République Algérienne Démocratique Et Populaire
Ministère de L'Enseignement Supérieur et de La Recherche Scientifique

Université BLIDA -1-

Faculté des Sciences

Département d'informatique

Memoire de fin d'études

En vue d'obtenir le diplôme de Master
Option : Génie des Systèmes informatique.

Extraction des itemsets fréquents à partir des données imparfaites dans le contexte du Big data

Présenté par :
Youcef Allaoua

Promotrice :
Mme Fatma Zohra Zahra

28 novembre 2017

Table des matières

Table des figures

Liste des tableaux

Résumé

Dans ce travail, nous proposons une extraction distribuées des itemsets fréquents à partir de données évidentielles. La solution est basée sur le modèle de programmation MapReduce afin de palier au problème de passage à l'échelle inhérent au Big Data.
La démarche utilisée se repose sur le calcul du support évidentiel précis, ensuite nous conservons que ceux qui sont fréquents (c'est-à-dire ceux qui ont des fréquences d'apparition appelées supports précis dont les valeurs sont supérieures ou égales à un seuil minimal fixé. L'algorithme proposé a été testé sur des données synthétiques et a montré son efficacité.

MOTS-CLéS : Itemsets fréquents, Données imparfaites, Supports précis, Big data, Hadoop, MapReduce.

Abstract

In this work, we propose a distributed extraction of frequent itemsets from obvious data. The solution is based on the MapReduce programming model to overcome the big data scaling problem.
The approach used is based on the calculation of the precise evidentiary support, then we keep only those which are frequent (that is to say those which have frequencies of appearance called precise supports whose values are greater than or equal to a threshold minimum fixed The proposed algorithm has been tested on synthetic data and has shown its efficiency.

Keywords : Imperfect data, Extraction of frequent itemsets, Big Data, Hadoop, MapReduce,

ملخّص

في هذا العمل ، نقترح استخراجًا موزعًا لمجموعات العناصر المتكررة من البيانات الواضحة.
يعتمد الحل على نموذج البرمجة ماب ردوس للتغلب على مشكلة تحجيم البيانات الضخمة. يعتمد النهج المستخدم على حساب الدعم الاستدلالي الدقيق، ثم نحتفظ فقط بالدعم المتكرر أي تلك التي لها ترددات مظهر تسمى الدعامات الدقيقة التي تكون قيمها أكبر من أو تساوي الحد الأدنى الثابت. تم اختبار الخوارزمية المقترحة على بيانات تركيبية وأظهرت كفاءتها.

Remerciements

Je tiens tout d'abord à remercier ALLAH le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce modeste travail.

Mes sincères remerciements vont à mes parents, qui m'ont toujours encouragés dans la poursuite de mes études, ainsi que pour leur aide, leur compréhension, et leur soutien.

Mes vifs remerciements vont également à ma promotrice Mme F.Zahra, pour ses précieux conseils et son aide durant toute la période du travail.

Mes remerciements s'adressent également aux membres du jury pour l'intérêt qu'ils ont portés à ma recherche en acceptant d'examiner mon travail et de l'enrichir par leurs propositions.

Enfin je remercie mes amis, qui par leurs prières et encouragement j'ai pu surmonter tous les obstacles.

Merci à tous et a toutes.

Introduction générale

Contexte

L'omniprésence d'Internet, la numérisation croissante des interactions de tout type, l'Open Data et l'émergence des objets connectés génèrent des volumes de données de plus en plus vertigineux. La variété des données s'est accrue aussi, avec notamment le développement des réseaux sociaux : images, sons et video [11], en effet, en 2015 les utilisateurs de YouTube ont téléchargé plus de 400 heures de nouvelles vidéos chaque minute de la journée [12], en 2012, Twitter affiche environ 175 millions de tweets tous les jours et compte plus de 465 millions de comptes [13], plus de 571 nouveaux sites Web sont créés chaque minute de la journée [14], en 2011, Walmart a géré plus d'un million de transactions clients chaque heure, ce qui est importé dans des bases de données qui contiennent plus de 2,5 petabytes de données [15], IDC International Data Corporation estime que d'ici 2020, les transactions commerciales sur Internet atteindront 450 milliards par jour [16].

L'utilisation du Big Data engendrera une nouvelle vague de croissance de la productivité et d'excédent du consommateur [17]. Par exemple, un Rapport de *McKinsey Global Institute* estime qu'un détaillant utilisant pleinement le Big data peut augmenter sa marge opérationnelle de plus de 60 %. Les Big data offrent des avantages considérables aux consommateurs ainsi qu'aux entreprises et aux organisations. Par exemple, les services mis à disposition par les données de localisation personnelle peuvent permettre aux consommateurs de capturer un excédent économique de 600 milliards de dollars [17], d'un autre coté, les administrateurs gouvernementaux européen pourraient économiser plus de 100 milliards d'euros (149 milliards de dollars) dans l'amélioration de l'efficacité opérationnelle uniquement en utilisant de Big data [17], mais afin d'utiliser cette masse phénoménale de données recueillie jour après jour, on doit passer par un processus nommé processus d'extraction de données, cette étape permet de filtrer la masse de donnée afin d'extraire des informations utilisables et utiles à nos besoins, mais reste à savoir si la masse de données collectées est parfaite ? si y'a pas un taux d'erreur à prendre en compte ? comme par exemple, les informations générées à partir des capteurs ou des fautes de frappe sur une information ou transaction...ect.

Selon [14] des données imparfaites dans les entreprises et le gouvernement coûtent à l'économie américaine 3,1 billions de dollars par année. En effet, la qualité de l'information a son importance cela influe directement sur les résultats obtenus, il existe 4 types d'imperfection de données, incertaine,

incomplète, imprécise et ambiguë, afin de minimiser cette imperfection, plusieurs recherche ont été faite et ça a permit de mettre en place des modèles théorique à suivre pour chaque cas d'imperfection.

Problématique

L'extraction des motifs fréquents est une des plus importantes techniques de data mining, en effet, plusieurs algorithmes ont été proposés dans la littérature pour extraire des motifs de données a partir de données Précises. Cependant, très peu de travaux traite le problème de l'imperfection de données dans le contexte de l'extraction de motif. La véracité de ses données est un des aspects les plus important dans l'extraction des motifs fréquents vu que cela a un rapport direct avec la qualité des résultats obtenu. D'un autre coté,le volume de données commerciales dans le monde entier, dans toutes les entreprises, est doublé tous les 1,2 ans , ce qui est problématique vu que plus la base de donnée est grande plus le processus d'extraction de données prendra plus de temps a extraire ses motifs.

Objectifs

L'objectif de notre travail est de proposer une conception et une implémentation de RS-Apriori qui s'exécute sous une architecture distribuée parallèlement afin de pallier le problème du temps de traitement des algorithmes d'extraction de motifs à partir des données imparfaites dans le contexte de Big Data en utilisent le framework haddop.

Organisation du mémoire

Le mémoire se réparti en quatre chapitres.

Chapitre 1 : Extraction des itemsets fréquentes à partir des données imparfaites

Ce chapitre est composé de trois grandes sections, la première section est destinée à définir les données imparfaites et les modèles théoriques mise en œuvre afin de gérer cette imperfection, la deuxième section est consacrée au processus d'extraction de connaissance c'est-à-dire les différentes étapes à suivre afin d'extraire une connaissance utile dans une grande masse de données, la dernière section décrit comment appliquer les modèles théoriques afin d'éxtraire des connaissances utiles par rapport à chaque type imperfection.

Chapitre 2 : Big Data

Dans ce chapitre on va définir les notions du big data et ses principaux problèmes, puis on va présenter un ensemble d'outils de développement afin

de pallier ses derniers et pour finir on expliquera en détaille le fonctionnement d'une de ces techniques la plus rependu, le framwork Hadoop.

Chapitre 3 : Solution proposée
Ce chapitre sera réservé à la définition des données évidentielles et la présentation du support utilisé afin d'extraire des itemsets a partir de cette dernière avec des exemples pour clarifier les choses.

Chapitre 4 : Test et Validation
Nous exposeront dans le dernier chapitre, notre travail, une démonstration d'utilisation avec des résultats.

Chapitre 1

Extraction des itemset fréquentes à partir des données imparfaites

1.1 Introduction

Récemment, l'Extraction des itemset fréquentes à partir de bases de données imparfaites commence à susciter de l'intérêt dans la communauté de fouille de données et certains travaux sont apparus dans ce sens [18]. Néanmoins, la majorité des techniques d'extraction des itemset fréquentes ne prennent pas en considération l'aspect imparfait des données générées par les applications du monde réel, Ce problème d'imperfection touche en effet plusieurs systèmes d'informations. En sciences expérimentales, les chercheurs se basent sur des expérimentations dont les données générées sont sauvegardées et par la suite traitées, ces valeurs observées ou mesurées sont la plupart du temps imprécises ou incertaines [19]. En médecine, les docteurs se trouvent souvent dans l'obligation d'émettre un diagnostic en présence de symptômes imprécis voir incertains [20]. Les données générées par certains systèmes à base de capteurs sont aussi imparfaites, les capteurs d'un système unique peuvent produire des informations à différents niveaux de confiance. En plus, chacun d'eux peut produire une information incertaine, imprécise ou incomplète [18].

Dans la plupart de ces travaux, l'imperfection des données est modélisée via la théorie des probabilités [21], la théorie des possibilités, les ensembles flous [22], la théorie d'évidence (nommée aussi théorie de Dempster-Shafer) [23] et la théorie des ensembles approximatifs.

Dans ce chapitre, on va définir l'imperfection des données et comment

traiter cette dernière, puis on se penchera sur l'extraction des itemsets fréquentes à partir des données imparfaites.

1.2 Données imparfaites

Dans la vie courante, l'homme fait souvent face à des décisions et des actions où l'information imparfaite est la seule information disponible, autrement dit, une impossibilité d'obtenir une information complète et certaine, car peut être elle n'existe pas.

Vu l'habilité et le comportement naturel de l'être humain et sa capacité de prendre des décisions en présence d'une grande masse d'informations imparfaites (incomplètes, incertaines ou imprécises. . .), l'automatisation de traitement de ce type d'information à fait l'objet de plusieurs recherches dont l'objectif était de proposer des modèles et méthodes pour représenter et manipuler l'information imparfaite pour faire face à des situations réelles[24].

L'objectif de cette section est de définir la donnée imparfaite, les types d'imperfection de données, les causes liées à l'existence des données imparfaites. Nous présentons également quelques méthodes pour modéliser l'imperfection de données.

1.2.1 Définition

On peut distinguer en étudiant les données, si les classes d'objets ou les objets suivants sont bien ou mal définis. Dans le cas où l'objet et la classe sont bien définis, on sera soumis à de l'incertitude. Dans les autres cas, (l'objet ou la classe sont mal définis) l'imperfection des données sera due à de l'imprécision, à de l'ambiguïté et/ou à de l'incomplétude [25].

1.2.2 Les types dimperfection des données

Beaucoup de tentatives ont été faites pour classifier les divers types possibles d'information imparfaite. Selon Peter Fisher [25], L'imperfection des données peut être classée en quatre grandes catégories rappelées dans la Figure 1.1.

1.2.2.1 Incertitude

L'incertitude est liée à la validité d'une connaissance [1], c'est à dir, l'objet où la classe est bien définie.

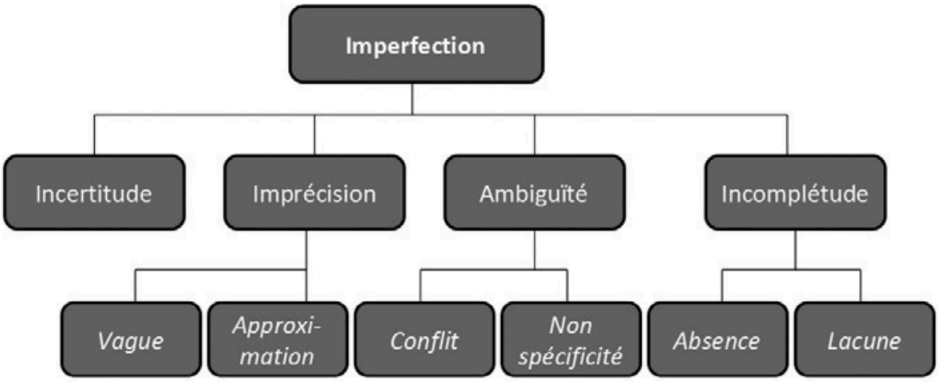

FIG.1.1: Une typologie de limperfection des données [1].

1.2.2.2 Imprécision

C'est la difficulté à exprimer clairement les connaissances [1], cela est due au caractère vague ou approximatif de la sémantique utilisée [26]. Autrement dit, l'objet n'est pas clairement défini. Il y a un manque de précision dans la définition de l'objet [1].

- **Vague :** L'information est vague, si elle est définie à l'aide de connaissances flexibles, par exemple : poids d'un bébé normal à la naissance se situe entre 2600 g et 4000 g, ou si elle est la conséquence d'une insuffisance des instruments d'observation, par exemple : l'erreur de mesure (poids à 1% près) [26].
- **Approximation :** L'information est approximative, si les limites sont mal définies, c'est-à-dire d'objets dont la représentation ne peut être délimitée clairement [26], par exemple : enfant, adulte, vieillard.

1.2.2.3 Ambiguïté

L'ambiguïté surgit lorsqu'il y a un doute sur la manière de définir un objet ou un phénomène, c'est-à-dire quand un élément peut appartenir à plusieurs catégories disjointes ou d'échelles différentes, ou encore quand la description de l'élément peut donner lieu à plusieurs sens [26] [1]. Selon Peter Fisher, il existe deux cas d'ambiguïté : le désaccord ou conflit, le manque de spécificité ou non-spécificité [25].

- **Conflit** : Si au minimum deux classifications contradictoires pour un unique objet sont possibles [26] [1].

- **Non-spécificité** : Lorsqu'une définition d'une relation ou d'un objet peut amener à plusieurs sens, ou lorsque l'échelle de l'analyse est susceptible d'amener à de multiples interprétations, on parle alors de non-spécificité [26] [1].

1.2.2.4 Incomplétude

Les incomplétudes sont des absences de connaissances ou des connaissances lacunaires sur des informations du système [26].

- **Absence :** L'absence est le phénomène qui survient quand dans une base de données, des valeurs manquent à la description de certains objets ou un manque d'information nécessaire à la définition d'un objet [26], par exemple : certaines valeurs ne sont parfois pas remplies dans une base de donnée.

- **Lacune :** Est le fait qu'un ou plusieurs objets de la base de données ne décrivent que de manière partielle une structure les englobant [26] [1].

1.2.3 Les causes dimperfection des données

L'imperfection des données est due à plusieurs raisons, deux d'entre elles sont :

- **L'obtention des données à partir du réel** : celle ci s'effectue en deux étapes (l'observation puis la représentation). La première se produit à cause des intermédiaires qui sont humains ou matériels. Ces derniers sont généralement soumis à des erreurs, des imprécisions et des incertitudes. La deuxième étape qui est la représentation des données entraîne elle aussi une perte d'informations qui augmente quand le système devient de plus en plus complexe [24].

- **L'absence de la flexibilité liée au système** : c'est le cas pour toutes les caractéristiques de phénomènes naturels tels que la durée de maturation d'un fruit, la taille d'un animal adulte, le passage progressif et non strict du jour à la nuit ; c'est aussi le cas de certains systèmes artificiels, tels que la charge maximale d'un ascenseur, indiquée en kilogrammes dans un souci de simplicité mais à laquelle on peut ajouter quelques grammes sans problème majeur ou le nombre maximal de voyageurs que peut contenir un wagon de métro, dépendant du degré de compression accepté par les passagers [27].

1.2.4 Représentations formelles de linformation imparfaite

Afin de manipuler des données imparfaites, de nombreux cadres théoriques ont été formalisés. Nous en présentons ici les principaux.

1.2.4.1 Théorie des probabilités

Elle est liée à la mesure ou à l'estimation d'événements bien définis mais dont la réalisation est caractérisée par le hasard. La probabilité qu'un tel événement va survenir est ainsi définie par une fonction P(A) à valeur dans [0;1]. Elle est très bien adaptée au traitement de l'incertitude [1]. Il existe deux larges catégories d'interprétation des probabilités : les probabilités objectives et les probabilités subjectives [26].

- **Les probabilités objectives** : ou approche fréquentiste des probabilités, sont associées aux systèmes ayant un fonctionnement aléatoire à l'image du jeux de la roulette, le lancer de dé ou la durée de vie d'un atome radioactif. Dans ces systèmes, la probabilité d'un événement (par exemple « le résultat du lancer d'un dé est 6 ») correspond à la fréquence de réalisation de l'événement lorsque l'on répète un nombre important de fois, l'expérience [26].

- **Les probabilités subjectives** : aussi appelées bayésiennes peuvent être affectées à n'importe quel fait (même issu de phénomène non aléatoire). Elles représentent la plausibilité subjective, ou le degré de confiance que l'on donne à la validité du fait. Le plus généralement, on considère que la valeur d'une probabilité subjective est un degré de croyance. Par exemple, à la question « quelle est la probabilité qu'il neige demain ? », un agent n'ayant pas accès aux modélisations météorologiques donnera une réponse subjective.[26].

1.2.4.2 Théorie des Ensembles flous

Introduite par Lotfi A. Zadeh [28], l'appartenance d'un objet x à un ensemble A n'est plus booléenne (vrai/faux) mais est caractérisée par un degré d'appartenance µA(x) pouvant varier dans l'intervalle [0;1] où les valeurs extrêmes correspondent aux valeurs booléennes classiques avec 0 pour faux et 1 pour vrai. De nombreuses extensions au "flou" de la représentation booléenne (tout ou rien) du monde ont été développées : logique floue, arithmétique floue… Ce formalisme conceptuel est bien adapté au traitement de l'imprécis [1].

1.2.4.3 Théorie des possibilités

Introduite par Lotfi A. Zadeh en 1978 [29] dans le but de traiter conjointement les informations imprécises et incertaines[26], l'incertitude d'un événement A, au contraire des probabilités, est caractérisée par deux valeurs : sa possibilité (A) et sa nécessité N(A)[1].

1.2.4.4 Théorie de lévidence

Aussi appelée théorie des fonctions de croyance ou encore théorie de Dempster-Shafer *DSET* [26], fut introduite par Arthur Dempster dans [30] puis formalisée par Glenn Shafer dans [23], les raisonnements sont effectués à partir de deux mesures et non une seule (la crédibilité et la plausibilité), qui permettent de représenter à la fois l'incertitude et l'imprécision [26].

1.2.4.5 Théorie des ensembles approximatifs

Introduite par Zdzislaw Pawlak en 1982 [31], le même auteur proposa en 1991 un ouvrage [32], présentant de manière plus approfondie cette théorie, elle permet de traiter les données incomplètes, imprécises et incertaines [24]. Avec chaque ensemble approximatif (imprécis) est associés deux ensembles exacts, approximation basse et un autre ensemble appelé approximation haute. C'est l'idée de base : si on ne peut pas définir un objet exactement (puisque par exemple, l'information disponible ne le permet pas), on peut néanmoins le délimiter par deux limites (bornes) inférieure et supérieure [24].

1.2.5 Extraction des itemsets fréquents à partir de données

C'est-à-dire des groupements d'items apparaissant ensemble avec une fréquence significative, qui constitue l'étape principale de l'extraction de règles d'association [33]. Afin de mieux illustrer les notions de base de l'extraction des itemsets fréquents, on as repris l'exemple [2] suivent :

En ligne, nous avons des clients d'une compagnie d'assurance ; en colonne, des contrats (produits) associés à divers risques Tableau : 1.2.5. Par exemple, le client n°1 a contracté les assurances S1 et S3, etc. L'objectif est d'identifier les produits qui sont placés de concert[2].

- **Item :** Un item correspond à un produit. Nous avons 4 items (S1, S2, S3 et S4) dans notre fichier [2].

S1	S2	S3	S4
1	0	1	0
0	1	0	0
0	0	0	1
0	1	1	1
0	1	1	0
0	1	1	0
1	1	1	1
1	0	1	0
1	1	1	0
1	1	1	0

Tableau1.1: Exemple comportent 10 observations (transactions) et 4 items [2].

  - **Support :** Le support d'un item est égal au nombre de transactions dans lesquelles il apparaît [2]. Par exemple, le support de S1 est égal à 5. Le support peut être exprimé également en termes relatifs . Dans ce cas, nous division le support (absolu) par le nombre total de transactions[2]. Pour S1, nous avons SUP($\{S1\}$) = 5/10 =20%.

- **Itemset :** Un itemset est un ensemble d'items (ex. $\{S1, S2\}$ est un itemset de cardinal $CARD(\{S1, S2\} = 2)$. Le support d'un itemset comptabilise le nombre de transactions dans lesquelles les items apparaissent simultanément (ex. SUP($\{S1, S2\}$) = 3 /10 = 0.3). Un itemset peut être composé d'un singleton[2] (ex. $\{S1\}$ avec SUP($\{S1\}$) = 5/10) [2].

- **Itemset fréquent :** Un itemset est dit fréquent si son support est supérieur à un seuil défini à l'avance, paramètre de l'algorithme de recherche. Dans notre exemple, en fixant le support minimum à 2 (ou 20% en relatif), nous observons dans la figure 1.3 les itemsets fréquents (toutes les combinaisons non-grisées) [2].

- **Superset :** Un superset est un itemset défini par rapport à un autre itemset[2] . Prenons un exemple pour clarifier les idées : $\{S1, S2, S3\}$ est un superset de $\{S1, S2\}$. Ainsi, de manière générale, B est un superset de A, si $CARD(A) < CARD(B)$ et que $A \subset B$ c'est-à-dire on retrouve dans B tous les items de A.
  Nous remarquons une propriété très importante du support : $SUP(B) \leqslant SUP(A)$. En particulier, si A n'est pas fréquent, alors B ne le sera pas également [2]. Ce résultat permet de réduire considérablement l'espace de recherche lors de l'extraction des itemsets fréquents dans une base de données.

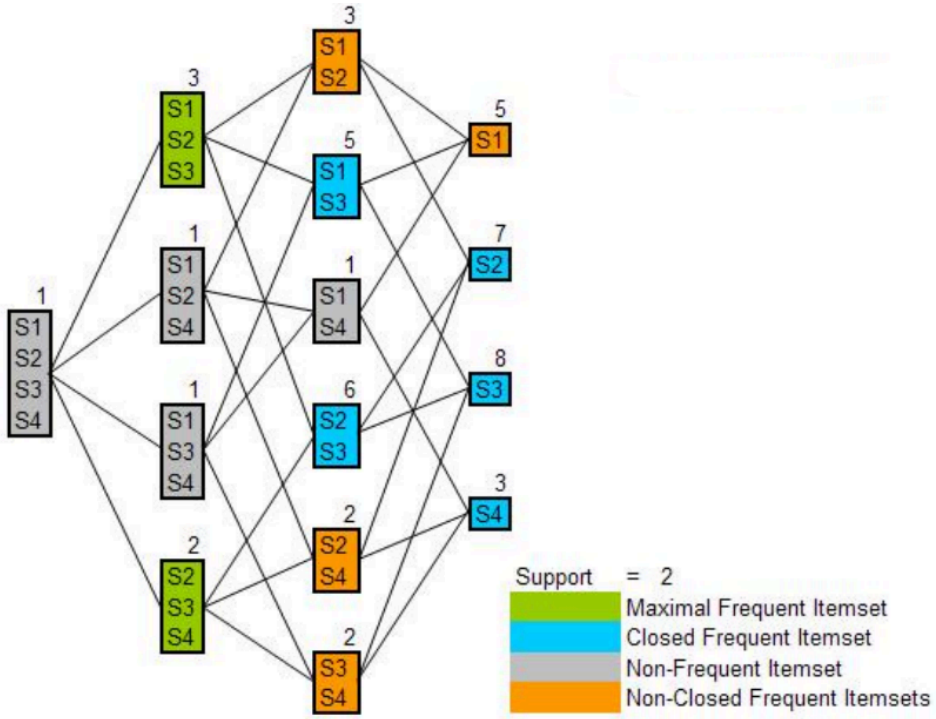

FIG.1.2: Extraction des différents itemsets [2].

- **Itemset fermé (closed itemset) :** Un itemset fréquent est dit fermé si aucun de ses supersets n'a de support identique [2]. Autrement dit, tous ses supersets ont un support strictement plus faible. Dans notre exemple ci-dessus, S1, S3 est fermé car aucun de ses supersets n'a de support égal à 5/10 : $SUP(\{S1, S2, S3\}) = 3/10$, $SUP(\{S1, S3, S4\}) = 1/10$.

- **Itemset maximal (maximal itemset) :** Un itemset est dit maximal si aucun de ses supersets n'est fréquent [2]. Dans notre exemple ci-dessus, $\{S1, S2, S3\}$ est maximal car son superset $\{S1, S2, S3, S4\}$ (il n'y en a qu'un) n'est pas fréquent avec un support de 1/10.

- **Itemset générateur (generator itemset) :** Un itemset A est dit générateur s'il n'existe aucun itemset B tel que $B \subset A$ et que $SUP(B) = SUP(A)$. Autrement dit, l'itemset est générateur si tous ses sous-itemsets ont un support strictement supérieur [2]. Dans notre exemple, $\{S1, S2, S3$ de support 4/10 n'est pas générateur puisqu'on trouve { S1, S2 } avec un support identique. Il en est de même en ce qui concerne $\{S1, S3\}$ à cause de $\{S1\}$. En revanche, $2, S4\}$, de support 2/10, est générateur parce que $SUP(\{S2\}) = 7/10$ et $SUP(\{S4\}) = 3/10$.

1.2.6 Règles dassociation

Le concept de règle d'association a été popularisé, en particulier, par un article de Rakesh Agrawal [33], la recherche des règles d'association est une méthode populaire étudiée d'une manière approfondie dont le but est de découvrir des relations cachées entre deux ou plusieurs variables stockées dans de très importantes bases de données [34].

1.2.6.1 Création des règles d'association

Les règles d'association sont généré par une analyse sur la base de donné en se basent sur deux mesures bien définie, le support et la confiance.

Le support : est la fréquence d'apparition simultanée des items figurant dans l'ensemble des données (définie dans la section précédente).

La confiance : est le rapport du nombre de fois que la relation (si / alors) a été déclaré comme étant vrai sur le Nombre de données où les items de la condition apparaissent simultanément.
Il existe plusieurs algorithmes qui permettent la création des règles d'association, parmi les plus connus : A-priori, Eclat, FP-growth.

1.3 Extraction des itemsets fréquents

Afin de bien illustré comment extraire des informations utile d'un ensemble de données nous allons présenter et appliquer un algorithme d'extraction des itemsets à partir de données parfaite, ce dernier est nommée A-Priori. L'algorithme Apriori 1 a été proposé par Agrawal et Srikant en 1994 [35], Conçu pour fonctionner sur des bases de données contenant des transactions (par exemple, des collections d'articles achetés par les clients) avec pour but, la détermination des règles d'association présentes dans un jeu de données, pour un seuil de support et un seuil de confiance fixés. Ces deux valeurs peuvent être fixées arbitrairement par l'utilisateur.

1.3.0.0.1 Fonctionnement de l'algorithme Apriori: L'algorithme A-Priori fonctionne en deux étapes :

1. La recherche des ensembles d''items fréquents (EIF),
2. utilisation de ces EIF pour déterminer les règles d'association dont la confiance est supérieure au seuil fixé [36].

La construction des EIF est itérative : on construit d'abord les EIF contenant un seul item, puis ceux contenant 2 items, puis ceux contenant 3 items, …On commence par déterminer les EIF de taille 1, on note cet ensemble L_1. Ensuite, on construit l'ensemble C_2 des EIF candidats de taille 2 : ce sont tous les couples construits à partir des EIF de taille 1, on obtient la liste des EIF de taille 2 (L_2) en ne conservant que les éléments de C_2 dont le support est supérieur au seuil., on construit alors C_3, l'ensemble des triplets d'items dont les 3 sous-paires sont dans L_2 et on ne retient que ceux dont le support est supérieur au seuil, ce qui produit L_3. Et ainsi de suite, tant que L_i n'est pas vide [36].

1.3.0.0.2 Remarque: Toutefois, si on considère X_k un sous-ensemble d'items fréquent, tous les sous-ensembles d'items contenus dans X_k et qui soient de longueurs inférieurs à k sont fréquents. Par exemple si ABCD est un sous-ensemble d'items fréquents, alors, les sous ensembles : ABC, ABD, BCD, AB, AC, BC, BD, CD, A, B, C, D le sont aussi [36].

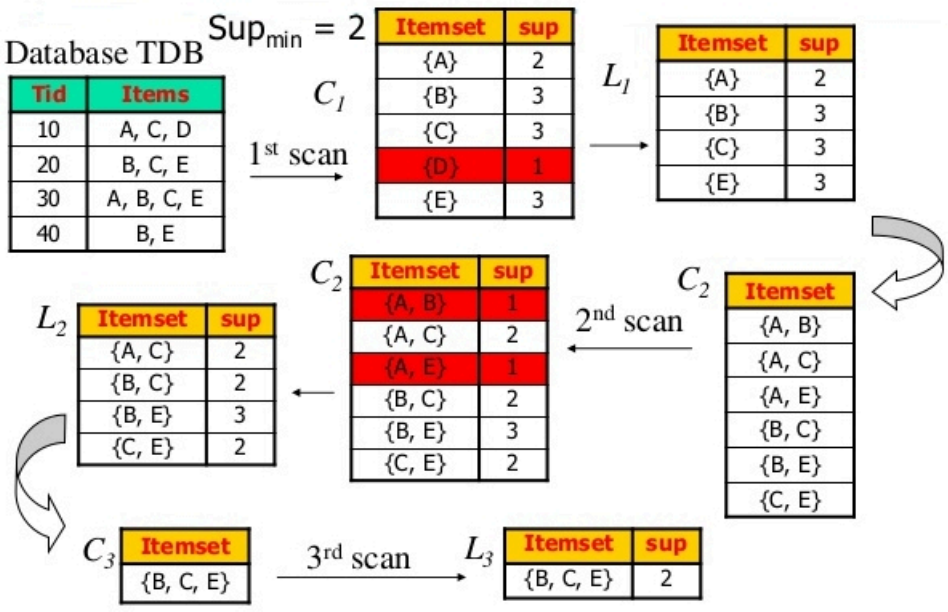

FIG.1.3: Exemple explicative de lalgorithme [2].

1.3.0.0.3 Génération des règles: Disposant des EIF, il nous faut maintenant les transformer en règles. Avec les notations précédentes, on dispose d'un ensemble de L_i pour des i croissant de 1 à une certaine valeur, chaque

L_i étant une liste de i items dont la fréquence d'apparitions est supérieure à un certain seuil [36]. Supposons que L_3 contienne le triplet (a, b, c). Plusieurs règles d'association peuvent être engendrées par ce triplet :

1. si a et b alors c
2. si a alors b et c
3. si b et c alors a
4. si b alors a et c
5. si a et c alors b
6. si c alors a et b

Parmi ces 6 règles candidates, on ne doit retenir que celles dont la confiance est supérieure au seuil fixé [36].

1.3.0.0.4 Remarque: pour cet exemple on a pu énumérer des règles candidates, il faut bien être conscient que dans une application réelle cela est impossible car le nombre de règles candidates généré devient beaucoup trop grand [36].

Algorithm 1 A-priori

$CalculerL_1$
$k \leftarrow 2$
while $L_{k-1} <> \phi$ **do**
  $C_k \leftarrow apriori - gen(L_{k-1})$
  while $t \in D$ **do**
    $C_t = sousensemble(C_k, t)$
    while $c \in C_t$ **do**
      $c.cont++$
    end while
  end while
  $L_k \leftarrow c \in C_k | c.cont \geq minSup$
  $k \leftarrow k + 1$
end while
return kUL_k

1.4 Lextraction des itemset fréquents à partir de données imparfaites

Cela à pour but, l'extraction d'un savoir ou d'une connaissance à partir de grandes quantités de données ou de base de données imparfaite [37], afin de présenter à l'utilisateur final une information fiable et interprétable pour l'aider à la prise de décision.
Les données imparfaites sont représentées par quatre grands types d'imperfection comme on l'a vu dans la première section Figure 1.1 (incertitude, imprécision, incomplétude, ambiguïté) [38], cependant chaque aspect de l'imperfection est traité d'une façon indépendante [3].

Nous aborderons en détail dans cette section les modèles théorique à suivre afin d'extraire des itemsets à partir de chaque type d'imperfection

1.4.1 Extraction des itemsets fréquents à partir de données incertaines

Une valeur est dite incertaine si la validité de la donnée est mise en question pour la validation de l'information [3]. Afin de généré les itemsets fréquents à partir des données incertaines plusieurs méthodes ont vu le jour *Expected Support*, *Probabilistic Support*, *Evidential Support*, illustre dans la figure 1.4.

Dans ce qui va suivre nous allons voir ces trois mesures de calculs.

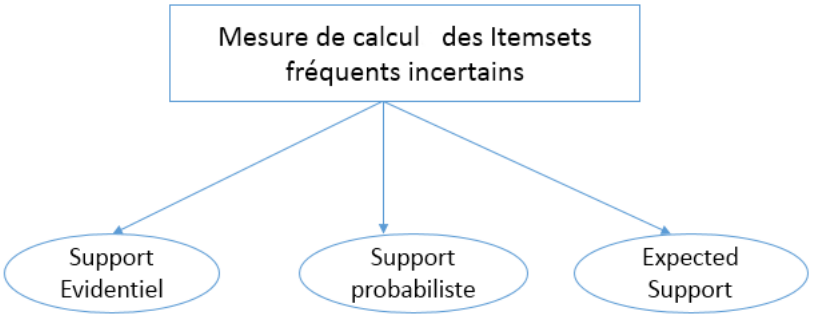

FIG.1.4: Les mesures de calcul des itemsets fréquents [3].

1.4.1.1 Expected Support

Expected support d'un itemests X dans la base de données incertaine est la somme de produit de probabilité $P(X, tj)$ de X dans la transaction tj sur tous les n transaction dans la base de données [3]. L'expression de la fonction Expected Support [39] est présentée comme suit :

$$ExpSup(X) = \sum_{j=1}^{n} P(X, tj) = \sum_{j=1}^{n} \left[\prod_{x \in X} P(x, tj) \right] \tag{1.1}$$

On dit que l'itemset X est un fréquent si son Expected support est supérieur ou égale au seuil minsup [40].

$$ExpSup(X) \geqslant minsup \tag{1.2}$$

1.4.1.2 Probabilistic Support

Dans les bases de données de transactions incertaines, le support d'un ou plusieurs éléments est représenté par une distribution de probabilité discrète [3].

Soit T, une base de données de transaction et W l'ensemble des éléments possibles de T, le $P_i(X)$ est la probabilité de support d'un itemset X tel que X a le soutien i [3].

$$P(X) = \sum_{w, W(S(X,wj)=i)} P(Wj) \tag{1.3}$$

Où $S(X, wj)$ est le support de X dans un élément wj.

Le support probabiliste d'un itemset X dans une base de données de transaction incertaine T est donné par les probabilités de support de $X(P_i(X))$ pour toutes les valeurs possibles de seuil $i \in \{0, ...|T|\}$ [3].

$$\sum_{0 \leqslant i \leqslant |T|} P_i(X) = 1.0 \tag{1.4}$$

Soit I, l'ensemble de tous élément possible et T la base de données incertaines, où une transaction $tj \in T$ est un ensemble des éléments incertains, à savoir, $tj \subseteq I$. Contrairement à la base de données précise traditionnelle, chaque item $xi/intj$ est associé à une probabilité existentielle $P(xi, tj) \in [0, 1]$, ce qui dénote la probabilité que xi est réellement présent dans tj. Pour un itemset $X \subseteq tj$, basé sur l'hypothèse commune que les éléments de X sont indépendants de la probabilité existentielle [3].

$$P(X \subseteq tj) = \prod x \in XP(x, tj) \tag{1.5}$$

Le seuil attendu (Expected support) de X est alors la somme de la probabilité existentielle sur toutes les transactions [3].

$$ExpSup(X) = \sum tj \in TP(X \subset tj) \tag{1.6}$$

La probabilité fréquente $P(X)$ de X peut être calculée par sommer $P_i(X)$ pour tout i $\geqslant minsup$:

$$P(X) \sum_{i=MinSup}^{|T|} P_i(X) \tag{1.7}$$

Où $P_i(X)$ enregistre la probabilité de X qui se produit exactement dans une transaction :

$$P_i(X) = \sum_{s \subseteq T, |S|=i} \left(\prod_{t \in S} P(X \subseteq t) \prod_{t \in T-S} (1 - P(X \subseteq t)) \right) \tag{1.8}$$

Si $P(X) \geqslant minprob$, alors X est un itemset fréquent probabiliste [3].
On donne :

- une base de données de transaction incertaine T,
- un seuil minimum de support minsup,
- un seuil minimum de la probabilité fréquente minprob, le problème d'extraction de motifs fréquents probabilistes est de trouver tous et seulement les motifs fréquents probabilistes ; à savoir, trouver tous les X tel que $P(X) \geqslant minprob$ [3].

1.4.1.2.1 Remarque: Un itemset X est un fréquent probabiliste si son existence dans une transaction minsup est supérieure ou égale au seuil d'utilisateur spécifique minprob [41].

$$P(sup(x) \geqslant minSup) \geqslant minProb \tag{1.9}$$

1.4.1.3 Evidential Support

Hewawasam et al [42], ont introduit une nouvelle approche pour support itemset informatique et appliqué sur un Fréquent Itemset Maintenance (FIM) problème. Toutes les méthodes ont été basés sur le produit cartésien entre

BBAs. Le support d'un itemset $X = \prod i \in [1....n]$ tel que xi est un item évidentiel appartenant au cadre de discernement i. étant donné que les items ne partagent pas le même cadre de discernement, toute règle de fusion ne peut pas être appliquée. Dans ce qui suit, nous étudions le support de croyance introduit par Dalvi et al [43] calculé par l'équation suivante [3] :

$$mj(X) = \prod_{x_i \in X} mij(x_i) \tag{1.10}$$

Où $mj(X)$ est le produit cartésien de tous BBA dans la transaction Tj. Ainsi, le BBA de l'itemset X exprimé dans l'ensemble EDB devient :

$$m_{EDB}(X) = \frac{1}{d} \sum_{j=1}^{d} mj(X) \tag{1.11}$$

Ensuite, le support de x dans la base de données EDB devient :

$$Support_{EDB}(x) = Bel_{EDB}(X) \tag{1.12}$$

Le produit cartésien d'un support à base, présenté ci-dessus, remplit plusieurs propriétés telle que la propriété anti-monotonie. Une mesure de support satisfaisant la propriété anti- monotonie consiste dans le fait qu'un itemset qui contient un itemset fréquent est également fréquent. L'inverse est vrai, tous les itemsets constituant un des plus fréquents sont également fréquents. Avec cette propriété satisfaite, la construction d'un algorithme Apriori devient simple [44].

1.4.2 Extraction des itemsets fréquents à partir de données imprécises

Selon Bloch, le concept d'ensembles flous gère l'incertitude et l'imprécision des données [45]. Cependant, ce dernier a été largement utilisé pour les règles d'associations à partir de données imprécises plutôt que des données incertaines. la mesure de calcul utiliser pour l'extraction des itemsets fréquents à partir de données imprécises est le *Fuzzy Support* que nous allons voir ci-dessous [3].

1.4.2.1 Fuzzy Support

On dénote fuzzy base de donnée par le triple $FBD = \{A\ O\ R\}$ ou R est le rapport floue entre un objet et une transaction exprimé par une fonction

d'adhésion [46] $Tj(i)$ tel que :

$$\mu Tj(i) = \propto (\propto \in [0\ 1]) \tag{1.13}$$

Ou la fonction $\mu Tj(i)$ évalue le degré d'adhésion de l'objet considéré à la transaction Tj.
Le calcul de Support dans de telles bases de données est fait en employant la fonction $Count(i)$ de la façon suivante :

$$Count(i) = \sum_{j=1}^{d} \mu Tj(i) \tag{1.14}$$

Le support d'un item i dans la base de donnée floue est défini comme suite :

$$Support(i) = \frac{Count(i)}{d} \tag{1.15}$$

Ainsi, pour un itemset X de la taille q tels que le $x_i \in X$ et $i \in [1, q]$ le Support devient :

$$Support(X) = \frac{\sum_{j=1}^{d} Min\left\{\mu Tj(xi), i = 1...q\right\}}{d} \tag{1.16}$$

1.4.3 Extraction ditemsets fréquents à partir de données incomplètes

Une donnée est dite incomplète, si elle contient au moins une valeur manquante, pour cela il existe différentes méthodes pour traiter l'incomplétude, tel que l'imputation de données ou on remplace les données manquantes par les données probables et/ ou possibles, par la suite les données deviennent incertaines. C'est-à-dire quand on élimine le problème d'incomplétude un autre problème apparaît celui de l'incertitude alors on applique les méthodes connus pour les itemsets incertains comme vu précédemment [3].

1.5 Conclusion

Jusqu'à maintenant nous avons vu les différentes natures d'imperfections de données et les théories connu afin d'extraire des connaissances à partir de ce dernier, dans le chapitre suivant on va exposer les notions du big data et on se penchera sur une des technologies les plus utilisées dans le traitement des données en parallèle dans ce contexte, le Framework hadoop.

Chapitre 2

Big Data

2.1 Introduction

Chaque minute passée sur internet, les utilisateurs de Facebook éditent 3,4 millions de statuts et génèrent 4 GB de données digitales, Google répond à 300 000 recherches et reçoit 126 heures de vidéos et pas moins de 700 nouveaux utilisateurs rejoignent Twitter. Au même moment, 350 000 tweets sont générés [5], en somme, nous générons 2,5 trillions d'octets de données, A tel point que 90% des données dans le monde ont été créées au cours des deux dernières années seulement [**?**]. Ces données proviennent de partout : de capteurs utilisés pour collecter des informations, de messages et publication sur les sites de médias sociaux, d'images numériques et de vidéos publiées en ligne, d'enregistrements transactionnels d'achats en ligne et de signaux GPS de téléphones mobiles. Ces données sont appelées Big Data ou volumes massifs de données [47].
Afin de faire face à l'énorme quantité de données numériques actuellement en circulation, il est devenu impératif de développer de nouvelles technologie pour les gérer et les analyser.

Dans le présent chapitre, nous allons aborder ce phénomène qu'on appelle Big Data. Nous commencerons par présenter ses définitions et caractéristiques, nous exposeront ses technologies majeures et ses différentes paradigme puis nous nous attarderons sur le framework Hadoop.

2.2 Définition

Les big data ou mégadonnées désignent l'ensemble des données numériques produites par l'utilisation des nouvelles technologies à des fins per-

sonnelles ou professionnelles, qu'aucun outil classique de gestion de base de données ou de gestion de l'information ne peut vraiment gèrer [48].

Le terme Big Data fait référence à la croissance exponentielle des données et au traitement de ces dernières ou de manière plus globale à toutes les étapes entrant en jeux dans le processus d'extraction d'informations utiles à partir de l'énorme lot de données brutes.[49]

Le concept du big data regroupe une famille d'outils qui répondent à une triple problématique dite règle des 3V.Volume,Variété,Vélocité [50].

Pour rendre plus pertinente et plus actuelle cette analyse tricéphale, certains acteurs peuvent ressentir le besoin d'y ajouter un ou deux autres V (à savoir la Véracité et la Valeur.On parlera alors des 5V du Big Data [51].

2.2.1 Volume

Le volume est issu de la croissance des systèmes embarqués dans le réseau intelligent, les villes intelligentes, le suivi des éléments logistiques, les réseaux sociaux.. etc. Les études estiment que la taille des données digitales augmentera jusqu'à atteindre 35 zettaoctets en 2020 alors qu'elle était a tout juste 0,5 zettaoctet en 2008 [4]. Ce qui a conduit à un volume de données considérables à traiter. (1 Zetaoctet= $10^{21} Octet$)

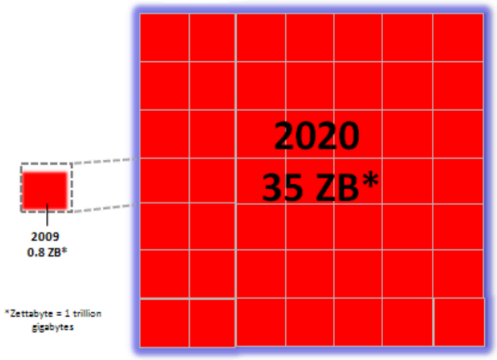

FIG.2.1: évolution de la taille des données entre 2008 et 2020 [4]

2.2.2 Vélocité

C'est la fréquence à laquelle les données sont générées, recueillies et analysées [52] pour une meilleure compréhension et décision [50], cette vitesse pose de nombreux problèmes lorsque ces données doivent être analysées en temps réel. Avant, réservé à certains secteurs de l'industrie, maintenant, il touche

de nombreux domaines : réseaux de capteurs sans fil, flux des publications dans les réseaux sociaux, données des observatoires scientifiques...,[49].

2.2.3 Variété

Le Big Data est alimenter par une masse de données qui proviennent de différentes sources et se présentent ainsi sous différents formats,[49] structurées ou non structurées (texte, données de capteurs, son, vidéo, données sur le parcours, fichiers journaux, etc.), Elles peuvent également être semi-structurées, entre les deux (format flexible pouvant être interprété : XML), ses données sont dites hétérogènes [5].

2.2.4 Véracité

Les 3 V (Volume, Variété, Vélocité) ne peuvent se déployer dans toute leur ampleur que si la donnée qu'ils mobilisent à la base est fiable. La véracité de la donnée, sa précision, sa pertinence, vont donc revêtir une importance cruciale, invitant les entreprises à une très grande rigueur aussi bien dans la façon dont elles orchestrent la collecte des données, que dans la manière dont elles vont les recouper, les croiser, les enrichir. [51]

2.2.5 Valeur

Rien ne sert de se lancer dans un projet de Big Data sans lui avoir assigné au préalable des objectifs précis qui se traduiront très concrètement par une génération de valeur pour l'entreprise [51]. Autrement dit, pouvoir extraire des données fiables et utiles d'une grande base de données pour l'entreprise.

2.3 Technologie du Big Data

C'est un ensemble d'outils développer pour répondre au triple problématiques (Volume,Vélocité,Variété) du big data, afin de permettre les traitements massivement parallèles, la gestion en temps réel des pannes de systèmes, la redondance systématique des données...

2.3.1 Propriétés

2.3.1.1 Scalabilité

C'est la capacité d'un système à améliorer ses performances en augmentant la taille ou le nombre de ses ressources lorsqu'il fait face à une charge

plus grande. il existe deux approches dites scalabilité verticale et son analogue horizontale.

- **Scalabilité verticale :** consiste à augmenter la taille du système et la puissance de ses composants (RAM, CPU…) [5].
- **Scalabilité horizontale :** présente sous la forme d'un cluster [5]. Il s'agit d'un système distribué composé de plusieurs machines de capacité modérée appelées nœuds. Ces machines ou nœuds communiquent dans le but de réaliser certaines opérations et manipuleront chacune une partie de la charge imposée au système adoptant ainsi la politique « diviser pour régner ». La charge peut représenter une problématique de stockage d'une grande masse de données ou leur traitement. La figure 2.2 résume le concept.

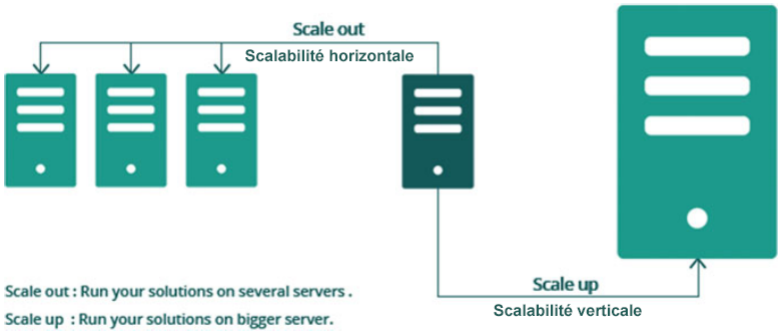

FIG.2.2: La scalabilite verticale et la scalabilite horizontale [5].

Arriver a un certain point, la scalabilité vertical rencontre des limites d'applicabilité, il n'est plus possible d'augmenter la puissance d'un système résidant sur une seule machine indépendamment de la disponibilité des ressources et de la taille du budget, comme la scalabilité est nécessaire à n'importe quel système Big Data vu le volume et la vélocité des données, l'approche horizontale est le plus souvent adoptée [5].
Dans notre cas, on utilisera l'approche de la scalabilité horizontale.

2.3.1.2 Théorème CAP

Aussi connu sous le nom de théorème de Brewer, il dit qu'il est impossible sur un système informatique de calcul distribué de garantir en même temps (c'est-à-dire de manière synchrone) les trois contraintes suivantes :

- **Consistance :** (ou Cohérence des données) : tous les nœuds du système voient exactement les mêmes données au même moment.
- **Disponibilité :** une garantie d'exécution de toutes les requêtes reçu avec succès au bout d'un temps fini.
- **Tolérance au partitionnement :** aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (ou encore : en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome).

D'après ce théorème, un système de calcul distribué ne peut garantir au même temps que deux de ces contraintes mais pas les trois[53], donc, les concepteurs sont obligé à faire un choix entre ces trois propriétés. Toutefois comme les défaillances de communication sont inévitables dans un cluster de plusieurs machines, la tolérance aux partitions devient primordiale et doit être obligatoirement assurée [54]. Pour le reste, les concepteurs choisissent soit d'assurer une disponibilité élevée au profit de lacunes dans la cohérence des données ou bien d'assurer la consistance au détriment de la disponibilité.

2.3.2 Modeles de stockage

Le stockage des données est une des missions principales d'un système gérant le Big Data, il existe trois solutions de stockage, chacune adoptant un modèle particulier [49].

2.3.2.1 Systemes a base d'objets

Cela consiste à stocker les données dans des champs $BLOB$ d'un $SGBDR$ traditionnels, généralement utiliser pour les données fréquemment lues, mais rarement mises à jour. Ils garantissent leur durabilité et une grande disponibilité sans fournir d'aspects structurels qui pourraient servir aux requêtes complexes. Ces solutions sont donc inadéquates pour des opérations d'analyse ou d'extraction des connaissances, voici quelques fournisseurs cloud qui proposent ce genre de systèmes :Google Cloud Storage, Azure BLOBS, Amazon S3… [49].

2.3.2.2 Systemes bases sur des modeles NoSQL

Le NoSql désigne les bases de données qui ne sont pas fondées sur l'architecture classique des bases de données relationnelles. Développé à l'origine pour gérer du big data, l'utilisation de base de données NoSQL a explosée

depuis quelques années. Cette nouvelle génération de systèmes est divisée en quatre catégories de bases de données [5].

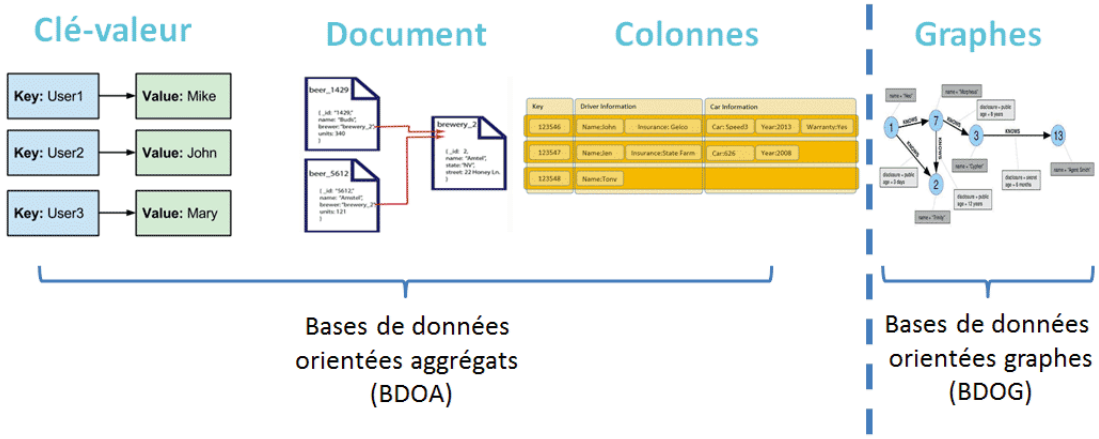

FIG.2.3: Categories de bases de données NoSQL [6].

- **Clé-Valeur :** il s'agit d'un modèle de données simple où des données sont associées à une clé formant un objet identifié de manière unique par cette dernière [5], autrement dit,c'est une clé plus un BLOB (dans lequel on peut mettre : nombre, date, texte, XML, photo, vidéo, structure objet).

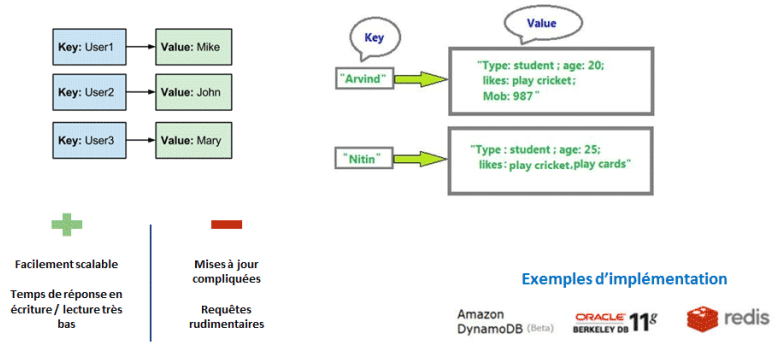

FIG.2.4: Exemple de bases de données NoSQL de type Cle-Valeur [6].

- **Orientées colonnes :** ressemble aux bases de données relationnelles, car les données sont sauvegardées sous forme de ligne avec des colonnes, mais se distingue par le fait que le nombre de colonnes peut varier d'une

ligne à une l'autre.

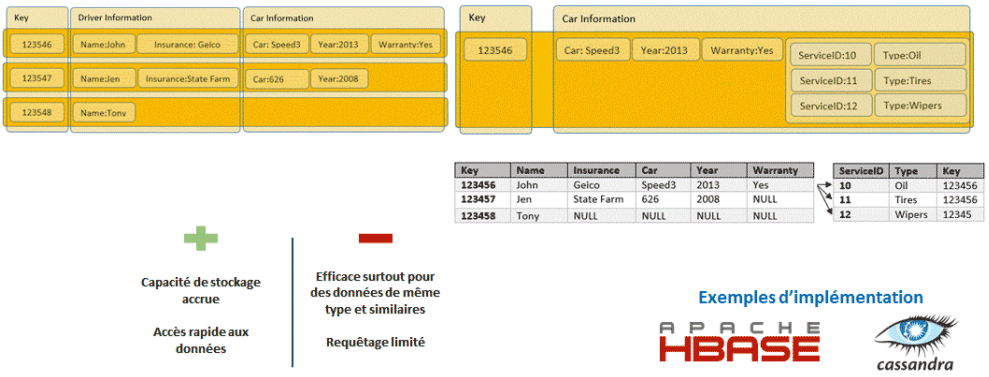

FIG.2.5: Exemple de bases de données NoSQL de type Orientees colonnes [6].

- **Orientées documents :** dans cette catégorie, des objets dits « documents » stockent les données sous forme d'attributs où chaque attribut peut être un autre document offrant ainsi une structure récursive[5].

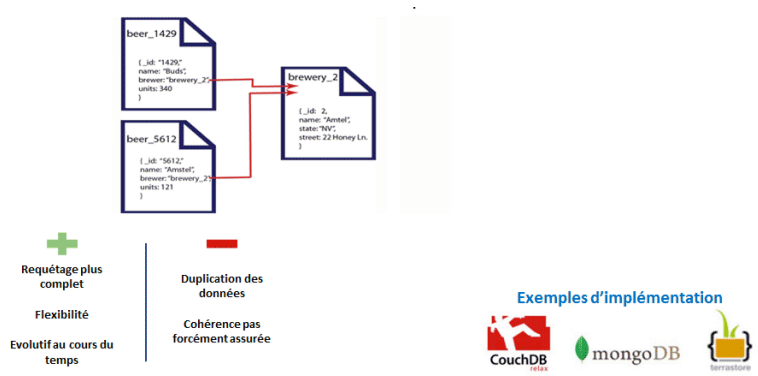

FIG.2.6: Exemple de bases de données NoSQL de type Orientees documents [6].

- **Orientées graphes :** basées sur la théorie des graphes, autrement dit cette catégorie offre un modèle basé sur des graphes avec des nœuds, arêtes et leurs propriétés respectives permettent la manipulation des

données avec un haut niveau d'interdépendance et assure la traverser rapide des données complexes en parcourant les relations entre les nœuds.

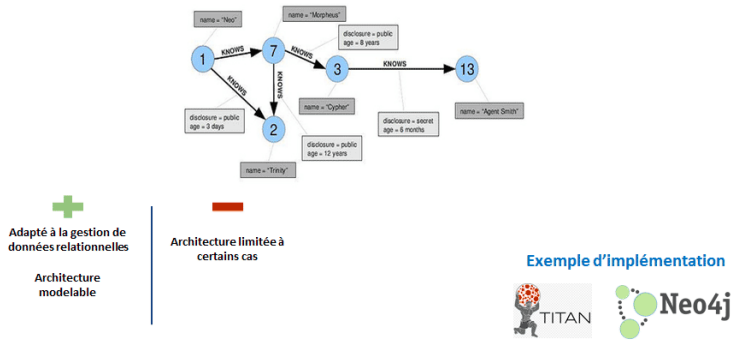

FIG.2.7: Exemple de bases de données NoSQL de type Orientees graphes [6].

Voici quelques exemples d'implémentation présenter dans la figure 2.8 :

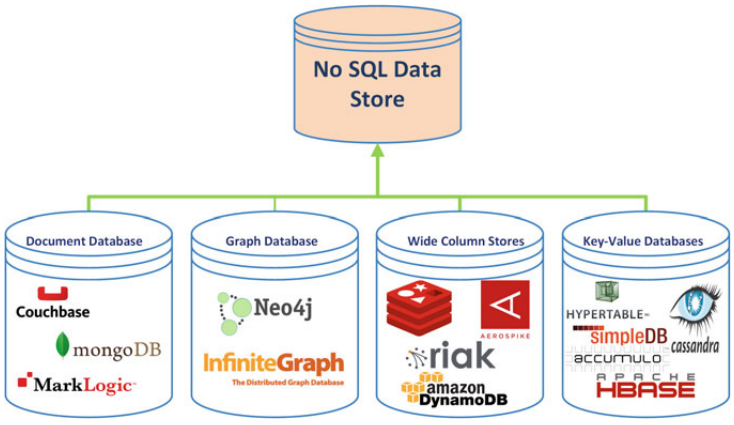

FIG.2.8: Exemple dimplémentation [5]

Les bases de données non relationnelle permet une scalabilité massive atteignant l'échelle des pétaoctets [49] et une haute disponibilité au détriment d'une perte en consistance comme le stipule le théorème CAP. Grace a ses caractéristiques, le NoSql devient le candidat idéal pour gérer les services Web dans les réseaux sociaux ou les sites de e-commerce qui regroupent des millions d'utilisateurs effectuant des opérations à tout moment [54].

2.3.2.3 Systemes de fichiers distribues

C'est un moyen d'unifier les fichiers situés sur des ordinateurs différents en un seul espace nom. Le système de fichiers distribués facilite la construction d'un affichage hiérarchique unique regroupant des serveurs de fichiers multiples et des partages de fichiers sur votre ordinateur [55]. c'est un système permettant d'accéder à une arborescence de fichiers ne résidant pas sur la machine locale mais sur d'autres machines du réseau (les serveurs de fichiers) tout en utilisant la même syntaxe que pour accéder au système de fichiers local. Qu'il s'agisse de fichiers locaux ou de fichiers accédés à travers le réseau, les requêtes auprès du système d'exploitation telles que open(), read() ou chdir() sont fonctionnellement équivalentes [56].

Par rapport aux modèles de stockage, on a opter pour l'utilisation d'un système de fichiers distribués, le apache HDFS (Hadoop Distributed File System).

2.3.3 Modeles de traitement

À mesure que des ensembles de données plus vastes et plus complexes émergent, les méthodes traditionnelles de traitement s'avèrent obsolètes à cause de leurs poursuits d'une haute consistance et la tolérance de panne. Selon la théorie de CAP, il est difficile de garantir la disponibilité et l'évolutivité. De nouveaux paradigmes ont été mis en place qui s'attaquent au volume du Big Data en parallélisant les traitements sur les parties des données. Ces techniques assurent l'adaptation à la taille des données en étendant le nombre de processus afin d'assurer la propriété de scalabilité. Il existe deux principaux paradigmes actuellement, MapReduce et la méthode par Workflows (ou flux de travail). Les systèmes traitant du Big Data sont classifiés selon qu'ils adoptent l'un ou l'autre de ces paradigmes [49].

2.3.3.1 Modele MapReduce

C'est un modèle de programmation parallèle proposé par Google en 2004 [57], adapté spécialement au traitement de quantités massive de données tout en permettant la scalabilité. Les programmes qui adoptent ce modèle sont automatiquement parallélisés et exécutés sur des clusters (grappes) d'ordinateurs. MapReduce assure le partitionnement et le plan d'exécution des programmes tout en gérant les inhérentes pannes informatiques et l'indisponibilité. Une application qui adopte ce paradigme peut traiter plusieurs téraoctets de données et exploite plusieurs milliers de machines. Ce paradigme consiste à exécuter de manière séquentielle les deux fonctions principales :

map et Reduce.

Les données en entrée sont partitionnées, ensuite la fonction Map est appliquée en parallèle à chacune des partitions. Un exemple de fonction Map peut être de compter les mots fréquemment utilisés dans chaque partition. Ses résultats sont appelés résultats intermédiaires et sont assignés à des processus exécutant la fonction Reduce dits Reducer. Ainsi, chacun de ces processus reçoit en entrée un ou plusieurs résultats intermédiaires et exécute un ensemble d'opérations, typiquement, un tri ou une fusion, et produit un résultat. Les résultats de toute la procédure sont ceux délivrés par les processus Reducer [49] . La figure 2.9 résume le fonctionnement du paradigme.

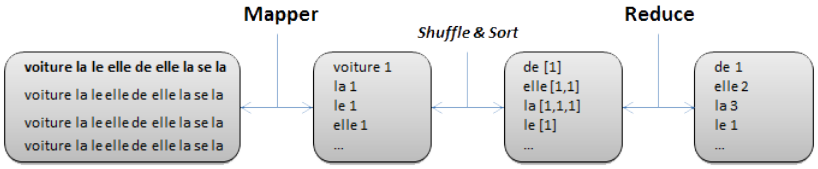

FIG.2.9: étapes de fonctionnement du paradigme MapReduce [7].

En résumé, il suffit de partitionner l'ensemble de données en entrée selon le nombre de processus mis à disposition de l'utilisateur qui s'assurera de personnaliser les fonctions Map et Reduce selon ses besoins [49].

2.3.3.2 Modele des Workflows

Un des points fort de MapReduce est sa simplicité, mais cette même propriété fait abstraction pour certaines applications où l'on a besoin d'exprimer plus précisément les interdépendances entre les tâches et le flot de données. Les Workflows présentent une manière abstraite de schématiser l'évolution des données et des opérations [49].

2.4 Hadoop

Le framework Apache Hadoop assure le traitement distribué de grands ensembles de données à partir de grappes d'ordinateurs utilisant des modèles de programmation simples. Il est conçu pour passer de serveurs simples à des milliers de machines, chacune offrant un calcul et un stockage local. Plutôt que de se baser sur le matériel pour offrir une haute disponibilité, la bibliothèque elle-même est conçue pour détecter et gérer les pannes sur la

couche d'application, fournissant ainsi un service hautement disponible sur un cluster d'ordinateurs, chacun pouvant être sujet à des pannes.

Hadoop est une solution Open Source gérée par Apache software Fundation. Aujourd'hui,c'est la principale solution de stockage et de traitement réparti, essentiellement conçu pour faciliter les analyses dites massives, c'est à dire traitant très rapidement un très grand nombre de données.

hadoop se base sur deux composants principaux :

2.4.1 Le système de fichiers distribué Hadoop

HDFS, le système de fichiers distribué Hadoop est un système de fichiers distribué conçu dans le but de contenir de très grandes masse de données (téraoctets ou même petabytes) avec un accès à haut débit à cette information. Les fichiers sont stockés de manière redondante sur plusieurs machines pour assurer leur durabilité à une défaillance et une grande disponibilité à des applications parallèles [58]
Hadoop repose sur de nombreux concepts proposés dans les systèmes de fichier classique comme Fat pour windows ou Ext de linux par exemple : la notion de blocs (la plus petite unité que l'unité de stockage peut gérer), les métadonnées qui permettent de retrouver les blocs à partir d'un nom de fichier, les droits ou encore l'arborescence des répertoires.
Cependant, HDFS se diffère d'un système de fichiers classique par les caractéristiques suivantes :

- HDFS assure une portabilité et peut être déployé sur différents systèmes d'exploitation [7].
- HDFS permet de faire des copies de blocs avec un nombre de copie configurable.au moment de l'écriture chaque bloque est répliqué sur plusieurs nœuds, pour la phase de lecture, si un bloc est indisponible sur un nœud, des copies de ce bloc seront disponibles sur d'autres nœuds [7].
- Sur un système classique, la taille du disque est généralement la limite globale d'utilisation, par contre dans un système distribué comme HDFS, chaque nœud d'un cluster correspond à un sous-ensemble du volume global de données du cluster. Pour augmenter la taille global, il suffira d'ajouter de nouveaux nœuds [7].
- La taille des blocs est fixé a 64 Mo par défaut. Cependant il est possible de la modifier (128 Mo, 256 Mo, 512 Mo ou 1 Go), alors que sur des systèmes classiques, la taille est généralement de 4 Ko, l'intérêt de cela, c'est la réduction du temps d'accès à un bloc [7].

2.4.1.1 Namenode

Le nameNode est le service central d'un système de fichiers HDFS. Il permet de conserver l'arborescence des répertoires de tous les fichiers dans le système de fichiers, et indique où, dans le cluster, les données du fichier sont conservées. Il ne stocke pas les données de ces fichiers eux-mêmes [59].
Les applications clientes sollicitent le NomeNode chaque fois qu'elles souhaitent localiser un fichier ou lorsqu'ils veulent ajouter / copier / déplacer / supprimer un fichier. Le NomeNode répond aux demandes réussites en renvoyant une liste de serveurs DataNode pertinents où les données existent [59].

Ces métadonnées, sont stockées physiquement sur le disque système dans deux types de fichiers spécifiques edits et fsimage, à part la position des blocs dans les Datanodes qui se reconstruit à chaque démarrage du Namenode dans un mode appelé safe mode. au moment du safe mode, l'écriture sur HDFS est impossible, le nameNode charge les deux fichiers edits et fsimag et attend la reponse des DataNode sur la position des blocs [7].
Pendant le safe mode, l'écriture sur HDFS est impossible, le Namenode charge les fichiers edits et fsimage et attend le retour des Datanodes sur la position des blocs. Des que les opérations sont réalisées, le safe mode est relâché et l'accès en écriture est de nouveau autorisé. La durée du safe mode est proportionnelle au nombre de fichier a traiter [7].

2.4.1.2 Secondary Namenode

Ce servie consiste à vérifier périodiquement l'état du NameNode principal et de copier les métadonnées via les fichiers edits et fsimage. Dans le cas ou le NameNode principal n'est plus disponible le Namenode secondaire prend sa place [7].

2.4.1.2.1 Remarque: Sans le NameNode, il n'y a pas moyen de pouvoir extraire les blocs d'un fichier donné [7].

2.4.1.3 Datanode

DataNode est constitué d'un ensemble de bloque de données, le DataNode est sollicité par les Namenodes au moment des opérations de lecture et d'écriture. Lors de la lecture, les Datenodes transmettent au client les blocs correspondant au fichier à transmettre. Dans le cas de l'écriture, les Datanodes retourne l'emplacement des blocs fraîchement créés. Comme il est sollicités lors de l'initialisation du Namenode et aussi de manière périodique,

afin de retourner la liste des blocs stockés [7].

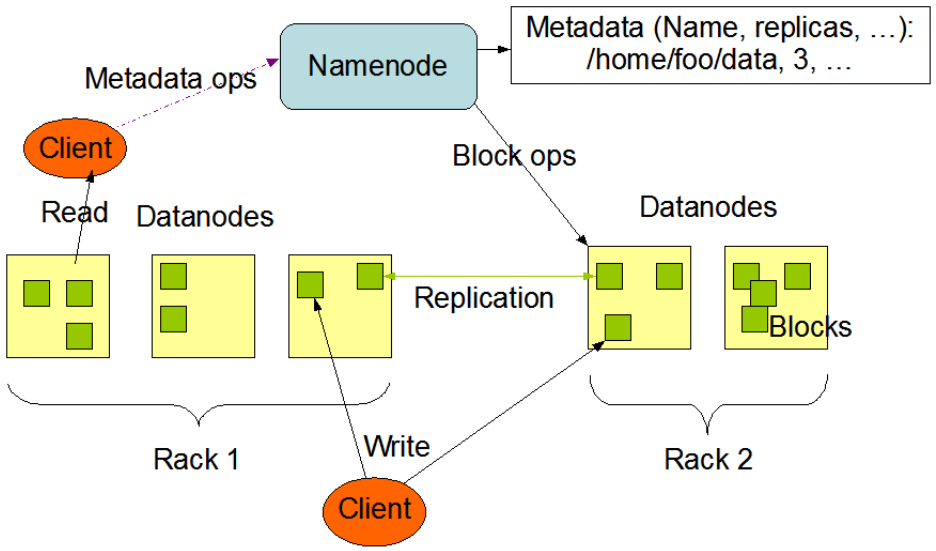

FIG.2.10: Architecture d'un HDFS[8].

2.4.2 MapReduce dans Hadoop

La figure 2.4.2 représente l'architecture générale du model MapReduce dans hadoop.

Voici quelques terminologies avant de rentrer dans l'exécution des jobs MapReduce dans Hadoop.

Job MapReduce : est une unité de travail, exécutée par le client qui se compose de :

- un fichier de données à traiter (Input file),
- un programme MapReduce,
- des informations de configuration (Métadonnées).

Le cluster exécute le job MapReduce en divisant le programme MapReduce en deux tâches : les tâches Map et les tâches Reduce [10].

Le cluster Hadoop est contrôlé par deux type de processus : le jobtracker et un ensemble de tasktrackers [10].

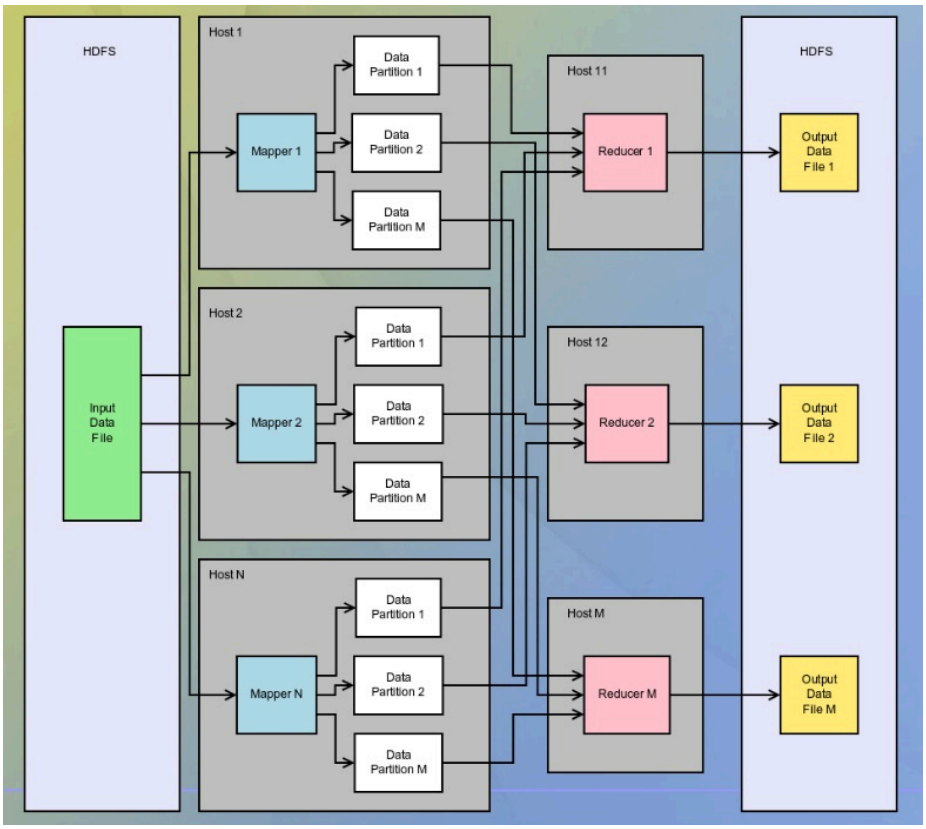

FIG.2.11: Architecture générale du model MapReduce [9].

- *Jobtracker :* c'est un processus qui est démarré au niveau du nœud de référence (le NameNode), il a pour but la coordinantion de tous les jobs qui s'exécutent sur le cluster, il gère les ressources du cluster et planifie les tâches à exécuter sur les tasktrackers. Le jobtracker reçoit des rapports d'avancement de la part des tasktrackers, et garde une copie du progrès général de chaque job. Si une tâche échoue, le jobtracker peut le replanifier sur un tasktracker différent [10].

- *Tasktracker :* c'est un processus qui est démarré au niveau des nœuds de données (les Datanode), afin de traiter le programme MapReduce, en exécutant les tâches Map ou Reduce et envoie des rapports d'avancement au jobtracker [10]. La figure 2.12 résume le jobtracker et les tasktrackers dans le cluster.

  Le jobtracker désigne le nœud de référence et le processus maître qui y est démarré, tandis que le Tasktracker désigne un nœud de données et

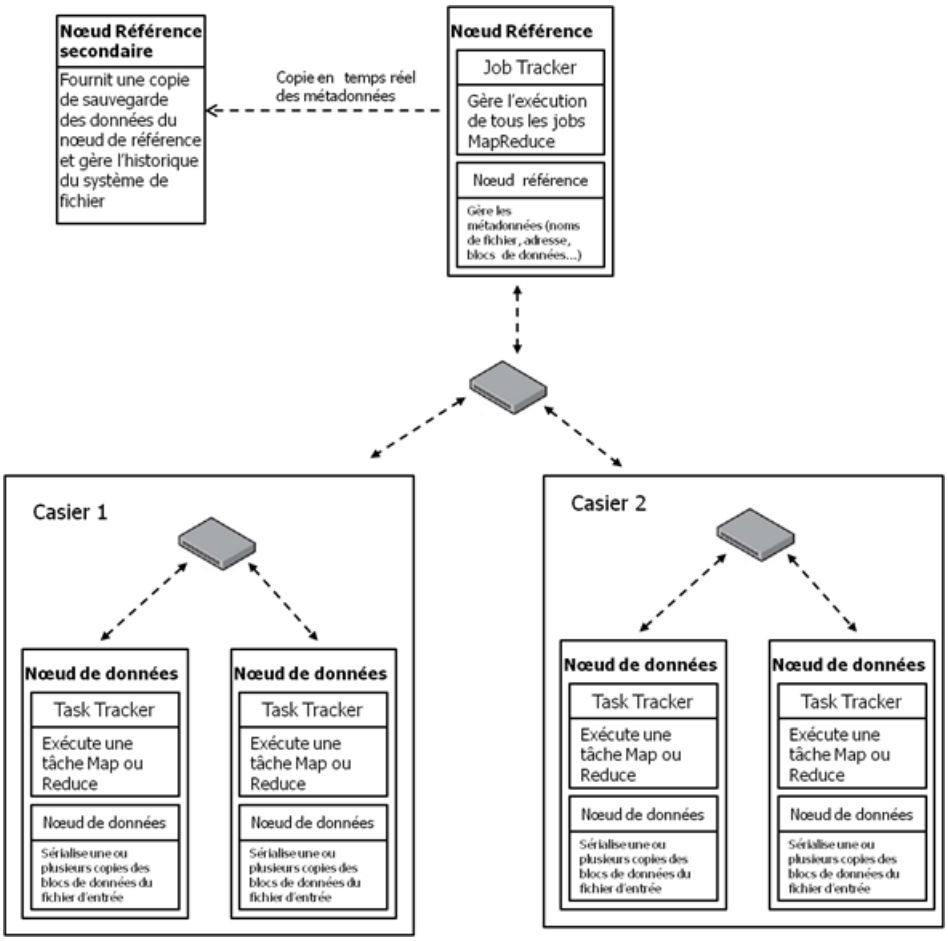

FIG.2.12: Architecture du HDFS [10]

le process esclave qui y est démarré. La figure 2.12 illustre le rapport entre le jobtracker et les tasktrackers dans le cluster.[10]

2.4.2.1 Détails d'exécution du modèle MapReduce dans Hadoop

L'exécuter du Job MapReduce qui est écrit par l'utilisateur, ce fait en 7 étapes comme le montre la figure 2.13 :

1. La première étape est la configuration du Job MapReduce par l'utilisateur en spécifiant :

   - La fonction map,
   - La fonction Reduce,

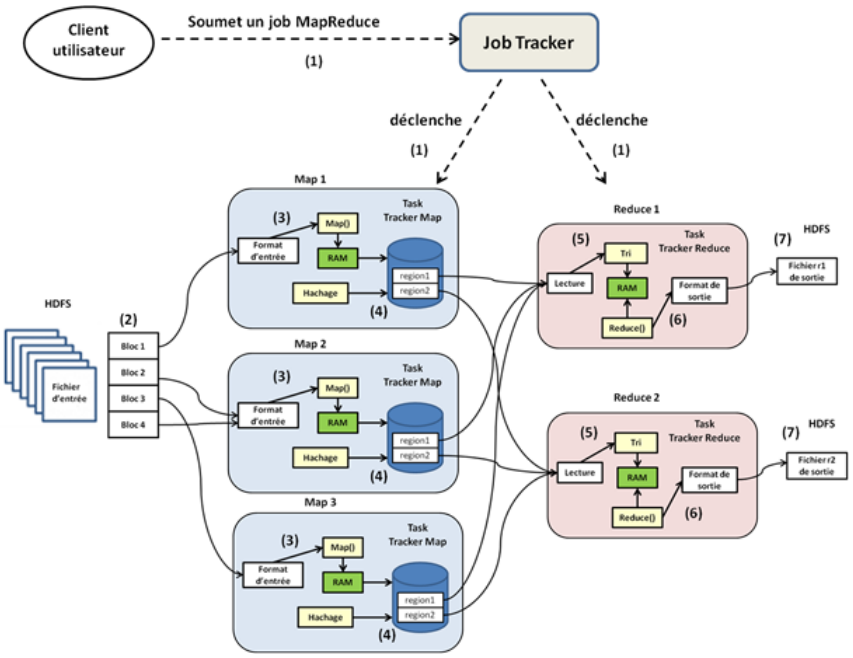

FIG.2.13: étapes d'exécution d'un job MapReduce dans un cluster Hadoop [10].

- Le nombre de tâches Reduce (r),
- Le format de lecture du fichier d'entrée,
- Le format de sortie des r fichiers Reduce,
- La taille des blocs du fichier d'entrée (par defaut 64 Mo),
- Le facteur de réplication (par default 3 copies).

Une fois la configuration terminée et l'exécution du job déclenchée, le jobtracker démarre les r tasktrackers qui vont effectuer les r tâches Reduce que l'utilisateur a spécifiées ;

2. La deuxième étape consiste à découper le fichier d'entrée en M blocs de taille fixe (spécifié à la première étape). Ensuite, le HDFS réplique ces blocs selon le facteur de réplication, et les distribue de façon redondante dans des nœuds différents dans le cluster. La division du fichier d'entrée en bloque de taille fixe nous permet de répartir de façon équilibrée la charge de traitement parallèle entre les nœuds du cluster, ce qui va permettre l'achèvement du traitement a peu près au même temps dans l'ensemble des nœuds du cluster [10] ;

3. La troisième étape c'est le déclenchement de M tasktrackers par le jobtracker sur les M nœuds de données dans lesquels ont été répartis les M blocs du fichier d'entrée, afin d'exécuter les tâches Map, soit un tasktracker Map pour chaque bloc de fichier. Chaque tasktracker lit le contenu du bloc de ficher par rapport au format d'entrée spécifié par l'utilisateur, le transforme par le processus de hachage défini dans la fonction Map en paires de clés/valeurs. Ce processus de hachage s'effectue en mémoire locale du nœud [10] ;

4. La quatrième étape consiste é sérialiser périodiquement les paires de clés/valeurs dans un fichier sur le disque dur local de chaque nœud, par la suite ce fichier est partitionné en r régions (correspondant aux r tâches Reduce spécifiées par l'utilisateur) par une fonction de hachage qui va assigner à chaque région une clé qui correspond à la tâche Reduce à laquelle elle a été assignée. Les informations sur la localisation de ces régions sont transmises au jobtracker, qui fait suivre ces informations aux r tasktrackers qui vont effectuer les tâches Reduce [10] ;

5. Lorsque les r tasktrackers Reduce ont notifiés des informations de localisation, ils utilisent des appels de procédures distantes (protocole RPC) pour lire depuis le disque dur des nœuds sur lesquels les tâches Map se sont exécutées, les régions des fichiers Map leur correspondant. Ensuite, ils les trient par clé. Notez au passage que le tri s'effectue en mode batch dans la mémoire du tasktracker Reduce. Si les données sont trop volumineuses, alors cette étape peut augmenter de façon significative le temps total d'exécution du job [10] ;

6. Les tasktrackers Reduce itèrent à travers toutes les données triées et pour chaque clé unique rencontrée, ils la passent avec sa valeur à la fonction Reduce écrite par l'utilisateur. Les résultats du traitement de la fonction Reduce sont alors sérialisés dans le fichier ri (avec i l'indice de la tâche Reduce) selon le format de sortie spécifié par l'utilisateur. Cette fois-ci, les fichiers ne sont pas sérialisés dans le disque dur du nœud tasktracker, mais dans le HDFS, ceci pour des raisons de résilience (tolérance aux pannes) [10] ;

7. Le job s'achève là, à ce stade, les r fichiers Reduce sont disponibles et Hadoop applique en fonction de la demande de l'utilisateur, soit un « Print écran », soit leur chargement dans un SGBD, soit alors leur passage comme fichiers d'entrée à un autre job MapReduce [10].

2.5 Conclusion

Dans ce chapitre , nous avons passé en revue les notions fondamentales autour du Big Data. Les défis auxquels on est confronté, par la suite, on a exposé le réel problème amené par les Big Data et aussi les solutions afin de parvenir à exploiter ses grosses masses de données, puis on a présenté le framework Hadoop qui va assurer l'exécution de notre application basée sur le modèle de programmation mapreduce.

Dans le chapitre suivant on va présenter une conception basée sur une des technologies du Big Data le *Mapreduce* afin de pouvoir extraire des itemsets à partir des données imparfaites dites évidentielles.

Chapitre 3

Solution proposée

3.1 Introduction

Dans ce chapitre, nous proposons une conception basée sur le modèle MapReduce afin de permettre une exécution distribuée parallèle sur plusieurs nœuds d'un algorithme d'extraction d'itemsets fréquents à partir d'une base de données évidentielles. Dans un premier temps on va détailler la définition des données évidentielles, puis on exposa le déroulement de notre application, la deuxième étape est la réduction de notre base de donnée evidentielles avec l application de la théorie rough-set, en troisième lieu on va extraire les itemsets fréquents à partir de cette dernière.

3.2 Architecture général

FIG.3.1: Architecture générale.

Notre application se compose de deux phases, la première consiste a réduire notre base de données evidentielles en appliquant le théorème de rough-set, la deuxieume phase permet d'extraire les itemsets fréquents à partir de notre base de données réduite en calculent le support précis proposé par Samet [60].

3.3 Données évidentielles

Une base de données évidentielles permet le stockage de données incertaines et imprécises qui sont modélisées à l'aide de la théorie de Dempster-Shafer. Une base de données évidentielles notée *EDB* contient n attributs et d lignes. Chaque attribut $i(1 \leqslant i \leqslant n)$ a un domaine θ_i de valeurs discrètes. La valeur de la colonne i pour la ligne j est dite valeur évidentielle et notée V_{ij} [18].

3.3.1 Valeur évidentielle

Soit V_{ij} une valeur évidentielle qui correspond à la colonne i et à la ligne j. V_{ij} correspond à un corps d'évidence composé du cadre de discernement θ_i (le domaine de la colonne i), de l'ensemble F_{ij} des éléments focaux et de la fonction de masse m_{ij} qui est définie comme suit [18] :

$$m_{ij} : 2^{\theta_i} \rightarrow [0\ 1] \qquad avec : m_{ij}(\theta) = 0 \qquad et \qquad \sum_{x \subseteq \theta_i} m_{ij}(x) = 1 \qquad (3.1)$$

Tableau3.1: Exemple dune base de données évidentielle.

	A	B
T01	$\mathrm{A}_{1(0.6)}, A_{2(0.4)}, \theta_{(0.0)}$	$\mathrm{B}_{1(1)}$
T02	$\mathrm{A}_{1(0.2)}, (A_1, A_2)_{(0.2)}, \theta_{(0.6)}$	$\mathrm{B}_{2(0.2)}, \theta_{(0.8)}$
T03	$\mathrm{A}_{1(0.2)}, (A_1, A_3)_{(0.3)}, \theta_{(0.5)}$	$\mathrm{B}_{3(0.5)}, \theta_{(0.5)}$
T04	$\mathrm{A}_{1(0.7)}, \theta_{(0.3)}$	$\mathrm{B}_{4(1)}$

Le tableau 3.1 représente un exemple d'une base de données évidentielle avec plusieurs types d'imperfections, selon [18],les données imparfaites peuvent être modélisées par des données évidentielles comme suit :

- **Données parfaites :** c'est quand ce corps d'évidence inclut exactement un élément focal qui est singleton, et dont la masse est par conséquent égale à l'unité, nous parlons de données parfaites [18] [3], dans le tableau 3.1 la valeur de la colonne B est parfaite dans les deux transactions (la première et la quatrième).
- **Probabilistes :** Quand le corps d'évidence inclut des éléments focaux singletons, là nous parlons d'une valeur probabiliste [18] [3]. Dans notre exemple, dans le tableau 3.1 la valeur de la colonne A dans la première transaction et la dernière sont probabilistes même chose pour la valeur de l'attribut B dans la deuxième et troisième transactions.

- **Manquantes :** si la valeur d'un attribut i pour une ligne j est manquante, alors la *bba* V_{ij} inclut un seul élément focal qui coïncide avec θ_i, soit le domaine de l'attribut i, avec une masse égale à l'unité [18] [3].
- **évidentielles :** A l'instar de la valeur de l'attribut A dans la troisième transaction du tableau 3.1 qui n'est ni parfaite, ni probabiliste, ni possibiliste, ni manquante [18] [3].
- **Probabilistes :** Ce sont les éléments focaux singletons inclus dans le corps d'évidence, dans notre exemple la valeur de la colonne A dans la premier ligne est probabiliste [18] [3].

3.4 Phase de réduction des données

Selon Nguyen dans [61],la théorie des ensembles d'approximations peuvent interpréter la prise de décisions dans un système d'information, ou les données imparfaites sont définie par :

- **Un system d'information :** définit par la paire $I = (U, A)$ où U est un ensemble fini non vide d'objets appelés l'univers et A est un ensemble fini non vide d'attributs tels que $fa : U \longrightarrow V_a$ pour chaque $a \in A$ La valeur discrète non vide V_a est appelé le domaine de a [62] [3].
- **Une table de décision :** un systéme avec la forme suivante :

$$I = (U, A \cup \{d\}) \qquad et \qquad V_d = \{d_1.....d_k\} \tag{3.2}$$

  indique l'ensemble de valeur de l'attribut de décision.

- **L'espace d'approximation :**
  - *Approximation haute* : permet de décrire les objets qui appartiennent probablement au sous ensemble d'intérêt [3].

$$B^*X = \{x|[x]_B \cap X \neq \emptyset\} \tag{3.3}$$

  - *Approximation basse :* permet de décrire les objets de domaine qui sont connus avec certitude pour appartenir au sous ensemble d'intérêt[3].

$$B_*X = \{x|[x]_B \subset X\} \tag{3.4}$$

– *Région limite :* une décision est dite Rough si la région limite est non vide.

$$BN_B(X) = B^*X - B_*X \tag{3.5}$$

3.4.0.1 Principe de la réduction

La réduction de nos données évidentielles se fera en se basent sur les roughset, afin que ses derniers ne perd pas leur sens. pour cela chaque transaction sera accompagnée par une décision basée sur la probabilité de son existence et de son ignorance, supposons :

- U l'ensemble de nos transactions,
- A l'ensemble de nos objets.

La condition de réduction est définie comme suit : Si la probabilité d'ignorance domine la probabilité d'existence alors la transaction sera éliminée.
Le tableau 3.2 illustre le calcul de la somme des masses P et la somme de l'ignorance I qui représentent respectivement l'existence et l'ignorance des objets par rapport à chaque transaction.

Tableau3.2: Systéme d'information.

	A	B	Décision
T01	$\mathrm{A}_{1(0.5)}, A_{2(0.5)}, \theta_{(0.0)}$	$\mathrm{B}_{1(1)}$	P=2, I=0
T02	$\mathrm{A}_{1(0.2)}, (A_1, A_2)_{(0.1)}, \theta_{(0.7)}$	$\mathrm{B}_{2(0.1)}, \theta_{(0.9)}$	P=0.4, I=1.6
T03	$\mathrm{A}_{1(0.1)}, (A_1, A_3)_{(0.4)}, \theta_{(0.5)}$	$\mathrm{B}_{3(0.5)}, \theta_{(0.5)}$	P=1, I=1
T04	$\mathrm{A}_{1(0.8)}, \theta_{(0.2)}$	$\mathrm{B}_{4(1)}$	P=1.8, I=0.2
T05	$\mathrm{A}_{1(0.2)}, (A_1, A_2)_{(0.1)}, A_{3(0.2)}, \theta_{(0.5)}$	$\mathrm{B}_{1(0.3)}, B_{2(0.1)}, \theta_{(0.6)}$	P=0.9, I=1.1
T06	$\mathrm{A}_{1(0.3)}, \theta_{(0.7)}$	$\mathrm{B}_{1(0.3)}, B_{2(0.2)}, \theta_{(0.5)}$	P=0.8, I=1.2

- P : représente la probabilité d'existence des objets j pour une transaction T_i.
- I : représente la probabilité d'ignorance des objets j pour une transaction T_i.

Cette phase a pour but, la prise de décision en ce qui concerne nos transaction et la construction d'un ensemble d'approximation, comme le montre la tableau 3.3.

Tableau3.3: Table de décision.

	A	B	Décision
T01	$A_{1(0.5)}, A_{2(0.5)}, \theta_{(0.0)}$	$B_{1(1)}$	Non
T02	$A_{1(0.2)}, (A_1, A_2)_{(0.1)}, \theta_{(0.7)}$	$B_{2(0.1)}, \theta_{(0.9)}$	Oui
T03	$A_{1(0.1)}, (A_1, A_3)_{(0.4)}, \theta_{(0.5)}$	$B_{3(0.5)}, \theta_{(0.5)}$	Rough
T04	$A_{1(0.8)}, \theta_{(0.2)}$	$B_{4(1)}$	Non
T05	$A_{1(0.2)}, (A_1, A_2)_{(0.1)}, A_{3(0.2)}, \theta_{(0.5)}$	$B_{1(0.3)}, B_{2(0.1)}, \theta_{(0.6)}$	Oui
T06	$A_{1(0.3)}, \theta_{(0.7)}$	$B_{1(0.3)}, B_{2(0.2)}, \theta_{(0.2)}$	Oui

La table de décision complète illustrée par le tableau 3.3, nous permet la construction des ensembles d'approximation afin de réduire les données évidentielles.

- *Approximation haute :* $B^*X = \{T02, T03, T05, T06\}$
- *Approximation basse :* $B_*X = \{T02, T05, T06\}$
- *Région limite :* $BN_B(X) = B^*X - B_*X = \{T03\}$

Tableau3.4: Données evidentielles reduites.

	A	B
T01	$A_{1(0.5)}, A_{2(0.5)}, \Theta_{(0.0)}$	$B_{1(1)}, \Theta_{(0.0)}$
T03	$A_{1(0.1)}, (A_2, A_3)_{(0.4)}, \Theta_{(0.5)}$	$B_{3(0.5)}, \Theta_{(0.5)}$
T04	$A_{1(0.8)}, \Theta_{(0.2)}$	$B_{4(1)}$

Le tableau 3.4 représente notre base de données evidentielles réduite après l'application du théorème des ensembles d'approximations en éliminant les données qui appartiennent à l'ensemble d'approximation basées, sans pour autant causer une perte d'information.

3.4.0.2 Implémentation de la réduction des données

la phase de réduction se compose de deux fonction principales le mapper et le reducer,

- *La fonction map* reçoit en entrée un couple de <Clef,Valeur>, la clef est un entier qui design le numéro de la ligne du ficher d'entré, la valeur est de type texte contenant les données de la ligne, le corps de la fonction map permet de calculer la valeur de P et I puis envoi en sortie le couplet <Clef1,Valeur1>, $Clef1$ pour le numero de la ligne et la $valeur1$ pour les données (Valeur) plus la valeur du P et du I.

- *La fonction Reduce* Reçoit comme entrées, les sortis de la fonction map <Clef1,Valeur1>, le corps de cette fonction permet de comparé les deux valeur P et I, la ligne avec un P supérieur au I sera renvoyé vers la sortie comme résultat.

3.5 Phase d'extraction des itemsets fréquents

Selon A.Samet [60] le support précis nous permet d'avoir une plus grande fiabilité par rapport aux travaux existants, c'est pourquoi nous avons décidé de l'utiliser pour l'extraction des items sets fréquents à partir de données évidentielles. Soit,

- EDB Une base de données évidentielles,
- Un itemset $X = x_1 \times ... \times x_n$, le produit de plusieurs items (éléments focaux) $x_i (1 \leqslant i \leqslant n)$ du cadre de discernement θ.

Le degré de présence de l'item x_i dans une transaction T_j est mesuré de la façon suivante :

$$Pr : 2^{\theta} \rightarrow [0\ 1] \tag{3.6}$$

tel que :

$$Pr(x_i) = \sum_{x \subseteq \theta_i} \frac{|x_i \bigcap x|}{|x|} \times m(x) \qquad \forall x_i \in 2^{\theta_i} \tag{3.7}$$

Le $Pr(.)$ permet de calculé une probabilité sur une fonction de masse unique. La mesure Pr est égale à la probabilité pianistique de $x \in \theta_i$. Le support précis d'un itemset est calculé comme suit :

$$Support^{Pr}_{T_j}(X) = \prod_{X_i \in \theta_i, i \in [1...n]} Pr(X_i) \tag{3.8}$$

$$Support_{EDB}(X) = \frac{1}{d} \sum_{j=1}^{d} Support^{Pr}_{T_j}(X) \tag{3.9}$$

3.5.0.1 Exemple

Soit, le tableau 3.5 représente une base de données évidentielles, θ_i un ensemble fini non-vide incluant n hypothèses , dans notre exemple $\theta_A = \{\{A_1\}, \{A_2\}, \{A_1, A_2\}\}$, et $\theta_B = \{\{B_1\}, \{B_2\}, \{B_1, B_2\}\}$,

Tableau3.5: Exemple de base de données évidentielle EDB

	A	B
T01	$m_{11}(A_1) = 0.7\,;\, m_{11}(\Theta_A) = 0.3$	$m_{21}(B_1) = 0.4\,;\, m_{21}(B_2) = 0.2\,;\, m_{21}(\Theta_B) = 0.4$
T02	$m_{12}(A_1) = 0.3\,;\, m_{12}(\Theta_A) = 0.7$	$m_{22}(B_1) = 1$

La premier étape est de calculer les Pr de tout les éléments de l'ensemble θ_A et θ_A,

$$Pr(A_1) = \sum_{A_1 \subseteq \theta_A} \frac{|A_1 \bigcap x|}{|x|} \times m(x) \qquad \forall A_1 \in 2^{\theta_A} \tag{3.10}$$

$$Pr(A_1) = \frac{|A_1 \bigcap A_1|}{|A_1|} \times m(A_1) + \frac{|A_1 \bigcap A_2|}{|A_2|} \times m(A_2) + \frac{|A_1 \bigcap \{A_1, A_2\}|}{|\{A_1, A_2\}|} \times m(\{A_1, A_2\}) \tag{3.11}$$

$$Pr(A_1) = \frac{1}{1} \times 0.7 + \frac{0}{1} \times 0 + \frac{1}{2} \times 0.3 \tag{3.12}$$

$$Pr(A_1) = 0.85 \tag{3.13}$$

3.5.0.1.1 Remarque: Si la masse d'un itmes n'existe pas dans la EDB, comme par exemple le A_2, la masse de cette dernière est égale à 0, mais si l'items en question est combiné avec un autre items connu, par exemple : $\{A_1, A_2\}$ la masse de ce dernier est égale à l'ignorance Θ , dans notre exemple $m(\{A_1, A_2\}) = 0.3$.

le Tableau 3.6 contient les valeurs des Pr de notre exemple tableau 3.5.

Le calcul d'un support d'un itemset $A1 \times B1$ est de la manière suivante :

$$Support^{Pr}_{T_1}(A_1 \times B_1) = Pr(A_1) \times Pr(B_1) = 0.51 \tag{3.14}$$

$$Support^{Pr}_{T_2}(A_1 \times B_1) = Pr(A_1) \times Pr(B_1) = 0.35 \tag{3.15}$$

$$Support_{EDB}(A_1 \times B_1) = \frac{1}{2} \sum_{j=1}^{2} Support^{Pr}_{T_j}(A_1 \times B_1) = \frac{0.51 + 0.35}{2} = 0.43 \tag{3.16}$$

Tableau3.6: Le tableau Pr déduit à partir de la base évidentielle EDB présentée dans le tableau 3.5

Transaction	Support transactionnel
T01	$Pr(A_1) = 0.85$ $Pr(A_2) = 0.15$ $Pr(A_\Theta) = 1$ $Pr(B_1) = 0.6$ $Pr(B_2) = 0.4$ $Pr(B_\Theta) = 1$
T02	$Pr(A_1) = 0.35$ $Pr(A_2) = 0.65$ $Pr(A_\Theta) = 1$ $Pr(B_1) = 1$ $Pr(B_2) = 0$ $Pr(B_\Theta) = 1$

3.5.1 Scenario

- **Entrées** : Une base de données evidentielles, support minimum.
- **étape 1** : Création d'un Job MapReduce.
- **étape 2** : La division de la base de donnée sur le nombre de Nœud de notre architecture n, Figure 3.2 partitionnement.
- **étape 3** : L'exécution de la fonction Map par les n Nœuds de notre architecture avec pour but le calcul de la valeur du produit cartésien entre les items, Figure 3.3.
- **étape 4** : L'exécution de la fonction Combinaison par les n Nœuds de notre architecture avec pour but de combiner les résultats de la fonction Map Figure 3.4.
- **étape 5** : L'exécution de la fonction Reduce par les n Nœuds de notre architecture va nous permettre d'extraire les itemsets frequents par rapport au support minimum et retourner que les résultats supérieur à ce dernier, Figure 3.5.
- **étape 6** : La copie des résultats de l'exécution de la fonction Reduce dans notre serveur hadoop, Figure 3.6 résultats.
- **Sortie** : Les itemset fréquents de notre EDB, Figure 3.6.

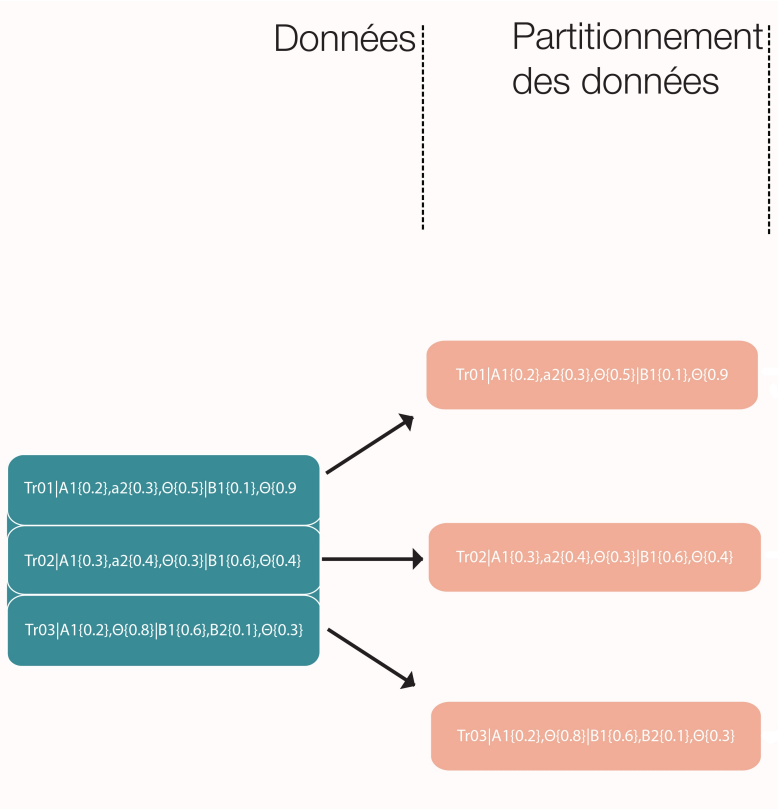

FIG.3.2: Partitionnement de la base de données.

3.5.2 Déroulement détaillé

L'exécution générale d'un programme MapReduce sous hadoop est détaillée dans le chapitre deux, section Hadoop, sous-section MapReduce dans Hadoop, dans cette sous-section nous exposerons en détaille que l'exécution des trois fonctions, Map, Combiner et Reduce.

1. **Map** :La fonction map reçoit en entré un coupler de $< clef, valeur >$ la $clef$ représente le numéro de la ligne de notre base de données, la $valeur$ représente la ligne elle mème, par exemple pour la EDB représentée par le tableau 3.1 l'entrée $< clef, valeur >$ de notre map est $< 1, T01|A_{1(0.6)}, A_{2(0.4)}, \theta_{(0.0)}|B_{1(1)} >$, la clef et de type $IntWritable$ et la valeur de type $texte$, l'exécution de la fonction map se fait comme suite :

   (a) Génère le cadre de discernement θ dans notre exemple pour l'objet $A\,\theta_A = \{\{A_1\}, \{A_2\}, \{A_1, A_2\}\}$. pour $B\,\theta_B = \{\{B_1\}, \{B_2\}, \{B_1, B_2\}\}$,

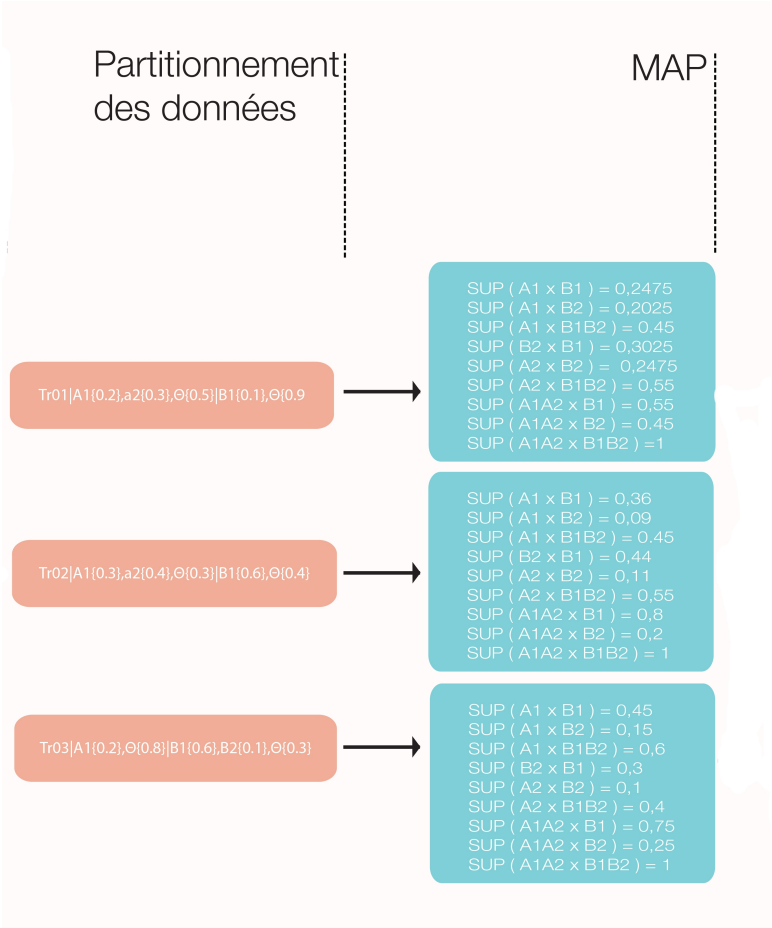

FIG.3.3: Mapping.

(b) Extraire les données de la $< valeur >$,

(c) Calculer le Pr pour chaque élément dans θ_i,

(d) Calculer le $Support_{T_j}^{Pr}(X)$,

(e) Retourner un couple de clef valeur $< Support_{T_j}^{Pr}(X)$,résultat du calcul du $Support_{T_j}^{Pr}(X) >$ le $Support_{T_j}^{Pr}(X)$ est de type $text$ et sa valeur et de type $FloatWritable$.

3.5.2.0.1 Remarque:

- la fonction map reçoit en entrée une seul ligne de notre base de donnée et non toute la base de donnée, donc on a pas accès aux

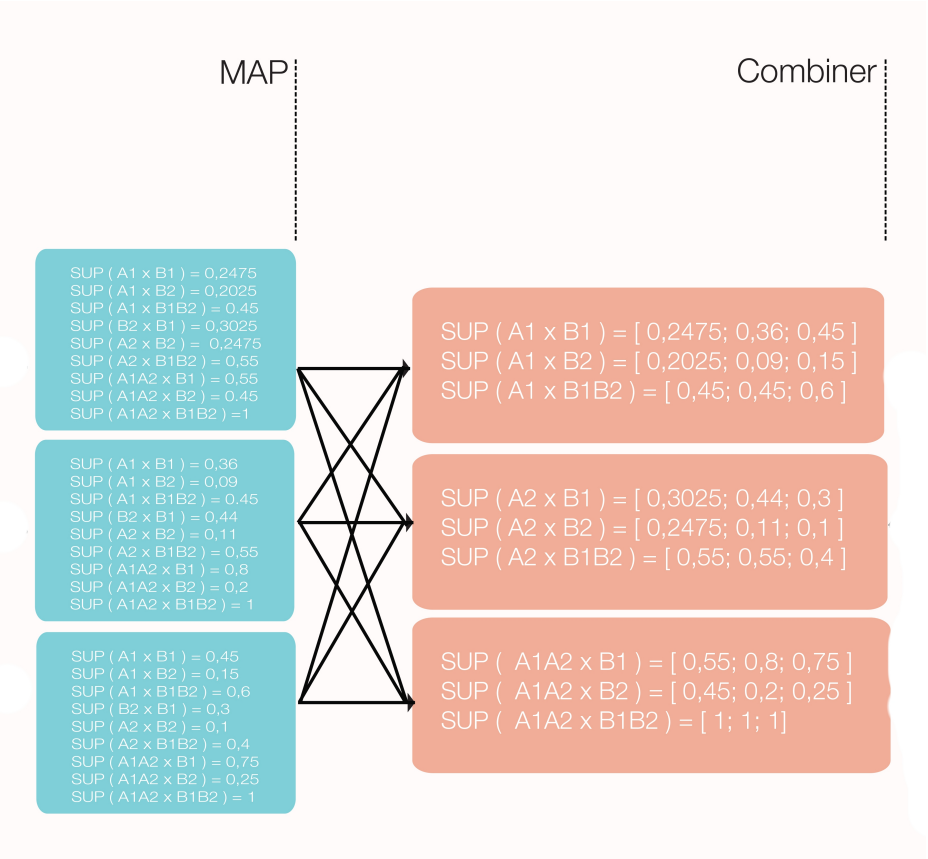

FIG.3.4: Combinaison.

autres données a par la ligne en entré.

- les types $FloatWritable$, $IntWritable$ et $Text$ sont des types prédéfinie pour les fonction MapReduce,
- la sortie de la fonction Map doit être l'entrée de la fonction Reducer (même type).

2. le retour se fait pour chaque calcul du $Support^{Pr}_{T_j}(X)$ donc plusieurs $< clef, valeur >$ pour une seul ligne.
3. **Combiner :** C'est une fonction implémenter et exécuter automatiquement par le framework hadoop pour chaque programme basée sur le modèle MapReduce, son rôle est de combiner tout les $valeurs$ d'entrées avec la même $clef$, par exemple on a en entrée :
$< Support^{Pr}_{T_1}(A_1 \times B_1), 0.25 >$

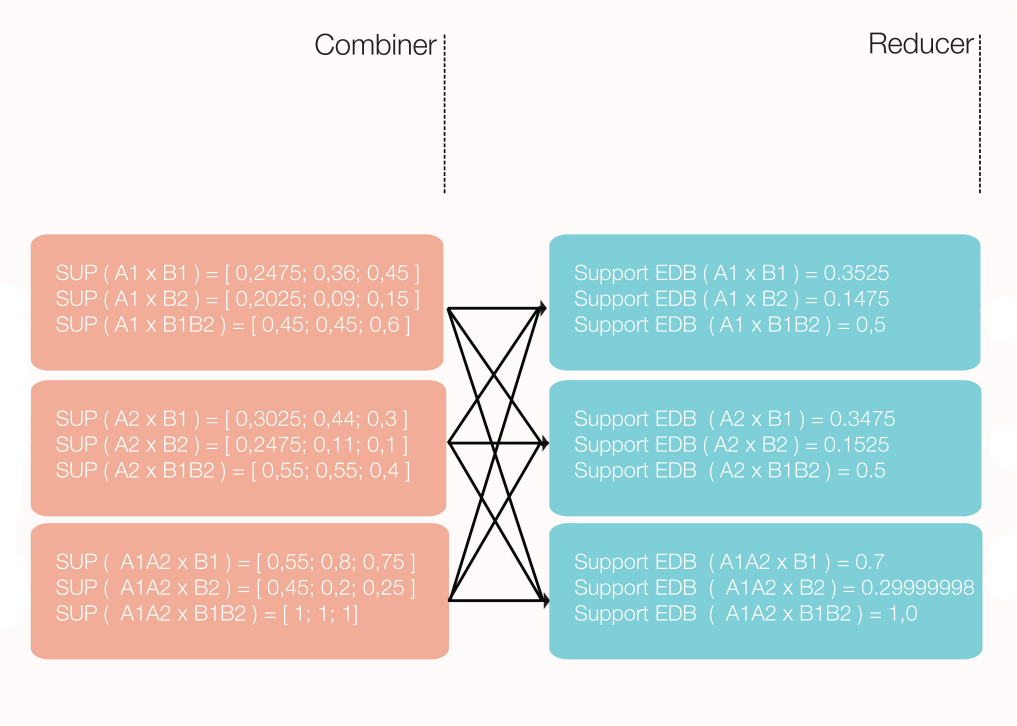

FIG.3.5: Reduce.

$< Support^{Pr}_{T_2}(A_1 \times C_1), 0.28 >$
$< Support^{Pr}_{T_1}(A_1 \times B_1), 0.32 >$
la sortie de la fonction combinée sera :
$< Support^{Pr}_{T_1}(A_1 \times B_1), [0.25, 0.32] >$
$< Support^{Pr}_{T_2}(A_1 \times C_1), [0.28] >$.

4. **Reduce :** cette fonction reçoit en entrée les $< clef, valeur >$ retournée par la fonction combiner, et exécute les étapes suivante

   (a) faire la somme des valeurs de chaque en entré, dans notre exemple l'entrer $< Support^{Pr}_{T_1}(A_1 \times B_1), [0.25, 0.32] >$ devient $< Support^{Pr}_{T_1}(A_1 \times B_1), [0.57] >$ et l'entré $< Support^{Pr}_{T_2}(A_1 \times C_1), [0.28] >$reste tel qu'elle est.

   (b) deviser chaque résultat de la premier étape sur le nombre de transaction total de notre EDB

   (c) comparer si le résultat de la deuxième étape est supérieur au support minimum voulu, si oui, la sortie sera un couple $< Support^{Pr}_{T_j}(X)$,$valeurdusupport >$ sinon y aura rien en sortie.

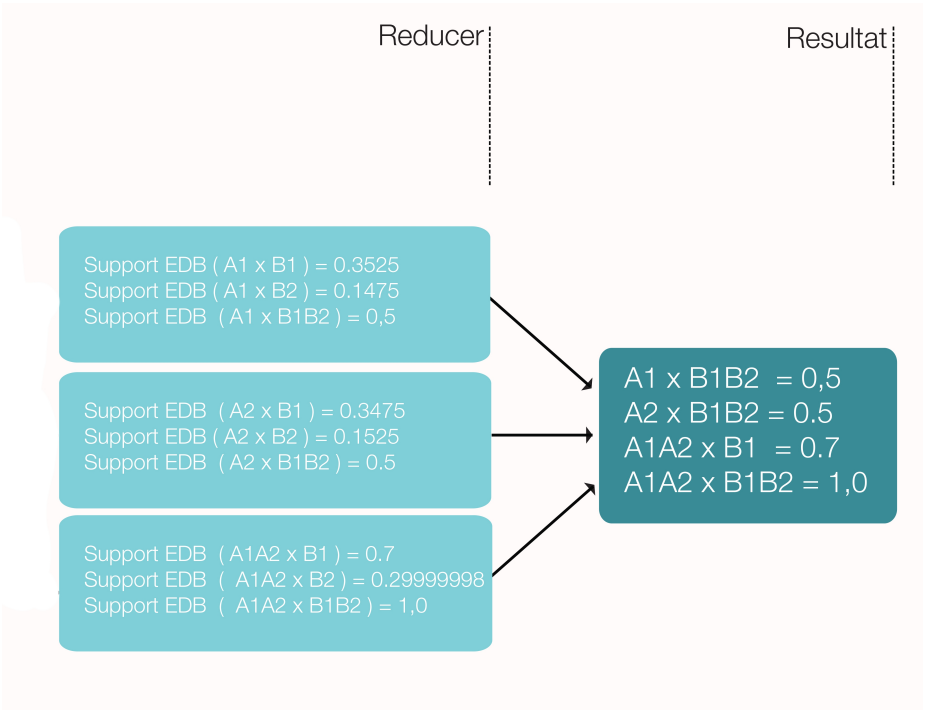

FIG.3.6: Résultat.

le traitement se fait toujours pour chaque ligne seulement.

3.6 Conclusion

Pour le moment on a défini les données évidentielles et introduit notre conception basée sur le modèle MapReduce, tout en détaillant le déroulement de chaque fonction, le chapitre suivant est consacré aux tests et la validation de notre application.

Chapitre 4

Test et Validation

4.1 Introduction

Ce chapitre présente notre environnement de développement, notre implémentation ainsi que le fonctionnement de notre application et les résultats obtenus dans la phase de test.

4.2 Environnement de développement

4.2.1 Environnement matériel

- **Processeur** : Intel® Core™ i5-6200U CPU 2.30GHz.
- **Disque dur** : 1Tr.
- **Mémoire Vive** : 8Go.

4.2.2 Environnement logiciel

- **Système d'exploitation** : Ubuntu 16.04 LTS 64 Bits.
- **Machine virtuelle** : Cloudera CDH 5.12 et une allocation de 4Go de mémoire vive. avec le framwork hadoop installé et configuré, la distribution cloudera nous offre un environnement de développement parfait pour tester des applications MapReduce.
- **Hadoop 2.6** : Afin de permettre l'exécution des applications MapReduce.
- **Jdk 7** : Pour l'exécution de notre programme MapReduce écrit en java.

4.2.3 Configuration minimal

- Hadoop 2.6.
- Jdk 7 ou supérieur.

4.3 Langage de développement:

Les applications MapReduce sont généralement implémentées soit en python, ou en java. Notre choix s'est porté pour Java à cause de la disponibilité de la documentation par rapport à python.

4.4 Exécution de notre application

- Vérifier si le serveur hadoop est allumé avec la commande <hadoop version> si la version hadoop est retournée alors il est allumé, sinon, on doit l'allumer par la commande <star-all.sh>. Généralement dans la distribution cloudra le hadoop est allumé par défaut figure 4.1, mais pour le cas de ubuntu il faut allumer à chaque utilisation.

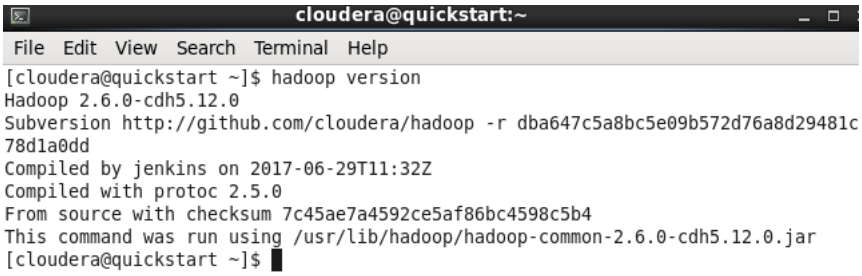

FIG.4.1: Hadoop version

- Création d'un dossier afin de mettre notre fichier qui contient la base de données évidentielles avec la commande <hadoop fs -mkdir /dossierData> figure 4.2, le "/" est le répertoire racine du HDFS, afin de s'assurer que le dossier a bien été créé, il suffit de taper dans l'explorateur internet le lien suivant : http ://quickstart.cloudera :50070/explorer.html/ comme le montre la figure 4.3.

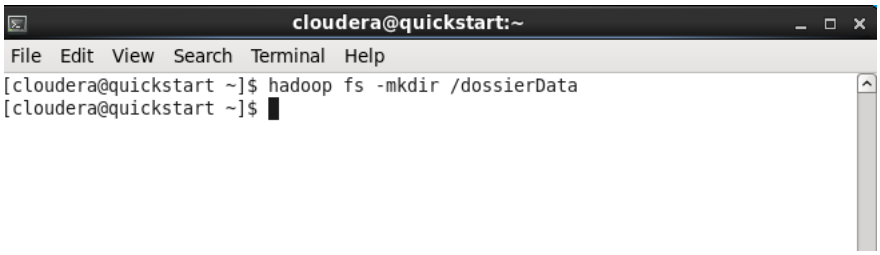

FIG.4.2: Création d'un dossier dans HDFS

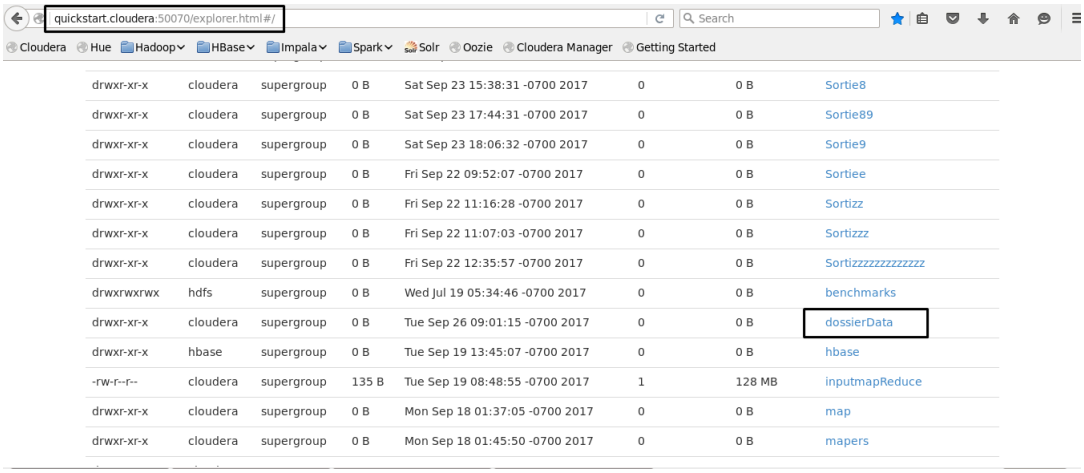

FIG.4.3: Confirmation de la création d'un dossier dans HDFS

- Pour le moment, il faut copier notre base de données evidentielles dans le dossier qu'on a créé avec la commande < hadoop fs -put /emplacementDuDataDentre /dossierData > comme le montre la figure 4.4 et pour confirmer on utilise toujours la même fenêtrehttp ://quickstart.cloudera :50070/explorer.html/ figure 4.5 .

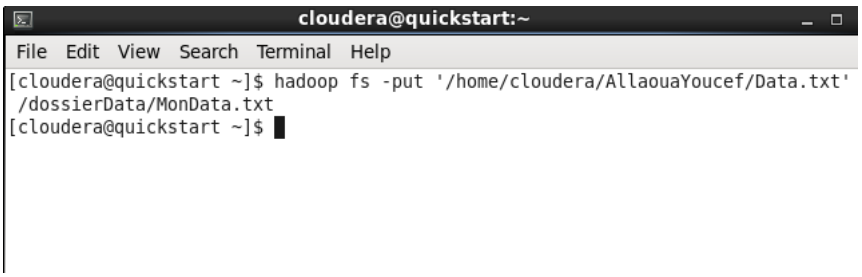

FIG.4.4: Copier le data dans le dossier du HDFS

- Maintenant que tout est en place, on peut exécuter notre fichier .jar,

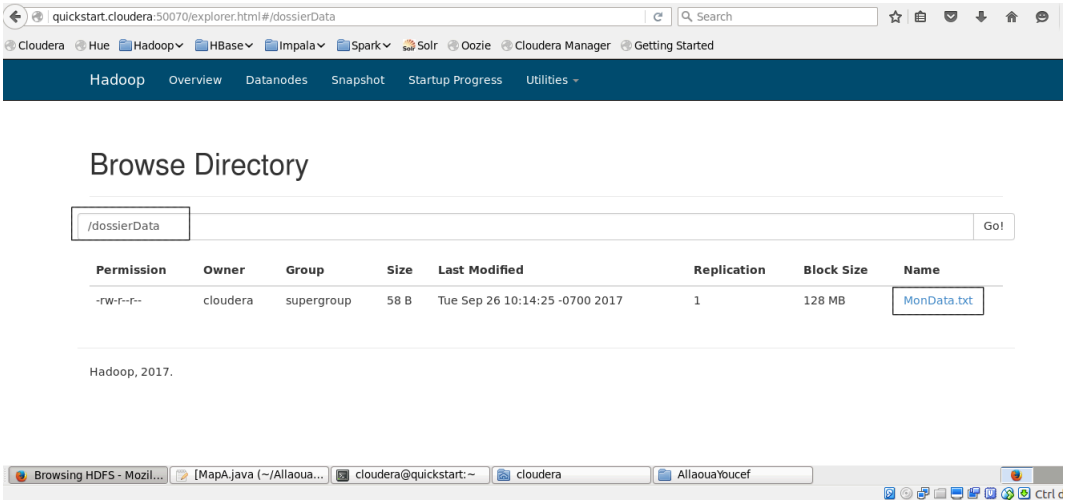

FIG.4.5: Confirmation de la copie du data dans le dossier du HDFS

pour se faire il faut se positionner la ou y a notre fichier .jar et exécuter la commande suivante <Hadoop jar EifMR.Jar /dossierData/MonData.txt /Sortie11> figure 4.6 EifMR.jar, c'est notre fichier jar a exécuté le /dossierData/MonData.txt est l'emplacement de notre fichier data dans le HDFS, /Sortie11 désigne l'emplacement du résultat dans le HDFS, afin d'accéder à notre résultat il suffit d'utiliser la fenêtre dans l'explorateur comme le montre la figure 4.7.

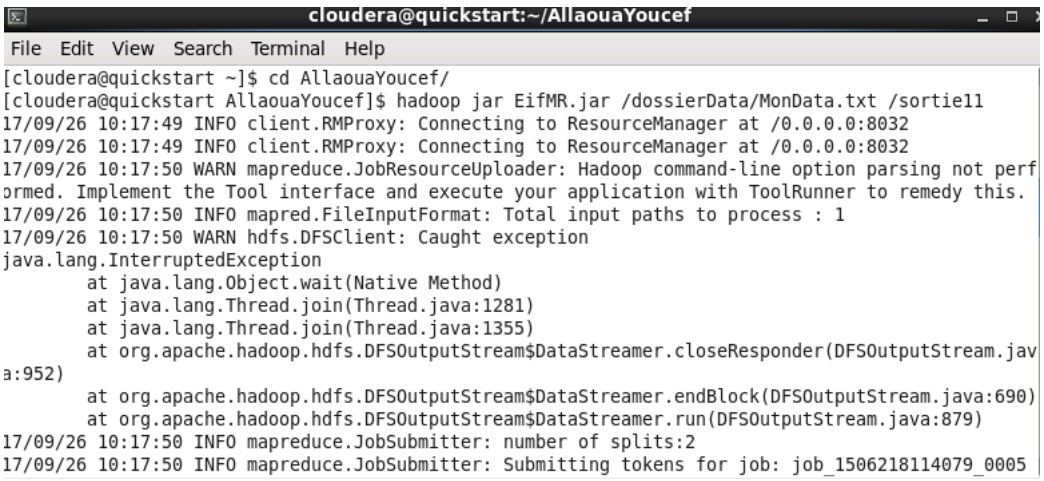

FIG.4.6: Exécution de notre application MapReduce

- la figure 4.8 est l'ensemble des itemsets fréquents avec un support minimum supérieur à 0,3.

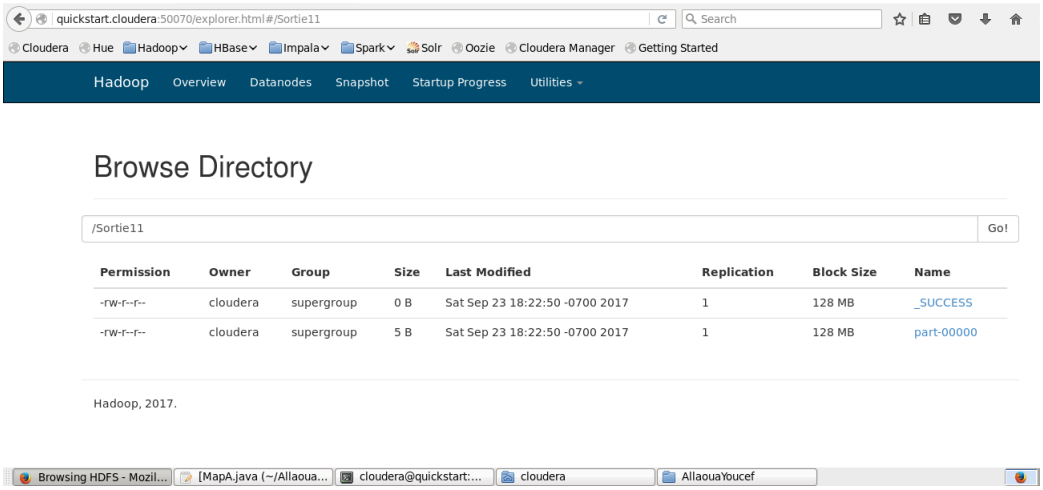

FIG.4.7: Confirmation de la copier du data dans le dossier du HDFS

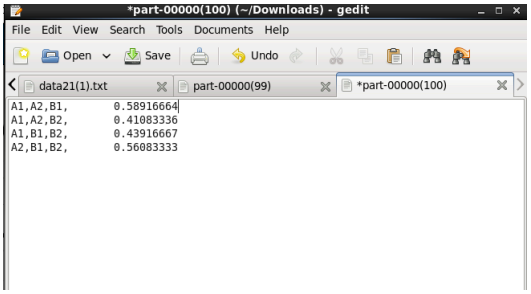

FIG.4.8: Liste des itemset fréquent

4.5 Conclusion

Dans ce chapitre on a exposé notre travaille et fait une démonstration du fonctionnement de notre application, mais par faute de moyen on à pas pu la tester dans un environnement a plusieurs nœuds afin de montrer la réelle différence de traitement entre l'exécution en parallèle et l'exécution séquentielle.

Conclusion général

Conclusion

L'extraction des itemsets à partir de données est une technique qui adopte un ensemble d'outils d'analyse de données pour déceler des motifs fréquents qui peuvent être utilisés dans la prise de décision, ou pour faire des prédictions valides. Cependant les données sont probablement extraites du monde réel et dans la plupart des cas imparfaits, de nombreux algorithmes et modèles ont été proposés pour gérer cette imperfection mais le problème est le temps de traitement ou d'exécution de ses algorithme dans le cas ou, on dispose d'une grande base de données.

Notre travail s'est limité aux base de données évidentielles. Des bases de données peuvent contenir des données incertaines et imprécises au même temps, grâce au calcul du support précis proposé par Samet[60] et afin de minimiser le temps de traitement, on a adopté une conception et implémentation de cette solution pour une exécution en parallèle basée sur le framework hadoop.

Perspectives

Les perspectives de ce travail préliminaire sont multiples. Tout d'abord,

un autre Job qui utilise comme entré, le résultat du JobExtraction pour généré l'ensemble des éléments focaux, un troisième Job afin de générer la liste des itemsets et l'utilisation de cette dernière et l'ensemble des éléments focaux comme des constantes globales pour notre Job principal qui permet l'extraction des itemsets fréquents.

Bibliographie

[1] E. Desjardin, O. Nocent, and C. de Runz, "Prise en compte de l'imperfection des connaissances depuis la saisie des données jusqu'à la restitution 3d," *established by : Mauro Cristofani and Riccardo Francovich*, 2012.

[2] "Extraction des itemsets à l'aide du composant frequent itemsets.." https ://eric.univ-lyon2.fr, Mis a jour le 3 octobre 2011, consulté le 13 aout 2017.

[3] L. K. BENATTOU Ibtissem, "Extraction des itemsets fréquents à partir des données imparfaites," Master's thesis, Université BLIDA -1-, 2017.

[4] J. Gantz and D. Reinsel, "Idc–iview," 2010.

[5] S. Sakr, *Big Data 2.0 Processing Systems : A Survey.* Springer, 2016.

[6] "Qu'est-ce qu'une base nosql ? les cas datastax et mongodb." http ://www.digora.com/fr/blog/definition-base-nosql-datastax-mongodb, Consulté le 12 aout 2017.

[7] M. Baron, "Introduction à apache hadoop." http ://mbaron.developpez.com/tutoriels/bigdata/hadoop/introduction-hdfs-map-reduce/images/mapreducesteps.png, Mis en ligne le 15 avril 2014, consulté le 12 aout 2017.

[8] D. Borthakur, "Hdfs architecture guide." https ://hadoop.apache.org/, Mis a jour le 08 Avril 2013, consulté le 13 aout 2017.

[9] A. Iacono, "Mapreduce by examples." https ://fr.slideshare.net/andreaiacono/mapreduce-34478449, Article paru 9 mai 2014, consulté le 12 aout 2017.

[10] J. CHOKOGOUE, "Hadoop : la nouvelle infrastructure de gestion de données." http ://juvenal-chokogoue.developpez.com/tutoriels/hadoop-fonctionnement, Mis en ligne le 26 avril 2017 , consulté le 14 aout 2017.

[11] D. Siegwald, "Le data mining, c'est quoi ?." http ://www.base-plus.fr/qqoqccp-du-data-mining/, Article paru 20 septembre 2017, Consulté le 27 septembre 2017.

[12] "Hours of video uploaded to youtube every minute as of july 2015." https ://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/, Article paru juillet 2015, Consulté le 19 aout 2017.

[13] D. D. Z. Keith Dawson, "A conversation on the role of big data in marketing and customer service." https ://www.mediapost.com/publications/article/173109/a-conversation-on-the-role-of-big-data-in-marketin.html, Article paru le 25 avril 2012, Consulté le 19 aout 2017.

[14] "Big data statistics facts for 2017." https ://www.waterfordtechnologies.com/big-data-interesting-facts/, Article paru 22 fevrier 2017, Consulté le 19 aout 2017.

[15] B. WIRE, "Ventana research survey shows more organizations use hadoop to perform data mining and in-depth analytics." http ://news.sys-con.com/node/1920943, Article paru 26 Juin 2011, Consulté le 19 aout 2017.

[16] D. Vellante, "Revisited : The rapid growth in unstructured data." http ://wikibon.org/blog/unstructured-data/, Article paru 17 avril 2010, Consulté le 19 aout 2017.

[17] B. B. J. B. R. D. C. R. A. H. B. James Manyika, Michael Chui, "Big data : The next frontier for innovation, competition, and productivity." https ://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation, Article paru mai 2011, Consulté le 19 aout 2017.

[18] M. A. B. Tobji and B. B. Yaghlane, "Extraction des itemsets fréquents à partir de données évidentielles : application à une base de données éducationnelles.," 2011.

[19] J. W. Kwan, M. J. Lee, A. F. Mack, J. F. Chiu, and R. D. Fernald, "Nonuniform distribution of cell proliferation in the adult teleost retina," *Brain research*, vol. 712, no. 1, pp. 40–44, 1996.

[20] S. Konias, I. Chouvarda, I. Vlahavas, and N. Maglaveras, "A novel approach for incremental uncertainty rule generation from databases with

missing values handling : application to dynamic medical databases," *Medical informatics and the Internet in medicine*, vol. 30, no. 3, pp. 211–225, 2005.

[21] Y. D. Lo, F. M. Lun, K. A. Chan, N. B. Tsui, K. C. Chong, T. K. Lau, T. Y. Leung, B. C. Zee, C. R. Cantor, and R. W. Chiu, "Digital pcr for the molecular detection of fetal chromosomal aneuploidy," *Proceedings of the National Academy of Sciences*, vol. 104, no. 32, pp. 13116–13121, 2007.

[22] Y. Wang, S. Wang, and K. K. Lai, "A new fuzzy support vector machine to evaluate credit risk," *IEEE Transactions on Fuzzy Systems*, vol. 13, no. 6, pp. 820–831, 2005.

[23] G. Shafer *et al.*, *A mathematical theory of evidence*, vol. 1. Princeton university press Princeton, 1976.

[24] B. Belgacem, "Extraction de connaissances à partir de données incomplètes et imprécises," Master's thesis, UNIVERSITE DE M'SILA, 2011.

[25] P. F. Fisher, "Models of uncertainty in spatial data," *Geographical information systems*, vol. 1, pp. 191–205, 1999.

[26] C. De Runz, *Imperfection, temps et espace : modélisation, analyse et visualisation dans un SIG archéologique*. PhD thesis, Université de Reims-Champagne Ardenne, 2008.

[27] S. B. Amor, A. Guitouni, and J.-M. Martel, "A synthesis of information imperfection representations for decision aid," *INFOR : Information Systems and Operational Research*, vol. 53, no. 2, pp. 68–77, 2015.

[28] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338 – 353, 1965.

[29] C. Negoita, L. Zadeh, and H. Zimmermann, "Fuzzy sets as a basis for a theory of possibility," *Fuzzy sets and systems*, vol. 1, no. 3-28, pp. 61–72, 1978.

[30] A. P. Dempster, "Upper and lower probabilities generated by a random closed interval," *The Annals of Mathematical Statistics*, pp. 957–966, 1968.

[31] Z. Pawlak, "Rough sets," *International Journal of Parallel Programming*, vol. 11, no. 5, pp. 341–356, 1982.

[32] Z. Pawlak, "Imprecise categories, approximations and rough sets," in *Rough Sets*, pp. 9–32, Springer, 1991.

[33] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Acm sigmod record*, vol. 22, pp. 207–216, ACM, 1993.

[34] G. Piatetsky-Shapiro, "Discovery, analysis and presentation of strong rules," *Knowledge discovery in databases*, pp. 229–248, 1991.

[35] R. Agrawal, R. Srikant, *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, pp. 487–499, 1994.

[36] khaled tannir, "L'algorithme a-priori." http ://blog.khaledtannir.net/2011/05/apriori/.WarLrJ-YHWU, Article paru 03 mai 2011, consulté le 02 septembre 2017.

[37] R. Bahloul, M. B. Ayed, and A. M. Alimi, "Vers une méthode d'évaluation de système interactif d'aide à la décision basé sur le processus d'ecd," in *Atelier : Évaluation des méthodes d'Extraction de Connaissances dans les Données, 10ème Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances, EGC*, 2010.

[38] R. B. Vaughn, J. Farrell, R. Henning, M. Knepper, and K. Fox, "Sensor fusion and automatic vulnerability analysis," in *Proceedings of the 4th international symposium on Information and communication technologies*, pp. 230–235, Trinity College Dublin, 2005.

[39] C. K.-S. Leung, M. A. F. Mateo, and D. A. Brajczuk, "A tree-based approach for frequent pattern mining from uncertain data," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 653–661, Springer, 2008.

[40] C. C. Aggarwal, Y. Li, J. Wang, and J. Wang, "Frequent pattern mining with uncertain data," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 29–38, ACM, 2009.

[41] M. Bach Tobji, B. Ben Yaghlane, and K. Mellouli, "Incremental maintenance of frequent itemsets in evidential databases," *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pp. 457–468, 2009.

[42] K. R. G. Hewawasam, K. Premaratne, M.-L. Shyu, and S. P. Subasingha, "Rule mining and classification in the presence of feature level and class label ambiguities," in *Proc. SPIE*, vol. 5803, pp. 98–107, 2005.

[43] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 16, no. 4, pp. 523–544, 2007.

[44] K. R. Hewawasam, K. Premaratne, and M.-L. Shyu, "Rule mining and classification in a situation assessment application : A belief-theoretic approach for handling data imperfections," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 6, pp. 1446–1459, 2007.

[45] I. Bloch, *Fuzzy Sets and Possibility Theory*, pp. 135–197. ISTE, 2010.

[46] A. Farhat, M. S. Gouider, and L. B. Said, "New algorithm for frequent itemsets mining from evidential data streams," *Procedia Computer Science*, vol. 96, pp. 645–653, 2016.

[47] "Le big data à l'écoute de votre business." https ://www-01.ibm.com/software/fr/data/bigdata/, Consulté le 12 aout 2017.

[48] "Big data." http ://www.futura-sciences.com/tech/definitions/informatique-big-data-15028/, Consulté le 12 aout 2017.

[49] R. Tudoran, *High-performance big data management across cloud data centers*. PhD thesis, ENS Rennes, 2014.

[50] D. laney, "3d data management." https ://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf, Mis en ligne le 06 fevrier 2001, consulté le 12 aout 2017.

[51] *L'Encyclopédie des Big Data 2016*. Electronic Business Group, 2016.

[52] B. Di Martino, D. Kranzlmüller, and J. Dongarra, "Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) : Preface," *Lecture Notes in Computer Science*, vol. 3666, 2005.

[53] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[54] H. Wang, J. Li, H. Zhang, and Y. Zhou, "Benchmarking replication and consistency strategies in cloud serving databases : Hbase and cassandra," in *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pp. 71–82, Springer, 2014.

[55] "Système de fichiers distribué : une vision logique du stockage physique." https ://technet.microsoft.com/fr-fr/library/bb967408.aspx, Mis en ligne dimanche 19 mars 2000, consulté le 12 aout 2017.

[56] M. Ouwehand, "Dfs, système de fichiers distribués du futur." http ://dit-archives.epfl.ch/FI94/8-94-page1.html, Article paru 25 octobre 1994, consulté le 12 aout 2017.

[57] J. Dean and S. Ghemawat, "System and method for efficient large-scale data processing." https ://www.google.com/patents/US7650331, Jan. 19 2010. US Patent 7,650,331.

[58] "The hadoop distributed file system." https ://developer.yahoo.com/hadoop/tutorial/module2.html, Mis en ligne le 20 avril 2005, consulté le 12 aout 2017.

[59] "Hadoop wiki : Namenode." https ://wiki.apache.org/hadoop/NameNode, Mis a jour le 10 septembre 2011, consulté le 12 aout 2017.

[60] A. Samet, *Théorie des fonctions de croyance : application des outils de data mining pour le traitement des données imparfaites.* PhD thesis, Artois, 2014.

[61] D. V. NGUYEN, "Rough set models and knowledge acquisition for imperfect information systems," 2014.

[62] Z. Pawlak, "Rough set theory and its applications to data analysis," *Cybernetics & Systems*, vol. 29, no. 7, pp. 661–688, 1998.