différentes stratégies pour la distribution des données existent [82] [38]. Elles sont utilisées selon le type du problème traité. Dans ce qui suit nous citons les différentes méthodes utilisées pour la répartition des données.

2.1 Distribution séquentielle (aléatoire)

Avec cette stratégie les données sont distribuées aléatoirement sur les machines en des groupes distincts. Les données existantes dans une partition donnée ne peuvent pas être présentes sur les autres partitions.

La distribution aléatoire peut s'effectuer soit en assignant chaque point de données à part à une partition aléatoire, ou bien en assignant un ensemble de points de données à une partition en gardant leur ordre comme dans le Data Set initial ce qui est considéré aussi comme une répartition aléatoire. Cette dernière méthode peut être nécessaire dans certains cas où l'ordre des données est important.

2.2 Distribution avec chevauchement (cover-based distribution)

Cette distribution correspond à la première stratégie sauf que dans ce cas, certains points de données peuvent être présents dans différentes partitions. Le but de cette stratégie est d'augmenter la probabilité des points de données dupliqués à être dans la bonne partition afin de participer à la génération de modèles de Clusters pertinents. Car, avec la distribution des données sur plusieurs partitions, les données présentes sur chacune peuvent ne pas être suffisamment corrélées afin de pouvoir générer une solution correcte de Clusters.

2.3 Distribution selon les valeurs

A la différence des deux premières stratégies, les données sont distribuées selon les valeurs de certains attributs. L'utilisateur peut créer des partitions de données où, par exemple, les valeurs d'un des attributs seront dans un intervalle bien défini. Cette méthode peut être appliquée dans le cas où l'utilisateur a des connaissances préalables sur la nature des données. Par exemple, tous les points de données ayant une valeur X au niveau de l'attribut Y doivent

être dans un même Cluster. Cette stratégie peut être mieux adaptée au algorithmes de Clustering par contraintes.

Remarque On note que dans le domaine du Clustering de données, les algorithmes basés sur le Framework MapReduce présentés dans la littérature utilisent principalement la méthode de distribution séquentielle aléatoire.

3 Notre contribution : un schéma parallèle pour le Clustering de données

Dans cette section nous présentons notre approche pour le Clustering parallèle qui a été présentée dans [18]. Au début, nous présenterons les étapes principales du schéma parallèle proposé. Ensuite, avant de détailler chacune de ces étapes, nous citons les différences entre notre approche et les travaux sur le Clustering parallèle utilisant le Framework MapReduce. Enfin, nous présentons en détails les différentes phases de notre solution.

Notre schéma parallèle peut être résumé dans les 3 étapes suivantes :

1. L'ensemble de données est réparti sur les machines du Cluster lors de la phase Map, de manière à maximiser la distance entre les points de données des différentes partitions (étape Map).
2. Un algorithme de Clustering est appliqué sur chaque partition au niveau de chaque Reducer (étape Reduce).
3. Après la collecte des résultats de chaque Reducer, ces derniers sont fusionnés en utilisant l'algorithme génétique.

D'après cette description, on remarque que la première différence par rapport aux travaux précédents est la méthode utilisée pour la répartition des données. Tous les travaux cités ont négligé l'étape de distribution des données en optant pour une distribution aléatoire.

Les solutions basées sur l'aléatoire ont apporté des résultats satisfaisants qui se rapprochaient de ceux des algorithmes séquentiels, ce qui a été constaté avec les résultats obtenus dans les chapitres précédents. L'utilisation de l'aléatoire nous a fournit des partitions de données équilibrées contenant des points de données diversifiés qui dans la plupart des cas peuvent être considérés comme des échantillons du Data Set initial. Ceci nous permettra d'avoir des résultats intermédiaires corrects, qui, avec la méthode adéquate de fusion, fourniront des résultats finaux pertinents.

Cependant, l'approche aléatoire ou toute autre approche utilisée, peut conduire à la création de partitions où les données contenues ne sont pas adéquates à la génération d'un

pattern de données. Comme pour la phase de fusion, la distribution des données a un impact direct sur la qualité du Clustering. Elle est donc la première étape essentielle pour l'obtention d'une configuration de Clusters précise.

Ainsi, contrairement à ces approches, nous utilisons une distribution non-aléatoire basée sur la similarité entre les partitions, où elle sera maximisée entre les données d'une même partition est minimisée entre les données provenant de différentes partitions.

La seconde différence entre notre approche et les autres travaux réside dans la méthode utilisée pour la fusion des résultats, où nous proposons d'utiliser une méthode basée sur l'algorithme génétique. Ce point a été détaillé dans le chapitre 4.

Dans la prochaine section nous présenterons l'approche utilisée pour la répartition des données.

3.1 Module de distribution des données

Les algorithmes de Clustering par partitionnement visent à découvrir des partitions de données où la dissimilarité entre les objets des différentes partitions est élevée.

Partant de ce principe, nous proposons de distribuer les données sur l'ensemble des machines de façon à maximiser la distance entre les points de données des différentes partitions. Avec cette stratégie nous supposons que les centroïdes générés au niveau de chaque « Reducer » (partition) seront le plus dissimilaires possibles des autres centroïdes générés par les autres « Reducers » en utilisant leurs propres partitions de données.

Pour réaliser cette tâche, nous procédons à classifier les données initiales sur l'ensemble des machines au lieu de les répartir aléatoirement. Chaque Reducer recevra ainsi les données d'un seul Cluster.

Pour pouvoir classifier les données sur l'ensemble des machines nous aurons besoin d'un pattern de données à suivre. Pour créer ce pattern, nous appliquons un algorithme de Clustering sur tout l'ensemble des données avec un nombre de Clusters K égal au nombre de machines utilisées. Cependant, puisque nous traitons un problème de Clustering parallèle où la taille des données est très importante, cette étape de création du pattern requerra un temps d'exécution très important.

Dans [27], les auteurs ont montré que l'utilisation d'un échantillon de données choisi aléatoirement représentant 25% des données de base peut produire une configuration de Cluster proche de celle obtenue en utilisant tout l'ensemble de données. Nous adoptons ainsi cette stratégie d'échantillonnage afin de créer un pattern de données dans un temps d'exécution réduit. L'algorithme K-means est utilisé pour la création du pattern de données. La quantité de données utilisée peut varier selon la nature du Data Set utilisé.

La méthode proposée pour la distribution des données peut être résumée dans les étapes suivantes :

- L'extraction d'un échantillon représentant 25% de l'ensemble des données initiales.
- Un Clustering est appliqué sur l'échantillon de données en utilisant l'algorithme K-means avec un nombre de Clusters K égal au nombre de machines du Cluster (nombre des partitions/Reducers).
- Le pattern de données créé par le Clustering est envoyé à tous les Mappers.
- Durant l'étape Map, les Mappers utilisent le pattern de données pour assigner chaque point de données au plus proche Cluster qui représente une partition de données et qui est traité par un seul Reducer.

La phase Map qui représente le processus de distribution de données est décrite avec l'algorithme 1 :

Algorithme 1 : Phase Map

Input : C :Distribution des centroïdes, Clé: index de l'instance de données, Valeur: point de données
Output : (Clé,valeur) avec: Clé est l'index du Reducer et valeur est l'instance de données.

```
instance:=Valeur;
Index:=1;
DistanceMin:=calculaterDist(C1,Valeur);
for i ← 2 to C.length do
    d:=calculaterDist(Ci,Valeur);
    if d < DistanceMin then
        DistanceMin:=d;
        Index:=i;
Clé:=Index;
Valeur:=instance;
Output:paire (Clé,Valeur) ;
```

La phase Reduce qui représente le processus du Clustering est représentée par l'algorithme 2 :

En appliquant notre approche de distribution, les partitions générées peuvent ne pas être équilibrées puisque la taille des Clusters générés par K-means n'ont pas forcément des tailles identiques, ce qui n'était pas le cas en utilisant la distribution aléatoire. Ainsi, certaines partitions de données peuvent ne pas être en mesure de découvrir des configurations correctes de Clusters due au nombre insuffisant de données.

Algorithme 2 : Phase Reduce

Input : Clé: index du Reducer, Valeurs: instance de données appartenant reducer designé par Clé

Output : (Clé,valeurs) avec Clé est l'index du Reducer et valeurs sont les centroïdes intermediaires généré par le Reducer actuel.

instances[Valeurs.length];
i:=1;
foreach *valeur* ∈ *Valeurs* **do**
    instances[i]:=valeur;
    i++;
CentroidesIntermediaires:=Clustering(instances,K); //Appliquer un Clustering local sur la partition de données locale avec K:nombre de Cluster
Value:=IntermediateCentroids;
Output:(Key,Value) pair;

De plus, contrairement à la distribution aléatoire où les données des partitions sont diversifiées et peuvent représenter des échantillons pour les données initiales, avec notre distribution, chaque partition représente seulement une partie des données et les Clusters générés sur chacune ne peuvent pas être considérés comme une solution approximative proche de la solution globale obtenue avec tout l'ensemble des données.

Ainsi, seule une combinaison des meilleurs centroïdes des différentes partitions peut fournir un résultat final, global et pertinent. Pour ces raisons, nous utilisons pour notre schéma parallèle l'approche de fusion par algorithme génétique présentée dans le chapitre 4. Ce point sera plus discuté dans la partie expérimentale.

Le processus entier de notre schéma peut être résumé par la figure (5.1) suivante :

3.2 Étude de la complexité temporelle

Dans cette section nous étudions la complexité de notre approche. La complexité de notre approche consiste en la somme des complexités de ses différentes phases qui sont :

1. Complexité de la phase de distribution.
2. Complexité de la phase de Clustering.
3. Complexité de la phase de fusion.

Soit n la taille du Data Set analysé, d la dimension des instances de données (nombres d'attributs) et k le nombre de Clusters. Comme nous travaillons sur de larges ensembles de données on aura : $n >> k$ et $n >> d$.

Dans ce qui suit la complexité de chaque étape est détaillée.

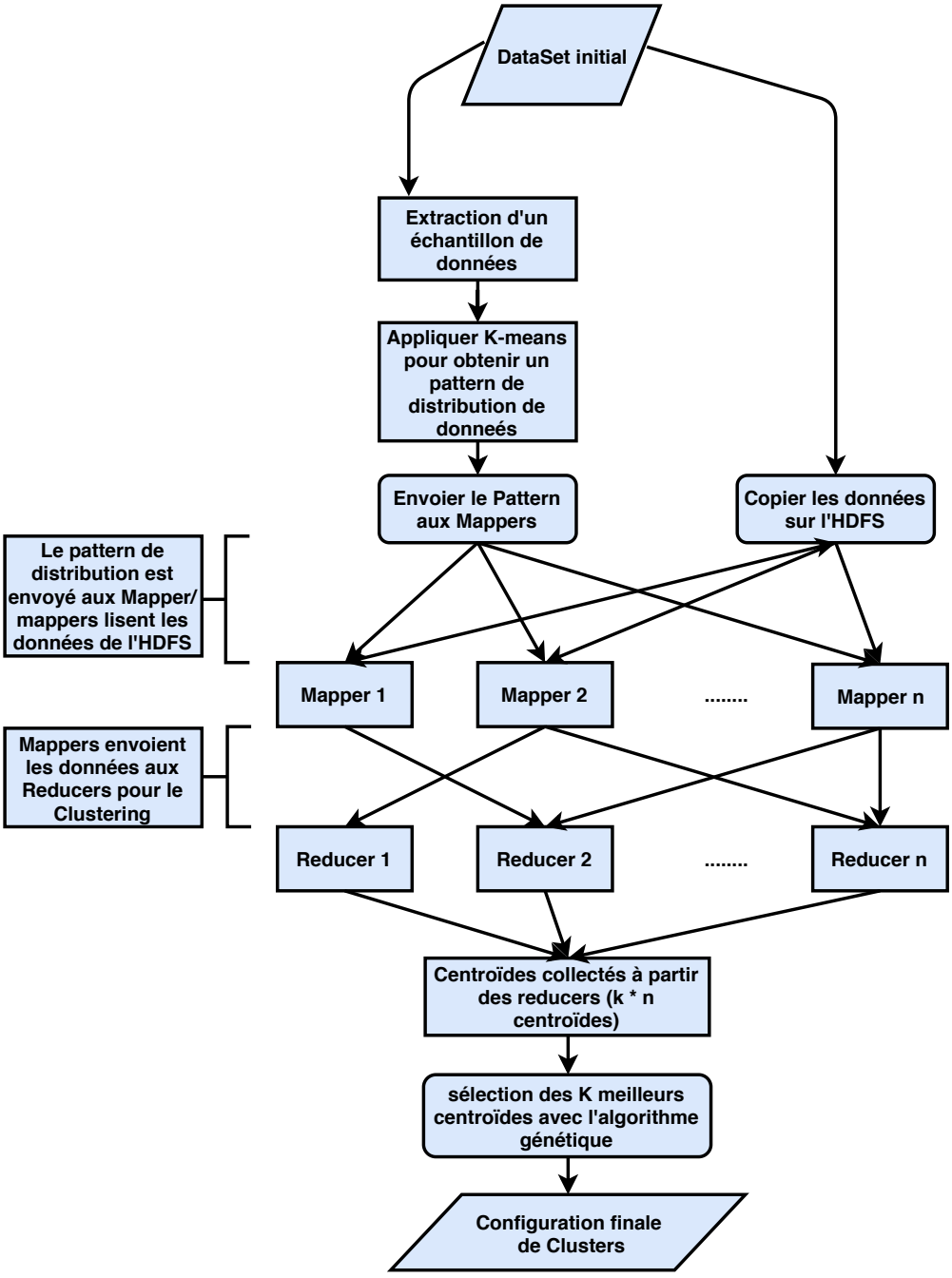


FIG. 5.1 – *Architecture parallèle*

3.2.1 Complexité de la phase de distribution

Dans cette première étape les données sont réparties en N partitions en suivant un modèle de données, où N est le nombre de machines dans le Cluster. Le modèle de données est créé par l'algorithme K-means en utilisant un échantillon extrait à partir des données initiales. La complexité temporelle de cette étape est $O(k' * m * n' * d)$; avec k' le nombre de Clusters qui est égal au nombre de machines utilisées ; m le nombre d'itération et n' la taille de l'échantillon de données avec $n' << n$.

Après la génération du modèle, on l'utilise pour répartir les données du Data Set initial sur l'ensemble des machines. Ainsi, la distance entre chaque point de données et les centroïdes du modèle de données généré est calculée. La complexité de cette étape est $O(k' * n * d)$; où k' est le nombre de Clusters/machines ; n est la taille de l'ensemble de données initiales et d est la dimension des données (nombre d'attributs). Ce processus de distribution est exécuté en parallèle par les Mappers qui sont au nombre de machines utilisées ; la complexité devient alors $O(k' * n * d/nb\_noeud)$. Comme le nombre de Clusters k' dans ce cas est égal au nombre de nœuds, la complexité devient $O(k' * n * d/k') = O(n * d)$. La complexité entière de cette phase devient alors$O(k' * m * n' * d) + O(n * d)$. Supposant que la valeur de $k' * m * n'$ est plus grande que n, la complexité dévient $O(k' * m * n' * d)$.

3.2.2 Complexité de la phase de Clustering

Cette seconde phase prend la majorité du temps d'exécution de la solution proposée, car elle représente le cœur du processus de Clustering. Notre schéma proposé peut être utilisé avec différents algorithmes de Clustering par partitionnement. Donc la complexité de cette étape est déterminée par l'algorithme de Clustering utilisé. Dans cette analyse, nous choisissons notre algorithme SPKmMR que nous avons utilisé comme algorithme principal pour présenter et évaluer notre schéma parallèle.

L'algorithme SPKmMR comprend trois phases principales :

a - Phase d'échantillonnage des données.

b - Phase de sélection des centroïdes initiaux en utilisant l'algorithme PSO.

c - Phase de Clustering final en utilisant K-means avec les centroïdes générés par PSO.

La complexité de la phase d'échantillonnage (a) est $O(k * m * n * d)$, et elle est la même que celle de l'algorithme K-means. Avec k : le nombre de Cluster, m : le nombre d'itérations, n : la taille des données et d : la dimension d'une instance de données.

Pour la phase de raffinement des centroïdes avec l'algorithme PSO (b), les opérations qui consomment le plus de temps d'exécution sont :

- La génération de la population.
- L'évaluation des solutions en utilisant la formule SSW (1.12).

Ainsi la complexité de l'algorithme PSO est définie par les complexités de ces deux opérations.

La complexité de la génération de la population est de l'ordre de $O(d*k*p)$, avec d le nombre d'attributs et k le nombre de Clusters ; $d*k$ représente la dimension du problème pour PSO et p représente la taille de la population.

Dans l'étape de l'évaluation, la fonction fitness assigne chaque point de données au Cluster le plus proche en calculant la distance entre ce dernier avec les centroïdes contenus dans la particule évaluée. Ce processus est appliqué pour chaque particule de l'essaim jusqu'à ce que le nombre maximal d'itérations est atteint. Ainsi la complexité de cette phase est de l'ordre de $O(k*n*d*p*i)$, avec p le nombre de particules et i le nombre d'itérations.

La complexité de l'algorithme PSO sera alors $O(d*k*p)+O(k*n*d*p*i)$. La complexité de la phase de génération de la population sera négligée en comparaison à l'opération d'évaluation. La complexité de l'algorithme PSO sera alors $O(k*n*d*p*i)$.

La dernière étape est l'exécution de l'algorithme K-means en utilisant comme centroïdes de départ ceux obtenus par l'algorithme PSO. Comme mentionné auparavant la complexité de l'algorithme K-means est $O(k*m*d*n)$.

La complexité globale de la phase de Clustering sera alors la somme des complexités de la phase d'échantillonnage, de l'algorithme PSO et de l'algorithme K-means :

$O(k*m*n*d)+O(k*n*d*p*i)+O(k*m*d*n) \longrightarrow O(k*n*d*p*i)+O(k*m*d*n)$. Supposant que$p*i>m$, la complexité devient alors: $O(k*n*d*p*i)$.

Toute cette phase de Clustering est parallélisée en utilisant MapReduce, ainsi la complexité devient $O(k*n*d*p*i/Nb\_noeud)$. Avec cette complexité on peut constater que la complexité temporelle du processus de Clustering diminue linéairement en augmentant le nombre de machines utilisées. Ce constat sera vérifié lors de la partie expérimentale.

3.2.3 Complexité de la phase de fusion des résultats

Dans cette dernière étape l'algorithme génétique est utilisé pour la fusion des résultats. Comme pour l'algorithme PSO, la complexité de l'algorithme génétique est représentée par la complexité de la phase de génération de la population et la phase d'évaluation des solutions (fonction fitness). La complexité de la phase de génération de la population est $O(d*k*p)$,

avec d le nombre d'attributs, k le nombre de Clusters et p le nombre de chromosomes constituant la population.

La fonction fitness utilisée pour l'évaluation des solutions est la même utilisée avec l'algorithme PSO. Par conséquent, la complexité de la phase d'évaluation est : $O(k*n''*d*p*g)$, avec k le nombre de Clusters, n'' la taille de l'échantillon de données utilisées pour l'évaluation des solutions, d est la dimension des données, p est la taille de la population et g est le nombre de générations.

La complexité totale de la phase de fusion sera alors : $O(d*k*p)+O(k*n''*d*p*g)$.

La complexité temporelle de la phase de génération de la population sera négligée par rapport à celle de l'évaluation des résultats. Donc, la complexité de l'algorithme génétique devient $O(k*n''*d*p*g)$.

Nous avons présenté les complexités temporelles des différentes phases de notre schéma parallèle dont la complexité est définie par la somme des complexités de ces dernières que nous notons : $O(k'*m*n'*d)+O(k*n*d*p*i/Nb\_noeud)+O(k*n''*d*p*g)$.

Sachant que $k' << k$ et $n'' << n' << n$, les complexités de la phase de distribution des données et de la phase de fusion des résultats seront négligées par rapport à la complexité de la phase de Clustering. La complexité globale de notre schéma parallèle sera alors définie par :

$$O(k*n*d*p*i/Nb\_noeud)$$

4 Implémentation et expérimentations

Afin de tester et valider notre approche, nous l'avons implémentée avec deux algorithmes de Clustering parallèles : l'algorithme KmeanMR et notre algorithme SPKmMR.

Plusieurs expérimentations ont été conduites afin de déterminer l'efficacité de notre approche sur différents aspects.

Dans un premier temps, nous étudions et analysons l'influence de notre schéma sur les résultats en termes de qualité de Clustering pour les algorithmes parallèles dans l'environnement MapReduce (KmeanMR et SPKmMR).

A travers cette première partie d'expérimentation, nous comparons les résultats obtenus avec différentes techniques de distribution de données combinées avec différentes techniques de fusion de résultats.

Par la suite, nous analysons les performances de notre approche en terme de temps d'exécution en implémentant l'algorithme SPKmMR.

Nous analysons et discutons ensuite la convergence de notre schéma en implémentant aussi l'algorithme SPKmMR.

TAB. 5.1 – *Description des Data Sets*

Data sets	Nombre d'instances	Dimensions
DataSet1	164860	3
DataSet2	2075259	9
DataSet3	989160	3
DataSet4	10376295	9

Dans une étape finale, nous comparons notre solution globale qui comprend notre algorithme SPKmMR utilisé avec une distribution non-aléatoire de données et une fusion de résultats avec l'algorithme génétique, avec de récents algorithmes parallèles de Clustering de données utilisant le Framework MapReduce.

4.1 Environnement de test

Dans les expérimentations de ce chapitre nous avons utilisé un Cluster de 16 machines dont 5 machines physiques et les autres étant des machines virtuelles. Chaque nœud est équipé d'un processeur Dual Core et de 2 Go de mémoire vive. Toutes les machines utilisent le système d'exploitation Linux Ubuntu 10.04. Pour la gestion du Cluster, nous avons utilisé la version 1.2.1 du Framework Hadoop qui est une implémentation open source du Framework MapReduce.

4.2 Les données de tests

Les expérimentations ont été conduites sur les mêmes Data Set utilisés dans le chapitre 4. Un Data Set de données synthétiques (DataSet1) [40], et un Data Set de données réel 'The Individual household electric power consumption' (DataSet2) [59], qui représente les données de consommation d'électricité dans des ménages sur une période temps.

Pour les expérimentations concernant le temps d'exécution, et afin de tester l'évolutivité de notre approche et d'étudier son comportement lors de la variation dans la tailles des données analysées quand ces dernières sont de tailles importantes, nous avons généré deux autres Data Sets qu'on nomme DataSet3 et DataSet4 à partir des Data Sets DataSet1 et DataSet2 respectivement, en dupliquant leurs données multiples fois.

La table (5.1), décrit les dimensions des 4 Data Set utilisés.

4.3 Évaluation de la qualité de Clustering

L'évaluation de notre schéma parallèle consiste à évaluer l'amélioration apportée par ses techniques de distribution des données et de fusion des résultats intermédiaires utilisés. Ces dernières sont comparées avec d'autres techniques de distribution et de fusions.

Pour la distribution des données nous l'avons comparée avec la distribution aléatoire et pour la fusion nous l'avons comparée avec la fusion simple et la fusion par Clustering. Toutes ces techniques sont appliquées aux algorithmes implémentés SPKmMR et KmeansMR. Les expérimentations conduites dans cette section, sont centrées principalement sur l'étape de distribution des données étant donné que dans le chapitre précédent nous avons présenté en détail l'approche de fusion des résultats avec l'algorithme génétique.

Ainsi dans ce qui suit, nous présentons les améliorations apportées par l'utilisation de la distribution non-aléatoire des données sur la qualité du Clustering.

Dans cette étape d'expérimentation nous utilisons les Data Sets DataSet1 et DataSet2. Comme nous utilisons dans notre environnement de test un Cluster de 16 machines, lors de l'étape de la création du pattern de distribution nous définissons la valeur $K = 16$.

Les mêmes modèles de données obtenus depuis chaque Data Set sont utilisés avec les deux algorithmes KmeansMR et SPKmMR.

Les paramètres utilisés pour les algorithmes implémentés sont les mêmes que ceux utilisés dans le chapitre 4 (table 4.1, table 4.2 et table 4.3).

La table 5.2 présente les résultats obtenus par les algorithmes KmeansMR et SPKmMR en utilisant une distribution non-aléatoire avec les différentes techniques de fusion implémentées (simple, Clustering, algorithme génétique). La métrique SSW, qui calcule le ratio de compacité des Clusters est utilisée pour déterminer la qualité des résultats. Cette dernière est calculée en utilisant la formule (1.12).

Discussion des résultats En analysant ces résultats, et en les comparant aux résultats obtenus précédemment dans la table (4.5) où une distribution aléatoire a été utilisée, nous remarquons que ces derniers sont meilleurs que les résultats obtenus en utilisant une distribution non-aléatoire dans tous les cas lors de l'utilisation de la fusion simple ou la fusion avec Clustering et avec les deux Data Set. De plus, à partir de la table (4.4) on peut aussi constater que les algorithmes K-means et PSO séquentiels apportent de meilleurs résultats.

On note que la différence entre les valeurs de fitness obtenues avec la distribution non-aléatoire (cas de la fusion simple ou la fusion avec Clustering) et ceux obtenus avec la distribution aléatoire est très importante, ce qui qualifiera ces résultats de non seulement de moindre qualité mais aussi de résultats imprécis et invalides.

TAB. 5.2 – *Résultats après l'application de la distribution non-aléatoire avec les différentes stratégies de fusion des résultats*

Techniques de fusion	KmeansMR			SPKmMR		
	DataSet1	DataSet2 (k=50)	DataSet2 (k=100)	DataSet1	DataSet2 (k=50)	DataSet2 (k=100)
Fusion simple	0.159748	0.270103	0.277858	0.156695	0.3009458	0.281269
Fusion par K-means	0.101498	0.094805	0.080865	0.097269	0.092251	0.079740
Fusion par AG	0.076442	0.045391	0.037061	0.075649	0.043310	0.034448

Ces résultats sont dus à la méthode utilisée pour la fusion des résultats et non à cause de la technique de répartition des données.

Pour clarifier ce point, nous analysons dans ce qui suit les résultats intermédiaires obtenus au niveau de chaque partition de données (machine).

Dans la table (5.3), nous avons utilisé les résultats intermédiaires obtenus au niveau de chaque partition afin de classifier l'ensemble entier des données. Nous avons pris deux cas d'exemple pour analyser cette étape qui sont : (1) les résultats intermédiaires de l'algorithme SPKmMR avec le Data Set DataSet1 et (2) les résultats intermédiaires de l'algorithme KmeansMR avec le Data Set DataSet2 dans le cas où k=100.

L'analyse des résultats de la table (5.3) nous permettra de soutenir des points cités auparavant comme l'importance de la technique de fusion ou les inconvénients qui peuvent se produire en utilisant une distribution non-aléatoire.

On remarque que tous les résultats intermédiaires du Data Set DataSet1 sont relativement identiques à ceux obtenus après l'utilisation de la fusion simple. Alors que sur le Data Set DataSet2, certaines partitions fournissent des résultats meilleurs que le résultat final après l'application de la fusion simple ; comme avec les partitions P3, P4 et P11 tandis que d'autre partitions comme P1, P2 et P5 procurent des résultats impertinents.

Les résultats impertinents des partitions sur les deux Data Sets sont dus soit à un nombre insuffisant de données au niveau des partitions en question, soit à une non diversité au niveau des points de données, ce qui ne permettra pas de générer une configuration de Clusters valide pour tout l'ensemble de données.

En effet, avec la distribution non-aléatoire les résultats intermédiaires ne sont pas destinés à être des configurations pertinentes pour tout l'ensemble de données vu que chaque partition contient un ensemble similaire de points de données à partir desquels on ne peut pas générer

TAB. 5.3 – *Résultats intermédiaires*

Partitions	DataSet1	DataSet2
P1	0.159275	0.134110
P2	0.136808	0.111946
P3	0.123890	0.077897
P4	0.149114	0.043421
P5	0.123619	0.129709
P6	0.141003	0.074752
P7	0.134218	0.076125
P8	0.146532	0.057816
P9	0.146205	0.185012
P10	0.121721	0.070572
P11	0.150959	0.062136
P12	0.176330	0.122667
P13	0.155793	0.245486
P14	0.129691	0.105169
P15	0.155635	0.054617
P16	0.134052	0.125209

une configuration de Clusters valide pour tout l'ensemble de données. Les cas des partitions P3, P4 et P11...Etc. du Data Set DataSet2 sont considérés comme des cas exceptionnels. Comme le Data Set DataSet2 est d'une taille très importante, ces partitions ont pu bénéficier d'un nombre suffisant et diversifié de points de donnés ce qui s'est traduit par des résultats pertinents. Malgré la présence de certains résultats intermédiaires pertinents, le résultat final après l'application de la technique de fusion simple reste impertinent due à la stratégie suivie par cette technique qui consiste à combiner tous les centroïdes intermédiaires sans sélection particulière.

Concernant la fusion avec Clustering, cette technique a amélioré les valeurs de fitness par rapport à la fusion simple pour tous les Data Sets et les algorithmes utilisés, mais ces résultats demeurent invalides. De plus, les résultats intermédiaires des partitions P3, P4 et P11... etc.

restent meilleurs que le résultat final après la fusion. Ainsi, comme pour la fusion simple, cette technique ne réussit pas aussi à exploiter les résultats intermédiaires en impliquant tous ces derniers pour la génération de la solution finale.

Contrairement aux résultats discutés jusqu'à présent, on peut voir sur la table (5.2) que la fusion avec algorithme génétique fournit les meilleurs résultats en choisissant les meilleurs centroïdes à partir des différents résultats intermédiaires qui, selon la table (5.3), paraissaient impertinents. En comparant les résultats de la table (5.2) avec les résultats du chapitre 4, on voit que les meilleures valeurs de fitness atteintes sont obtenues lors de l'utilisation de la distribution non aléatoire combinée avec la fusion par algorithme génétique en utilisant l'algorithme SPKmMR. Les meilleures valeurs obtenues par l'algorithme KmeansMR sont aussi obtenues en combinant la distribution non-aléatoire avec la fusion par AG.

Un autre point important à mentionner et que l'algorithme KmeansMR avec notre schéma parallèle apporte de meilleurs résultats que l'algorithme SPKM séquentiel ainsi que l'algorithme SPKmMR en utilisant une distribution aléatoire avec les différentes techniques de fusion utilisées.

On note aussi que les améliorations apportées par l'utilisation de notre schéma parallèle sont plus importantes sur le Data Set DataSet2 que sur le Data Set DataSet1. Ceci peut être justifié par la différence entre la taille des deux Data Sets. Le Data Set DataSet1 est relativement plus petit que le Data Set DataSet2, ainsi, l'optimisation des résultats en appliquant le processus de distribution de données n'est pas autant visible sur le Data Set DataSet1 que sur le Data Set DataSet2. Car, sur le Data Set DataSet1 les algorithmes utilisés sont moins affectés par le problème de convergence sur les données de grandes tailles.

Nous résumons les remarques les plus importantes retenues de cette analyse dans les points suivants :

- L'utilisation de la distribution non-aléatoire génère des partitions de données non diversifiées qui ne sont pas valides pour l'utilisation directe.
- La limite des techniques conventionnelles de fusion de résultats tels que la fusion simple ou par Clustering était plus visible lors de l'utilisation de la distribution non aléatoire, ce qui soutient les remarques citées dans le chapitre 4 concernant ces techniques de fusion.
- La fusion avec AG a montré son habilité pour la génération d'une solution finale pertinente, même à partir d'un ensemble de résultats intermédiaires qui jusqu'alors étaient considérés invalides. Ceci nous a permis de souligner l'avantage de la distribution non-aléatoire.
- La combinaison entre la distribution non-aléatoire et la fusion par AG, ce qui représente le principe de notre schéma proposé, apporte les meilleurs résultats.

TAB. 5.4 – *Temps d'exécution en minutes sur DataSet2 (K=100)*

Algorithmes	Temps d'exécution (Minutes)
K-means	18.46
SPKM	28.07
KmeansMR	7.11
SPKmMR	7.61

4.4 Évaluation du temps d'exécution

Dans cette section, nous analysons les performances de notre schéma parallèle en terme de temps d'exécution. Plusieurs tests couvrant différents aspects ont été réalisés. Dans un premier temps nous comparons le temps d'exécution entre une exécution séquentielle et une exécution parallèle. Dans un second temps, nous analysons le comportement de l'algorithme SPKmMR en utilisant notre schéma tout en variant le nombre de machines utilisées. Dans ce test, l'algorithme est évalué en utilisant la métrique « SpeedUp ». Enfin, l'efficacité du schéma proposé est testée en utilisant la métrique « Efficiency ».

4.4.1 Expérimentation 1: Comparaison entre l'exécution séquentielle et l'exécution parallèle (Temps d'exécution)

Dans cette partie des expérimentations, nous comparons le temps d'exécution des algorithmes KmeansMR et SPKmMR en utilisant notre schéma parallèle, avec leurs versions séquentielles respectives K-means et SPKM.

Dans ce test, nous utilisons le Data Set DataSet2 avec la valeur $k = 100$ avec les différents algorithmes implémentés. La table (5.4) représente les résultats de temps d'exécution obtenus exprimés en minutes.

D'après ces résultats, on peut voir que les algorithmes parallèles surpassent leurs versions séquentielles. On remarque que l'algorithme KmeansMR est 61% plus rapide que son implémentation séquentielle (K-means). Pour l'algorithme SPKmMR, ce dernier a bénéficié d'un temps d'exécution 71% plus rapide par rapport à l'algorithme SPKM.

Comme on a pu le constater, l'amélioration apportée par l'algorithme SPKmMR par rapport à sa version séquentielle est plus importante que celle apportée par l'algorithme KmeansMR par rapport à sa version séquentielle. Ceci peut être expliqué par la sensibilité de l'algorithme PSO aux données volumineuses. Car, avec la version séquentielle tout

l'ensemble de données a été analysé simultanément ce qui résultera en un temps d'exécution (28,07 minutes) beaucoup plus important que celui de l'algorithme K-means (18,46 minutes). Mais, avec l'exécution parallèle qui consiste à répartir les données en plusieurs partitions, l'algorithme PSO sur chaque machine analysera une quantité de données beaucoup moins volumineuse, ce qui s'est traduit par un temps d'exécution de l'algorithme SPKmMR (7,61 minutes) très proche de celui de KmeansMR (7,11 minutes).

4.4.2 Expérimentation 2: Étude du temps d'exécution et du SpeedUp avec la variation du nombre de machines utilisées

Dans cette étape nous étudions les performances de notre approche en termes de temps d'exécution et de SpeedUp tout en variant le nombre de machines utilisées dans le traitement. Le nombre de machines utilisées sera de 2, 4, 8 et 16.

Les expérimentations conduites dans cette section sont effectuées avec l'algorithme SPKmMR, qui selon les précédentes expérimentations, proposait les meilleurs résultats en terme de qualité de Clustering.

Les tests seront conduits sur tous les 4 Data Sets cités sur la table (5.1) afin d'observer le comportement de notre approche avec des données de tailles différentes.

Pour le calcul du SpeedUp, nous utilisons la formule (2.1). Le SpeedUp nous permettra de déterminer le ratio de rapidité d'un algorithme parallèle par rapport à sa version séquentielle ou sa version parallèle en utilisant un nombre réduit de machines par rapport à son exécution actuelle.

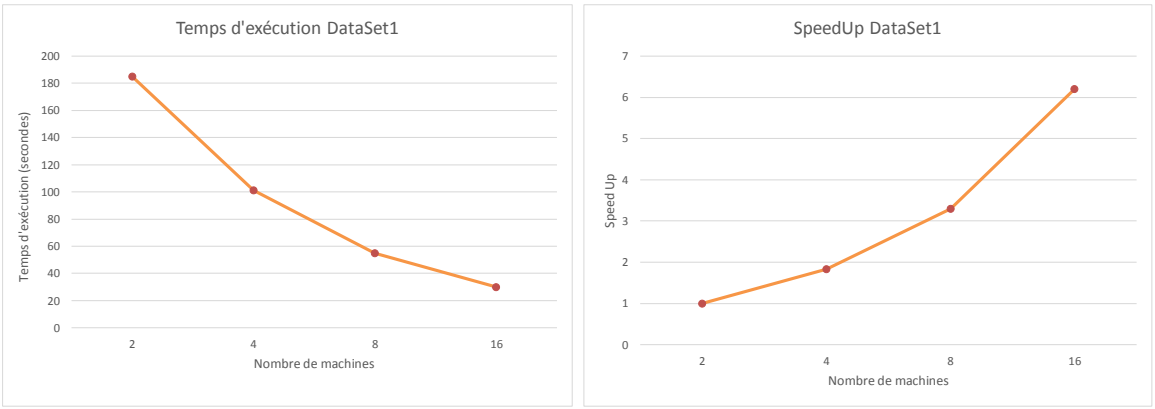
Pour le calcul du SpeedUp, dans notre cas, on considère comme point de départ le cas où le nombre de machines est égal à 2. Ce choix est pris parce que dans le cas où on utilise une seule machine, l'étape de distribution de données ne pourra pas avoir lieu. Ainsi, la valeur de SpeedUp dans le cas où on utilise deux machines sera égale à 1.

On note que durant ces expérimentations, le temps d'exécution de l'étape de fusion en utilisant l'algorithme génétique n'est pas incluse avec le temps d'exécution total, puisque cette dernière est exécutée sur une seule machine séquentiellement, et elle requière un temps d'exécution presque similaire pour tous les cas étudiés.

Dans ce qui suit, nous présenterons les résultats du temps d'exécution et de SpeedUp pour chaque Data Set. Les résultats sont représentés sur les figures 5.2,5.3,5.4 et 5.5.

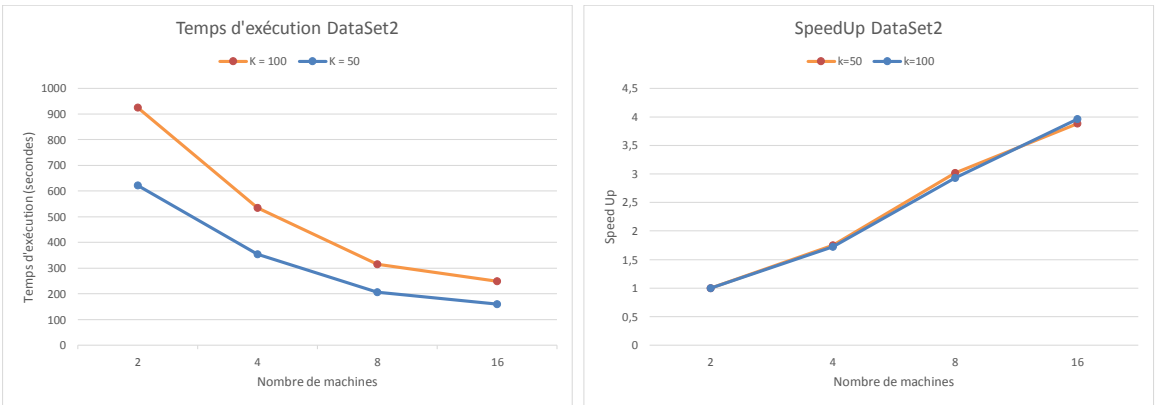
D'après ces figures, on peut constater que sur tous les Data Sets, le temps d'exécution diminue linéairement, de même, les valeurs de SpeedUp augmente linéairement, ce qui correspond à la complexité théorique présentée auparavant.

On peut observer aussi que les valeurs de SpeedUp diffèrent selon la taille du Data Set utilisé. Les valeurs de SpeedUp obtenues sur le Data Set DataSet3 sont plus importantes que



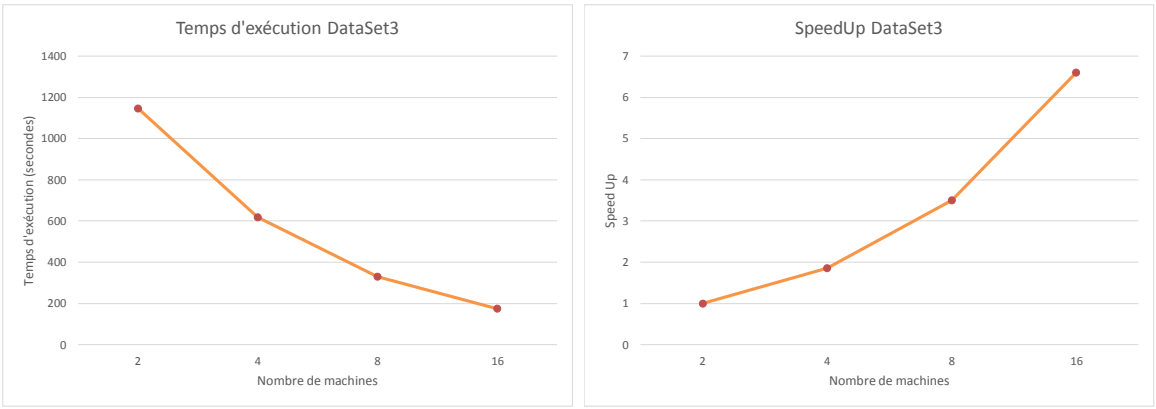
(a) *Temps d'exécution.* (b) *SpeedUp.*

FIG. 5.2 – *Temps d'exécution et SpeedUp avec la variation du nombre de machines utilisées sur le Data Set DataSet1*



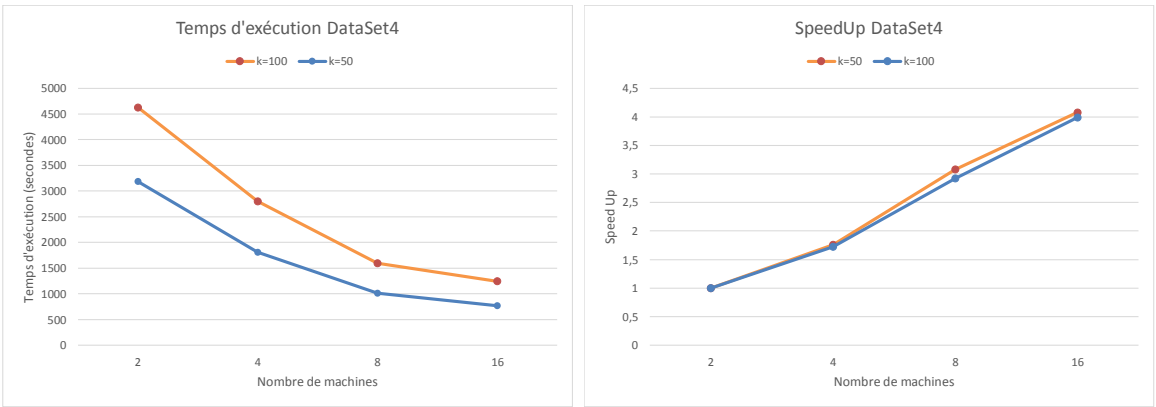
(a) *Temps d'exécution.* (b) *SpeedUp.*

FIG. 5.3 – *Temps d'exécution et SpeedUp avec la variation du nombre de machines utilisées sur le Data Set DataSet2*



(a) *Temps d'exécution.* (b) *SpeedUp.*

FIG. 5.4 – *Temps d'exécution et SpeedUp avec la variation du nombre de machines utilisées sur le Data Set DataSet3*



(a) *Temps d'exécution.* (b) *SpeedUp.*

FIG. 5.5 – *Temps d'exécution et SpeedUp avec la variation du nombre de machines utilisées sur le Data Set DataSet4*

celles obtenues avec le Data Set DataSet1. De même, les valeurs de SpeedUp du Data Set DataSet4 sont aussi plus importantes que celle du Data Set DataSet2. Ainsi, plus la taille du Data Set utilisé est importante plus les valeurs de SpeedUp sont importantes. Ceci est expliqué par le fait que l'utilisation d'un Data Set d'une taille volumineuse générera un temps d'exécution très important en utilisant un algorithme séquentiel ou un algorithme parallèle avec un nombre réduite de machines (2 machines dans notre cas).

On conclut alors que l'avantage de l'utilisation d'une implémentation parallèle est plus visible lorsque la taille des données analysées est plus importante. Mais, si on compare les valeurs de SpeedUp du Data Set DataSet1 avec celle du Data Set DataSet3, on remarque que celles du Data Set DataSet1 sont plus importantes malgré que le Data Set DataSet1 est moins volumineux que le Data Set DataSet3 ce qui contredit le constat précédent. Ce constat peut aussi être observé entre les Data Sets DataSet2 et DataSet4. Ces résultats inattendus sont dus à la technique non-aléatoire utilisée pour la distribution de données. En effet, avec notre stratégie de distribution, les partitions de données générées n'ont pas forcément des tailles identiques contrairement à la distribution aléatoire où dans la majorité des cas les partitions générées sont de tailles similaires. Ainsi, avec notre stratégie la valeur du SpeedUp dépendra de la nature du Data Set analysé et de ses partitions.

Si on prend l'exemple du Data Set DataSet1, la plus petite partition générée est composée de 4072 instances de données, et la plus large des partitions contient 20445 instances de données, ce qui représente approximativement 5 fois la taille de la plus petite partition. Donc, avec les 16 machines utilisées, le temps d'exécution global de la phase Reduce (Clustering) sera représenté par le temps d'exécution requis par la machine traitant la plus grande partition de données. Ceci ralentira le temps d'exécution global et influencera négativement sur les valeurs de SpeedUp. Comme pour le Data Set DataSet1, le Data Set DataSet2 présente le même cas pour la taille des partitions de données. La plus petite des partitions est composée de 27933 instances de données, et la plus grande partition est 10 fois plus importante avec 280809 instances de données, ce qui s'est traduit par des valeurs de SpeedUp moins performantes que celle du Data Set DataSet1 comme on l'a déjà distingué sur les figures.

Afin de mieux visualiser ces cas, nous calculons dans ce qui suit les valeurs d'Efficacité (Efficiency) de notre algorithme avec les différents Data Sets utilisés. Cette mesure est calculée en utilisant la formule (2.2). Comme l'Efficacité détermine le taux d'exploitation des machines (ressources) utilisées par un algorithme parallèle, cela nous permettra de déterminer l'impact du déséquilibre entre les tailles des partitions sur les performances de notre algorithme en terme de temps d'exécution.

La table (5.5) représente les résultats d'Efficacité obtenus, en utilisant l'algorithme SPKmMR avec notre schéma parallèle sur les 4 Data Sets.

TAB. 5.5 – *Valeurs d'Efficacité sur les différents Data Sets*

Data sets	Efficacité
DataSet1	0,770
DataSet2	k=50: 0,485 \ K=100: 0,496
DataSet3	0,823
DataSet4	k=50: 0,510 \ K=100: 0,499

A partir des résultats de la table (5.5), en comparant les valeurs d'Efficacité du Data Set DataSet1 avec celles du Data Set DataSet3, on remarque une nette amélioration dans les résultats vu que le Data Set DataSet3 est une duplication du Data Set DataSet1 et sa taille est supérieure à ce dernier. Pour le Data Set DataSet4, en le comparant avec le Data Set DataSet2, seule une légère amélioration est constatée, ce qui est du au problème des tailles des partitions de données cité auparavant.

La valeur d'Efficacité théorique maximal est de 1, où aucune perte de performance n'est enregistrée. Or, la meilleure valeur obtenue parmi les différents Data Set est de 0.823, et la plus mauvaise valeur est de 0.485. Ces valeurs d'Efficacité soutiennent les remarques citées auparavant à propos de l'influence des tailles des partitions sur le SpeedUp.

En dehors des pertes de performances dues au transfert d'informations via le réseau du Framework MapReduce. Ces pertes de performances enregistrées sont dues aussi au fait que lors de la phase Reduce et pendant le traitement de la plus grande partition par sa machine allouée. Les autres machines traitant les partitions moins volumineuses et après avoir terminer leurs traitements, entreront dans un état d'attente jusqu'à la fin du traitement de la plus grande partition, ce qui ne représente pas une exploitation optimale des ressources disponibles.

4.5 Analyse de la convergence

Dans cette section nous analysons et discutons la convergence de notre schéma parallèle implémenté avec l'algorithme SPKmMR. Lors de cette analyse, nous présenterons les résultats de convergence du processus global après application de la phase de fusion et nous présenterons aussi les résultats de convergence en utilisant les résultats intermédiaires des différentes partitions. Nous comparons aussi ces résultats avec ceux de la convergence de l'algorithme SPKM séquentiel.

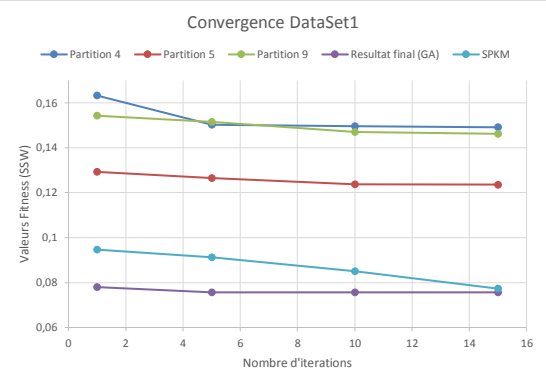


FIG. 5.6 – *Convergence sur le Data Set DataSet1*

Dans cette analyse les expérimentations ont été conduites sur les Data Sets : DataSet1 et DataSet2 (cas où $K = 50$).

La figure (5.6) représente le graphe de convergence sur le Data Set DataSet1.

Comme la plupart des partitions de données présentent presque la même courbe de convergence, nous avons choisi de présenter 3 partitions seulement avec des tailles différentes:

1. La première partition avec 4072 instances de données (P4).
2. La deuxième partition avec 20445 instances de données (P5).
3. La troisième partition avec 12359 instances de données (P9).

A partir des résultats présentés sur le graphe de la figure (5.6), on remarque que lors de l'utilisation de tout l'ensemble des données avec l'algorithme séquentiel SPKM, ce dernier ne converge pas jusqu'au moment d'atteindre la 15^{eme} itération, tandis que les résultats intermédiaires des différentes partitions, convergent tous avant la 10^{eme} itération, mais sans atteindre des solutions valides. La plus petite partition $P4$ converge dans 5 itérations due à sa petite taille. Les partitions $P5$ et $P9$, qui ont un plus grand nombre d'instances de données par rapport à $P4$, requièrent 10 itérations pour atteindre des résultats stables. Concernant la convergence du résultat final, après l'application du processus de fusion par algorithme génétique, il requerra seulement 5 itérations.

On note que les résultats de convergence finaux concernant l'utilisation de la fusion par algorithme génétique à une certaine itération, représentent la fusion des résultats intermédiaires obtenus à cette itération précise sur chaque machine et non pas les résultats obtenus durant les itérations de l'exécution de l'algorithme génétique. Cette remarque sera valable aussi pour les résultats de convergence pour le Data Set DataSet2 que nous présenterons par la suite.

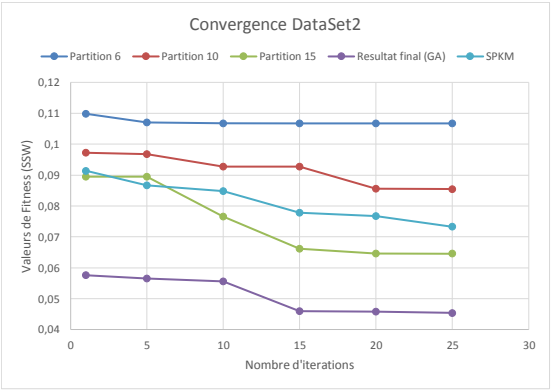


FIG. 5.7 – *Convergence sur le Data Set DataSet2*

La figure (5.7), illustre les graphes de convergences obtenus sur le Data Set DataSet2. Comme pour le Data Set DataSet1, concernant les résultats intermédiaires, nous avons sélectionné seulement 3 partitions de données à représenter sur le graphe :

1. La première partition avec 48575 instances de données (P6).
2. La deuxième partition avec 172927 instances de données (P10).
3. La troisième partition avec 280809 instances de données (P15).

A partir des résultats présentés sur la figure (5.7), on remarque que l'algorithme séquentiel SPKM requière 25 itérations afin d'atteindre un état stable puisqu'il utilise tout l'ensemble des données du Data Set. Contrairement à ce dernier, avec la partition $P6$ seules 5 itérations sont nécessaires avant de converger vu que cette dernière a un nombre réduit d'instances de données. Pour les partitions $P10$ et $P15$, comme ces dernières ont un nombre plus important d'instances de données, elles ne convergent qu'après la 15^{eme} itération. On distingue aussi que les résultats finaux après l'application de la fusion se stabilisent après la 15^{eme} itération.

A partir des résultats obtenus sur les deux Data Sets analysés, on remarque qu'en partitionnant les données en des sous-ensembles moins volumineux, le nombre d'itérations requises pour atteindre des résultats finaux est moins important que lors de l'utilisation des ensembles des données entiers, ce qui représente un gain en performance en terme de temps d'exécution en plus des performances gagnées en utilisant les ressources des machines du Cluster MapReduce.

4.6 Comparaison des algorithmes

Dans cette dernière partie des expérimentations, nous comparons notre approche parallèle avec des algorithmes parallèles de Clustering de données récemment proposés. Les comparai-

TAB. 5.6 – *Paramètres Parallel KPSO*

Paramètres	Valeurs	
	DataSet1 / DataSet3	DataSet2 / DataSet4
Inertia factor	0.5	0.3
Coefficient de confiance dans la position courante	0.1	0.72
Coefficient de confiance voisinage	3.0	1.49
Nombre de Clusters	11	50
Nombre d'itérations (K-means)	15	25
Nombre de nœuds	16	

sons réalisées dans cette partie concernent la qualité du Clustering et l'Efficacité (Efficiency) des algorithmes implémentés.

Nous avons comparé notre schéma parallèle en implémentant l'algorithme SPKmMR avec les algorithmes suivants:

- Parallel K-PSO based on MapReduce (Parallel KPSO) [85] qui est un algorithme parallèle basé sur le Framework MapReduce, hybride entre l'algorithme PSO et l'algorithme K-means.
- Optimized big Data K-means Clustering using MapReduce (Opt MR-Kmeans) [25] qui est une solution parallèle pour l'algorithme K-means basée sur MapReduce, optimisée pour le traitement des données très volumineuses.

En plus des algorithmes parallèles implémentés, nous avons inclus l'algorithme K-means séquentiel pour mieux situer les performances des algorithmes testés.

Les évaluations ont été réalisées en utilisant les 4 Data Sets présentés sur la table (5.1).

Les tables 5.6 et 5.7 représentent les paramètres des algorithmes «Parallel KPSO » et « Opt MR-Kmeans » respectivement.

4.6.1 Comparaison de la qualité de Clustering

Dans un premier temps, nous avons évalué la qualité de Clustering à l'aide de deux métriques différentes: SSW en utilisant la formule (1.12) et DBI en utilisant la formule (1.14). Cette dernière métrique nous permettra de calculer le ratio de similarité des points de données

TAB. 5.7 – *Paramètres Opt MR-Kmeans*

Paramètres	Valeurs	
	DataSet1/DataSet3	DataSet2/DataSet4
Nombre de Clusters	11	50
Nombre d'itérations	15	25
Nombre de nœuds	16	

TAB. 5.8 – *Validation du Clustering*

	K-means		Parallel KPSO		Opt MR Kmeans		Notre Approche	
Data sets	DBI	SSW	DBI	SSW	DBI	SSW	DBI	SSW
DataSet1	1.035278	0.079604	0.970420	0.077609	1.030236	0.079442	**0.940197**	**0.075649**
DataSet2	1.19255	0.079233	1.165538	0.074396	1.204050	0.080650	**1.057580**	**0.043310**
DataSet3	1.031842	0.079399	1.017055	0.077940	1.022694	0.079817	**0.946913**	**0.075928**
DataSet4	1.196346	0.080747	1.169648	0.075900	1.192127	0.080011	**1.065280**	**0.045710**

inter-Cluster à celui de la dissimilarité des points de données intra-Clusters. L'utilisation de deux métriques différentes nous permettra de mieux valider nos résultats en évaluant différents critères.

Les résultats de qualité de Clustering sont résumés dans la table (5.8).

A partir des résultats de la table (5.8), on peut voir que notre approche apporte les meilleurs résultats comparée à tous les autres algorithmes implémentés sur tous les Data Sets et avec les deux différentes métriques d'évaluation. On remarque aussi, qu'en comparant les résultats obtenus à partir du Data Set DataSet1 avec ceux du Data Set DataSet3, la différence dans la qualité du Clustering est minime malgré la grande différence entre les tailles des Data Sets. Cette même remarque peut être observée en comparant les résultats obtenus à partir du Data Set DataSet2 avec ceux du Data Set DataSet4. Ces constats démontrent l'évolutivité de notre solution où l'augmentation dans les tailles des données analysées ne démunie pas la qualité des résultats.

TAB. 5.9 – *Valeurs d'Efficacité des algorithmes sur le Data Set DataSet4*

Algorithmes	Efficacité
Parallel KPSO	0.6
Opt MR Kmeans	0,521
Notre approche	0,510

4.6.2 Comparaison de l'Efficacité

Dans une dernière expérimentation, nous comparons la valeur d'Efficacité de notre solution avec celles des autres algorithmes parallèles implémentés.

Nous avons choisi de comparer les valeurs d'Efficacité au lieu de comparer le temps d'exécution, car dans le cas où les algorithmes comparés ne suivent pas le même déroulement et n'utilisent pas les même architectures, les valeurs de temps d'exécution peuvent ne pas être significatives et représentatives. Par exemple l'algorithme « Parallel KPSO » commence par une étape de raffinement des centroïdes à l'aide de l'algorithme PSO, qui, n'est pas présente dans l'algorithme « Opt MR kmeans ». Ainsi ce dernier aura forcément un temps d'exécution plus réduit.

D'après la table (5.5), les valeurs d'Efficacité les moins optimales ont été obtenues sur le Data Set DataSet4, ainsi nous utilisons ce dernier lors de la comparaison des algorithmes afin d'éviter le cas du meilleur scénario.

La table (5.9) résume ainsi les résultats d'Efficacité obtenus par les différents algorithmes sur le Data Set DataSet4.

A partir des résultats présentés sur la table (5.9), on observe que notre approche, et en dehors d'un scénario optimal, propose une Efficacité acceptable et compétitive par rapport aux autres algorithmes. De plus, l'Efficacité enregistrée par notre algorithme est due à la nature des données utilisées comme on a pu l'observer sur la table (5.5), contrairement à l'algorithme « Parallel KPSO » où la perte d'efficacité est due à l'architecture «parallélisme par tâches » utilisée, ce qui ne changera pas avec d'autre Data Sets. Ceci s'applique aussi pour l'algorithme « Opt MR Kmeans » où un processus d'échantillonnage est appliqué aux données en utilisant l'architecture «parallélisme indépendant », ce qui a causé une réduction dans les valeurs d'Efficacité.

5 L'application de notre solution au problème de détection des communautés

Afin de démontrer l'efficacité et la pertinence des résultats obtenus par notre approche, nous l'avons appliquée au problème de détection de communautés sur le réseau social Bibsonomy [11].

Le problème de détection de communautés consiste à découvrir un ensemble d'utilisateurs qui partagent des points d'intérêt en commun sur un réseau social. Une des solutions à ce problème est l'utilisation des algorithmes de Clustering, où chaque Cluster contient un ensemble d'utilisateurs communs potentiels.

Dans [75], les auteurs ont proposé une solution basée sur l'algorithme K-means afin de découvrir les communautés d'utilisateurs sur le réseau social académique Bibsonomy.

Afin de réaliser cette solution, la similarité entre les différents utilisateurs est calculée. Ensuite, ces valeurs de similarité sont utilisées afin de créer une matrice de distances. Où chaque élément de cette dernière représente la distance ou la similarité entre deux utilisateurs du réseau social.

La similarité entre deux utilisateurs est calculé en utilisant la formule (5.1).

$$Sim\left(User_i,User_i\right) = \left|tags_{Ui} \cap tags_{Uj}\right| / \left|tags_{Ui} \cup tags_{Uj}\right| \tag{5.1}$$

Où $tags_{Ui}$ et $tags_{Uj}$ représentent les tags utilisés par les utilisateurs $user_i$ et $user_j$ respectivement afin d'annoter leurs documents (bookmarks).

La matrice des distances est créé en utilisant la formule (5.2) suivante:

$$Mat\_dist\left[i,j\right] = \begin{cases} 1/Sim\left(User_i,User_j\right), & si\ Sim\left(User_i,User_j\right) > 0 \\ 100000, si\ Sim\left(User_i,User_j\right) = 0 \\ 0,\ si\ i = j \end{cases} \tag{5.2}$$

Où $Mat\_dist[i,j]$ est la distance entre les utilisateurs $user_i$ et $user_j$. Ainsi, chaque ligne i à partir de la matrice des distances représente les valeurs de distances entre l'utilisateur $user_i$ et tous les autres utilisateurs de la collection.

La matrice des distances sera utilisée comme données d'entrée pour l'algorithme de Clustering utilisé pour cette tâche, ce qui nous permettra de découvrir les communautés d'utilisateurs.

5.1 Expérimentations

Dans cette section, nous présenterons les détails des expérimentations réalisées afin d'appliquer notre approche au problème de détection de communautés sur le réseau Bibsonomy.

5.1.1 Le Data Set Bibsonomy

Afin d'appliquer notre approche à ce problème nous avons utilisé la version « 2016-10-10 » du Data Set Bibsonomy [1]. Ce Data Set contient trois fichiers différents : Tas, Bookmarks et le fichier Bibtex. Dans nos expérimentations nous utilisons seulement le fichier « Tas » qui contient les tags utilisés par chaque utilisateur afin d'annoter ces propres bookmarks.

Les informations contenues dans le fichier « tas » seront utilisées pour calculer la similarité entre les utilisateurs et par la suite créer la matrice des distances. A partir du fichier « tas » nous avons utilisé un échantillon de 1078 utilisateurs, ce qui a résulté en une matrice de dimensions $1078x1078$.

5.2 L'évaluation des solutions

Afin d'évaluer les résultats obtenus par notre approche sur ce problème, nous utilisons dans un premier lieu la métrique SSW en utilisant la formule (1.12). Pour examiner la pertinence des résultats obtenus nous utilisons l'évaluation dite « human evaluation » sur les Clusters générés. Cette dernière évaluation consiste à analyser chaque Cluster afin de déterminer quels et combien d'utilisateurs sont réellement similaires selon leurs tags utilisés. Avec cette méthode nous pouvons vérifier la pertinence entre les utilisateurs groupés dans un même Cluster. Pour cela nous réutilisons le fichier « tas » qui contient les tags utilisés par chaque utilisateur. Quand nous comparons deux configurations de Clustering, la meilleure solution sera celle avec le plus grand nombre d'utilisateurs communs trouvés [75].

5.3 Expérimentations et résultats

Durant cette partie d'expérimentation, nous avons implémenté le problème de détection des communautés avec l'algorithme SPKmMR en utilisant notre schéma parallèle proposé, et nous l'avons comparé avec les algorithmes K-means, Opt-MR-Kmeans [25] et Parallel KPSO [85].

Plusieurs tests ont été conduits sur la matrice des distances afin de déterminer le nombre de Clusters adéquats. Deux valeurs différentes ont été retenues $K = 10$ et $K = 20$.

TAB. 5.10 – *Résultats du Clustering (SSW)*

Algorithmes	K=10	K=20
K-means	6.8506	6.6333
Opt-Kmeans	6.8963	6.6205
Parallel KPSO	6.5692	6.3461
Notre approche	5.8317	5.2788

Dans la première partie d'expérimentations, nous appliquons tous les algorithmes implémentés sur la matrice des distances et évaluons la qualité des Clusters obtenus en utilisant la métrique SSW 1.12.

La table (5.10) représente les résultats obtenus en terme de qualité de Clustering en utilisant la métrique SSW avec les différents algorithmes implémentés et les différentes valeurs de K utilisées.

A partir des résultats de la table (5.10), on peut voir que l'algorithme «Opt-MR-Kmeans» fournit des résultats presque similaires à ceux obtenus par l'algorithme K-means séquentiel. Quant à l'algorithme «Parallel KPSO», il apporte des résultats légèrement supérieurs à ceux obtenus par les algorithmes K-means et «Opt-MR-Kmeans». Quant à notre approche, elle apporte les meilleurs résultats comparée à tous les trois autres algorithmes et avec les deux différentes valeurs de K.

Chaque Cluster représente une communauté d'utilisateurs potentielle, mais les utilisateurs classifiés dans un même Cluster ne sont pas forcément tous similaires. Pour cela, dans une seconde partie des expérimentations, nous analysons les configurations de Clusters obtenues dans l'expérimentation précédente afin de déterminer la pertinence réelle des résultats obtenus. Ceci est réalisé en analysant nous même les Clusters obtenus en calculant le nombre réel d'utilisateurs similaires sur chaque Cluster.

Le calcul des utilisateurs similaires se fait en comparant le nombre de tags communs utilisés par ces derniers en utilisant les informations contenues dans le fichier « tas ».

Durant cette expérimentation, nous avons analysé les résultats obtenus par les algorithmes K-means, « Parallel KPSO » et notre approche. L'algorithme « Opt-MR-Kmeans » n'a pas été analysé, étant donné que selon la table (5.10), ce dernier présente presque les mêmes résultats que l'algorithme K-means. Nous avons choisi d'analyser le cas où $K = 10$.

La table (5.11) présente les résultats obtenus en terme de nombre d'utilisateurs similaires au niveau de chaque Cluster en utilisant les trois algorithmes cité dessus.

TAB. 5.11 – *Nombre d'utilisateurs similaires sur chaque Cluster*

Algorithmes	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Total
K-means	10	12	14	37	34	7	61	18	101	9	**303**
Parallel KPSO	45	9	8	16	41	42	101	20	58	17	**357**
Notre approche	5	26	63	20	52	31	16	77	66	30	**386**

D'après les résultats présentés sur la table (5.11), on peut distinguer encore la qualité et la supériorité des résultats obtenus par notre approche. Le nombre total d'utilisateurs similaires trouvés par notre approche est supérieur à ceux obtenus en utilisant les algorithmes K-means et « Parallel KSPO », ce qui prouve l'efficacité de notre approche.

L'utilisation de « human evaluation » a confirmé les résultats obtenus en analysant les Clusters à l'aide de la métrique SSW sur la table (5.10).

Ainsi, d'après tous les résultats obtenus dans cette section, on peut conclure que notre approche fournit des résultats pertinents et cohérents pour la résolution des problèmes réels de Clustering de données.

6 Conclusion

Dans ce chapitre nous avons proposé un schéma parallèle pour les algorithmes de Clustering par partitionnement basés sur l'environnement MapReduce.

L'objectif de notre approche était d'améliorer les résultats du Clustering dans un environnement parallèle en améliorant la phase de distribution des données sur les nœuds du Cluster et en exploitant les résultats intermédiaires obtenus sur chaque nœud en utilisant une stratégie de fusion de résultats basée sur l'algorithme génétique.

L'approche proposée a été validée et testée avec deux algorithmes de Clustering parallèles : SPKmMR et KmeanMR.

Avec l'utilisation de notre approche, les deux algorithmes implémentés ont vu leurs performances améliorées en terme de qualité de Clustering en les comparant à leur exécution en utilisant des techniques conventionnelles de distribution des données et de fusion des résultats.

L'algorithme SPKmMR a été aussi testé avec quatre Data Sets de différentes tailles afin d'examiner l'évolutivité de notre solution proposée. De plus, il a été comparé avec d'autre algorithmes de Clustering parallèles récemment proposé sur les quatre Data Sets. Les

résultats des expérimentations ont montré que l'algorithme SPKmMR avec notre schéma proposé a surpassé les algorithmes « Opt-MR-Kmeans » [25] et « Parallel KPSO » [85] en terme de qualité de Clustering et a fourni des valeurs d'Efficacité compétitives dans son pire scénario.

Dans une étape d'expérimentation finale, notre approche a été appliquée au problème de détection des communautés d'utilisateurs sur le réseau social Bibsonomy, et elle a été comparée avec les algorithmes K-means et «Parallel KPSO». Encore une fois, les résultats ont montré l'efficacité de notre solution.

Toutes les expérimentations conduites dans ce chapitre ont prouvé l'efficacité de notre solution proposée, ainsi que l'importance et l'influence des étapes de distribution de données et de fusion des résultats dans le Clustering parallèle des données.

Conclusion générale et perspective

1 Conclusion générale

Les travaux présentés dans cette thèse se situent dans le contexte du Clustering parallèle de données. L'objectif de la thèse consistait en l'étude du domaine de Clustering, et plus particulièrement le Clustering parallèle afin de proposer un nouvel algorithme de Clustering parallèle ou l'adaptation d'un algorithme de Clustering déjà existant.

Dans cette thèse, nous nous sommes orientés vers l'utilisation du parallélisme basé sur le Framework MapReduce. L'utilisation du Framework MapReduce suit un paradigme bien précis basé sur deux phases principales : Map et Reduce, ainsi la parallélisation d'un algorithme en utilisant ce Framework doit suivre ce modèle.

Notre première contribution consistait en la proposition d'un schéma parallèle basé sur le paradigme MapReduce pour un algorithme de Clustering existant «Sampling-PSO-K-means». En plus de l'adoption du modèle MapReduce, plusieurs architectures existent pour l'implémentation d'une solution parallèle. L'étude de l'algorithme «Sampling-PSO-K-means» nous a permis d'opter pour l'architecture de parallélisme « SPMD » qui est la mieux adapté à ce dernier. Cette architecture consiste à (1) partager les données analysées en des sous-ensembles moins volumineux et (2) d'appliquer sur chacun l'algorithme de Clustering utilisé, et par la suite, (3) les résultats obtenus sur chaque partition seront fusionnés. Tout ce processus nécessitera un seul job MapReduce. Ces caractéristiques font que l'architecture SPMD (Single program, Multiple Data) soit la mieux adaptée pour l'algorithme «Sampling-PSO-K-means» due à la sensibilité de l'algorithme PSO aux données volumineuses ainsi qu'à la non adaptabilité du Framework MapReduce aux processus itératifs (Plusieurs Jobs MapReduce).

Toutes les contributions apportées dans cette thèse se situent au niveau des trois points cités plus haut. Ainsi dans notre première contribution nous avons aussi proposé d'utiliser un parallélisme local en plus de l'utilisation du Framework MapReduce. Ce parallélisme local est appliqué à l'étape (2) afin d'exploiter les ressources de calculs disponibles. Le parallélisme local consiste en l'utilisation de la mémoire partagée afin d'exploiter les CPU

multi-cœurs disponibles au niveau des machines du Cluster géré par MapReduce. Cette solution nous a permis d'améliorer les performances de notre algorithme en terme de temps de calcul.

Dans notre deuxième contribution, nous nous sommes orientés vers l'étape (3) qui concerne la fusion des résultats intermédiaires. Dans la première contribution une technique conventionnelle a été utilisée pour la génération d'une solution finale. Après l'analyse des résultats intermédiaire séparément, nous avons constaté que l'implication de tous ces derniers pour la génération d'une solution finale peut s'avérer ne pas être une solution optimale. Comme certains de ces résultats peuvent être impertinents à cause d'une mauvaise initialisation ou à cause de la manière dont les partitions sont créées, nous avons proposé de sélectionner parmi ces résultats ceux les plus pertinents afin de représenter la solution finale. Nous avons utilisé l'algorithme génétique afin d'obtenir la combinaison de centroïde la plus optimale. Cette solution nous a permis d'améliorer considérablement la qualité du Clustering en excluant les centroïdes imprécis et impertinents de la solution finale. Cela nous a montré l'importance de l'étape de la fusion des résultats pour la détermination de la qualité des résultats finaux.

Dans notre troisième et dernière contribution, nous nous sommes intéressés à l'étape (1) qui concerne la distribution des données. Nous avons suggéré que la qualité du Clustering peut être améliorée en changeant la distribution aléatoire adoptée par la majorité des travaux présentés dans la littérature.

Dans cette solution nous avons proposé de répartir les données sur l'ensemble des machines de manière à maximiser la distance entre les points de données appartenant à de différentes partitions. Avec cette démarche, nous avons supposé que la dissimilarité entre les centroïdes générés à partir des partitions différentes sera maximisée. La fusion de ces centroïdes en utilisant une méthode conventionnelle a généré des résultats impertinents, seule l'utilisation de la fusion par algorithme génétique nous a apporté des résultats pertinents. La distribution non aléatoire et la fusion par AG sont ainsi considérées complémentaires. Ceci nous a conduit alors à proposer dans cette contribution un schéma parallèle incluant une étape de distribution non-aléatoire et une étape de fusion basée sur l'AG. Les expérimentations ont montré que l'utilisation de cette solution nous a permis d'obtenir les meilleurs résultats en terme de qualité Clustering, de plus cette solution est applicable à différents algorithmes de Clustering par partitionnement.

Enfin, afin de tester la validité et la cohérence des résultats obtenus par notre approche, nous l'avons appliquée au problème de détection des communautés d'utilisateurs sur le réseau social Bibsonomy. Encore une fois notre approche a apporté les meilleurs résultats comparée à d'autres algorithmes de Clustering parallèles.

2 Perspectives

Notre première perspective est d'appliquer le schéma proposé à d'autres algorithmes de Clustering autres que les algorithmes de Clustering par partitionnement, afin d'observer les améliorations qu'on peut tirer à partir de notre approche et de profiter de ses avantages.

Notre deuxième perspective concerne l'amélioration de notre schéma proposé, plus précisément au niveau de l'étape de distribution de données. Nous visons à résoudre les inconvénients de cette étape qui consiste en la génération de partitions de données non équilibrées, ce qui a conduit à une perte en performance en terme de temps d'exécution.

Bibliographie

[1] (2016). Knowledge and data engineering group, university of kassel: Benchmark folksonomy data from bibsonomy, version of january 01st, 2016.

[2] Ahmadyfard, A. and Modares, H. (2008). Combining pso and k-means to enhance data clustering. In *Telecommunications, 2008. IST 2008. International Symposium on*, pages 688–691. IEEE.

[3] Almasri, A. and Shukur, G. (2008). Clustering using wavelet transformation. *Handbook of research on cluster theory*, 1:169.

[4] Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM.

[5] Appuswamy, R., Gkantsidis, C., Narayanan, D., Hodson, O., and Rowstron, A. (2013). Scale-up vs scale-out for hadoop: Time to rethink? In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 20. ACM.

[6] Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

[7] Bajorski, P. (2011). Clustering–unsupervised learning. *Statistics for Imaging, Optics, and Photonics*, pages 297–327.

[8] Bandyopadhyay, S., Giannella, C., Maulik, U., Kargupta, H., Liu, K., and Datta, S. (2006). Clustering distributed data streams in peer-to-peer environments. *Information Sciences*, 176(14):1952–1985.

[9] Banharnsakun, A. (2017). A mapreduce-based artificial bee colony for large-scale data clustering. *Pattern Recognition Letters*, 93:78–84.

[10] Ben-Dor, A., Shamir, R., and Yakhini, Z. (1999). Clustering gene expression patterns. *Journal of computational biology*, 6(3-4):281–297.

[11] Benz, D., Hotho, A., Jäschke, R., Krause, B., Mitzlaff, F., Schmitz, C., and Stumme, G. (2010). The social bookmark and publication management system bibsonomy. *The VLDB Journal The International Journal on Very Large Data Bases*, 19(6):849–875.

[12] Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer.

[13] Berry, M. J. and Linoff, G. (1997). *Data mining techniques: for marketing, sales, and customer support*. John Wiley & Sons, Inc.

[14] Bezdek, J. C., Ehrlich, R., and Full, W. (1984). Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2-3):191–203.

[15] Boley, D. (1998). Principal direction divisive partitioning. *Data mining and knowledge discovery*, 2(4):325–344.

[16] Bousbaci, A. and Kamel, N. (2014). A parallel sampling-pso-multi-core-k-means algorithm using mapreduce. In *Hybrid Intelligent Systems (HIS), 2014 14th International Conference on*, pages 129–134. IEEE.

[17] Bousbaci, A. and Kamel, N. (2016). Efficient results merging for parallel data clustering using mapreduce. In *Distributed Computing and Artificial Intelligence, 13th International Conference*, pages 349–357. Springer.

[18] Bousbaci, A. and Kamel, N. (2018). Efficient data distribution and results merging for parallel data clustering in mapreduce environment. *Applied Intelligence*, 48(8):2408–2428.

[19] Bradley, P. S. and Fayyad, U. M. (1998). Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99. Citeseer.

[20] Chaimontree, S., Atkinson, K., and Coenen, F. (2011). A multi-agent based approach to clustering: harnessing the power of agents. In *International Workshop on Agents and Data Mining Interaction*, pages 16–29. Springer.

[21] Choi, S.-S., Cha, S.-H., and Tappert, C. C. (2010). A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48.

[22] Cui, X., Charles, J. S., and Potok, T. (2013). Gpu enhanced parallel computing for large scale data clustering. *Future Generation Computer Systems*, 29(7):1736–1741.

[23] Cui, X., Gao, J., and Potok, T. E. (2006). A flocking based algorithm for document clustering analysis. *Journal of systems architecture*, 52(8-9):505–515.

[24] Cui, X. and Potok, T. E. (2005). Document clustering analysis based on hybrid pso+ k-means algorithm. *Journal of Computer Sciences (special issue)*, 27:33.

[25] Cui, X., Zhu, P., Yang, X., Li, K., and Ji, C. (2014). Optimized big data k-means clustering using mapreduce. *The Journal of Supercomputing*, 70(3):1249–1259.

[26] Danielsson, P.-E. (1980). Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248.

[27] Davidson, I. and Satyanarayana, A. (2003). Speeding up k-means clustering by bootstrap averaging. In *IEEE data mining workshop on clustering large data sets*.

[28] Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227.

[29] Davis, L. (1991). Handbook of genetic algorithms.

[30] Day, W. H. and Edelsbrunner, H. (1984). Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of classification*, 1(1):7–24.

[31] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[32] Dom, B. E. (2002). An information-theoretic external cluster-validity measure. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 137–145. Morgan Kaufmann Publishers Inc.

[33] Dubreuil, M., Gagné, C., and Parizeau, M. (2006). Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):229–235.

[34] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.

[35] Ene, A., Im, S., and Moseley, B. (2011). Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM.

[36] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[37] Fahim, A., Salem, A., Torkey, F. A., and Ramadan, M. (2006). An efficient enhanced k-means clustering algorithm. *Journal of Zhejiang University-Science A*, 7(10):1626–1633.

[38] Ferreira Cordeiro, R. L., Traina Junior, C., Machado Traina, A. J., López, J., Kang, U., and Faloutsos, C. (2011). Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 690–698. ACM.

[39] Flatt, H. P. and Kennedy, K. (1989). Performance of parallel processors. *Parallel Computing*, 12(1):1–20.

[40] Fränti, P. (2015). Clustering datasets.

[41] Gath, I. and Geva, A. B. (1989). Unsupervised optimal fuzzy clustering. *IEEE Transactions on pattern analysis and machine intelligence*, 11(7):773–780.

[42] Grama, A., Kumar, V., Gupta, A., and Karypis, G. (2003). *Introduction to parallel computing*. Pearson Education.

[43] Guénoche, A., Hansen, P., and Jaumard, B. (1991). Efficient algorithms for divisive hierarchical clustering with the diameter criterion. *Journal of classification*, 8(1):5–30.

[44] Hand, D. J. (2007). Principles of data mining. *Drug safety*, 30(7):621–622.

[45] Harman, D. (2011). Information retrieval evaluation. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 3(2):1–119.

[46] Hore, P., Hall, L., and Goldgof, D. (2006). A cluster ensemble framework for large data sets. In *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference on*, volume 4, pages 3342–3347. IEEE.

[47] Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666.

[48] Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254.

[49] Kamel, N., Ouchen, I., and Baali, K. (2014). A sampling-pso-k-means algorithm for document clustering. In *Genetic and Evolutionary Computing*, pages 45–54. Springer.

[50] Karp, R. M. (1988). A survey of parallel algorithms for shared-memory machines.

[51] Kaufman, L. and Rousseeuw, P. J. (1986). Clustering large data sets. In *Pattern Recognition in Practice, Volume II*, pages 425–437. Elsevier.

[52] Kaufman, L. and Rousseeuw, P. J. (2009). *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons.

[53] Kerdprasop, K. and Kerdprasop, N. (2010). A lightweight method to parallel k-means clustering. *International Journal of Mathematics and Computers in Simulation*, 4(4):144–153.

[54] Kraus, J. M. and Kestler, H. A. (2010). A highly efficient multi-core algorithm for clustering extremely large datasets. *BMC bioinformatics*, 11(1):169.

[55] Kriegel, H.-P., Kröger, P., Sander, J., and Zimek, A. (2011). Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240.

[56] Kwedlo, W. and Iwanowicz, P. (2010). Using genetic algorithm for selection of initial cluster centers for the k-means method. In *International Conference on Artificial Intelligence and Soft Computing*, pages 165–172. Springer.

[57] Lee, S.-S. and Lin, J.-C. (2012). An accelerated k-means clustering algorithm using selection and erasure rules. *Journal of Zhejiang University SCIENCE C*, 13(10):761–768.

[58] Li, H.-G., Wu, G.-Q., Hu, X.-G., Zhang, J., Li, L., and Wu, X. (2011). K-means clustering with bagging and mapreduce. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–8. IEEE.

[59] Lichman, M. (2013). UCI machine learning repository.

[60] Lin, K. W., Lin, C.-H., and Hsiao, C.-Y. (2014). A parallel and scalable cast-based clustering algorithm on gpu. *Soft Computing*, 18(3):539–547.

[61] Liu, Y., Li, Z., Xiong, H., Gao, X., and Wu, J. (2010). Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 911–916. IEEE.

[62] Lv, Z., Hu, Y., Zhong, H., Wu, J., Li, B., and Zhao, H. (2010). Parallel k-means clustering of remote sensing images based on mapreduce. In *International Conference on Web Information Systems and Mining*, pages 162–170. Springer.

[63] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[64] Maulik, U. and Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. *Pattern recognition*, 33(9):1455–1465.

[65] Merigo, J. M. and Casanovas, M. (2011). A new minkowski distance based on induced aggregation operators. *International Journal of Computational Intelligence Systems*, 4(2):123–133.

[66] Milligan, G. W. and Isaac, P. D. (1980). The validation of four ultrametric clustering algorithms. *Pattern Recognition*, 12(2):41–50.

[67] Moon, T. K. (1996). The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60.

[68] Ng, R. T. and Han, J. (2002). Clarans: A method for clustering objects for spatial data mining. *IEEE transactions on knowledge and data engineering*, 14(5):1003–1016.

[69] Niknam, T. and Amiri, B. (2010). An efficient hybrid approach based on pso, aco and k-means for cluster analysis. *Applied soft computing*, 10(1):183–197.

[70] Olson, C. F. (1995). Parallel algorithms for hierarchical clustering. *Parallel computing*, 21(8):1313–1325.

[71] Park, N. H. and Lee, W. S. (2004). Statistical grid-based clustering over data streams. *Acm Sigmod Record*, 33(1):32–37.

[72] Ramaswamy, S., Sapatnekar, S., and Banerjee, P. (1997). A framework for exploiting task and data parallelism on distributed memory multicomputers. *IEEE transactions on parallel and distributed systems*, 8(11):1098–1116.

[73] Saatchi, S. and Hung, C. C. (2005). Hybridization of the ant colony optimization with the k-means algorithm for clustering. In *Scandinavian Conference on Image Analysis*, pages 511–520. Springer.

[74] Salman, R., Kecman, V., Li, Q., Strack, R., and Test, E. (2011). Fast k-means algorithm clustering. *arXiv preprint arXiv:1108.1351*.

[75] Saoud, Z. and Platoš, J. (2017). Community detection in bibsonomy using data clustering. In *International Conference on Information Systems Architecture and Technology*, pages 149–158. Springer.

[76] Sarstedt, M. and Mooi, E. (2014). Cluster analysis. In *A concise guide to market research*, pages 273–324. Springer.

[77] Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 101–102. IEEE.

[78] Shirkhorshidi, A. S., Aghabozorgi, S., Wah, T. Y., and Herawan, T. (2014). Big data clustering: a review. In *International Conference on Computational Science and Its Applications*, pages 707–720. Springer.

[79] Singh, A., Yadav, A., and Rana, A. (2013). K-means with three different distance metrics. *International Journal of Computer Applications*, 67(10).

[80] Sinha, A. and Jana, P. K. (2016). A novel k-means based clustering algorithm for big data. In *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*, pages 1875–1879. IEEE.

[81] Stoffel, K. and Belkoniene, A. (1999). Parallel k/h-means clustering for large data sets. In *European Conference on Parallel Processing*, pages 1451–1454. Springer.

[82] Talia, D. (2002). Parallelism in knowledge discovery techniques. In *International Workshop on Applied Parallel Computing*, pages 127–136. Springer.

[83] Ting, K. M. (2011). Precision and recall. In *Encyclopedia of machine learning*, pages 781–781. Springer.

[84] Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., et al. (2001). Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584.

[85] Wang, J., Yuan, D., and Jiang, M. (2012). Parallel k-pso based on mapreduce. In *Communication Technology (ICCT), 2012 IEEE 14th International Conference on*, pages 1203–1208. IEEE.

[86] Wang, W., Yang, J., Muntz, R., et al. (1997). Sting: A statistical information grid approach to spatial data mining. In *VLDB*, volume 97, pages 186–195.

[87] White, T. (2012). *Hadoop: The definitive guide*. O'Reilly Media, Inc.

[88] Xu, S. and Zhang, J. (2004). A parallel hybrid web document clustering algorithm and its performance study. *The Journal of Supercomputing*, 30(2):117–131.

[89] Xu, X., Jäger, J., and Kriegel, H.-P. (1999). A fast parallel clustering algorithm for large spatial databases. In *High Performance Data Mining*, pages 263–290. Springer.

[90] Zhang, M. and Yu, J. (2004). Fuzzy partitional clustering algorithms [j]. *Journal of Software*, 6:007.

[91] Zhao, Q. (2012). Cluster validity in clustering methods. *Publications of the University of Eastern Finland*.

[92] Zhao, W., Ma, H., and He, Q. (2009). Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer.

[93] Zhou, P., Lei, J., and Ye, W. (2011). Large-scale data sets clustering based on mapreduce and hadoop. *Journal of Computational Information Systems*, 7(16):5956–5963.