Parallélisation des algorithmes de recherche de vérité

SOUMAHORO FANTA

African Institute for Mathematical Sciences (AIMS), Sénégal
Institut Africaine des Sciences Mathématiques (AIMS), Sénégal

Sous la supervision de : Dr Mouhamadou Lamine BA
Université Alioune Diop de Bambey (UADB), Sénégal

15 mai 2022

Soumission pour l'obtention du Master en Science Mathématique, Option Big Data à AIMS-Sénégal

AIMS | African Institute for Mathematical Sciences SENEGAL

Déclaration

Ces travaux ont été réalisés à AIMS Sénégal dans le respect partiel des exigences d'un Master en Science.

Je déclare par la présente, sauf mention contraire, que ce travail n'a jamais été présenté en totalité ou en partie pour l'attribution d'un diplôme à AIMS Sénégal ou à toute autre université.

Etudiant : SOUMAHORO Fanta 

Remerciements

Je tiens à remercier tous ceux qui ont participé à l'élaboration de ce travail particulièrement :

— tout le personnel administratif de AIMS-Sénégal ;
— Dr Mouhamadou Lamine BA, pour avoir accepté de superviser ce travail et pour ses conseils et apports à la réalisation du présent document ;et
— TOSSOU Osias Nicodème pour son aide.

Dédicaces

En mémoire

de

Mon défunt Père **Amadou SOUMAHORO**

Résumé

La vérification de faits, encore appelée la découverte de la vérité, la recherche de vérité ou fusion de données est un processus qui recherche la vraie valeur d'un élément de donnée du monde réel (propriété ou attribut d'un objet) lorsque des informations contradictoires sont fournies par différentes sources à propos de cet élément. Nous avons plusieurs méthodes qui ont été proposées pour résoudre le problème de vérification de faits pour des applications diverses. Nous pouvons citer comme exemple le *majority voting* et *TruthFinder* qui sont des techniques de référence.

Dans ce travail, nous nous intéressons à la parallélisation des méthodes de recherche de vérité en particulier la parallélisation de l'algorithme **TruthFinder**. En effet les méthodes de découverte de la vérité sont séquentielles et ont du mal à passer à l'échelle en présence de grand volume de données. En informatique, le calcul parallèle consiste en l'exécution simultanée d'une même tâche, partitionnée et adaptée afin de pouvoir être répartie entre plusieurs processeurs en vue de traiter plus rapidement des problèmes plus grands. D'abord Nous faisons une étude approfondie des algorithmes de découverte de vérité en les classifiant en algorithme parallelisable et algorithme non parallelisable. Ensuite nous proposons la version parallèle de notre algorithme TruthFinder, pour ce fait nous utilisons le paradigme MapReduce en vue d'éviter l'augmentation du temps d'exécution et de la consommation de mémoire lorsque la taille des données d'entrée augmente. En fin, nous validons notre parallélisation en exécutant des tests sur les données synthétiques et réelles en vu de vérifier les performances des versions séquentielle et parallèle. Ces tests montrent que les deux versions ont les mêmes performances (accuracy, precision, recall et F1-score) mais concernant le temps d'exécution, la version parallèle de l'algorithme nous donne une réduction exponentielle (99%) et d'au moins 90% pour la consommation en mémoire.

Mots-clés :Vérification des faits, attribut, parallélisation, MapReduce, performance.

Abstract

Fact-checking, also known as truth discovery, truth finding or data fusion, is a process that searches for the true value of a real-world data item (property or attribute of an object) when conflicting information is provided by different sources about that item. We have several methods that have been proposed to solve the fact checking problem for various applications. We can quote as examples the *majority voting* and *TruthFinder* which are reference techniques.

In this work, we are interested in the parallelization of truth finding methods, in particular the parallelization of the algorithm **TruthFinder**. Indeed, truth finding methods are sequential and have difficulties to scale up in the presence of large volume of data. In computer science, parallel computing consists in the simultaneous execution of the same task, partitioned and adapted so that it can be distributed among several processors in order to process larger problems faster. First, we make a thorough study of truth discovery algorithms by classifying them into parallelizable and non-parallelizable algorithms. Then we propose the parallel version of our TruthFinder algorithm, for which we use the MapReduce paradigm in order to avoid the increase in execution time and memory consumption when the size of the input data increases. Finally, we validate our parallelization by running tests on synthetic and real data in order to verify the performance of the sequential and parallel versions. These tests show that both versions have the same performances (accuracy, precision, recall and F1-score) but concerning the execution time, the parallel version of the algorithm gives us an exponential reduction (99%) and of at least 90% for the memory consumption.

Translated with www.DeepL.com/Translator (free version) **Keywords :**Fact checking, attribute, parallelization, MapReduce, performance.

Table des matières

Table des figures

Liste des tableaux

Liste des équations

Glossaire

API : Application Programming Interface . 33

BI : Business Intelligence . 33

DS : Donnée Synthétique . 38

GHz : Gigahertz . 35
Go : Gigaoctets . 35

JDBC : Java DataBase Connectivity . 33
JSON : JavaScript Object Notation) . 33

LCA : Latent Credibility Analysis. 21
LTM : Latent Truth Model. 9

MLE : Maximum Likelihood Estimation. 22

RAM : Random Access Memory (Mémoire vive) . 35
RDD : Resilient Distributed Dataset . 33, 34

SQL : Structured Query Language . 33
SSTF : Semi Supervised Truth Finding . 10, 21

TCD : Taux de Couverture des Données. 44
TD-AC : Truth Discovery with Attribute Clustering. 12, 21
TFAP : Truth Finding with Attribute Partitioning . 21

1. Introduction

Dans ce chapitre nous présentons d'abord le contexte et la motivation de ce travail de mémoire de Master. Ensuite, nous donnons la problématique étudiée et les objectifs visés. Enfin, nous terminons ce chapitre en précisant l'organisation de la suite du document.

1.1 Contexte et motivation

Les réseaux sociaux, les encyclopédies en ligne, les sites de recherche et les sites d'informations sont les premières sources d'information les plus courants de nos jours malgré leur diversité et leur hétérogénéité. Cependant la vérification des informations données par ces source est suivie malgré leur accessibilité facile. En effet nous avons de façon courante les affirmations contradictoires pour un même problème donné, nécessitant ainsi une interrogation à propos de la fiabilité de chacune des sources et de la confiance à accorder à chacune de leurs affirmations. La qualité de l'information étant très importante dans différents domaine de la vie courante, une preuve sur la fiabilité de ces sources et la véracité de l'information est primordiale. Il existe plusieurs méthodes de vérification de la véracité des données provenant de différentes sources. Ces méthodes sont appelées les algorithmes de vérification de faits, les algorithmes découverte de la vérité, etc. La vérification de faits consiste à distinguer les fausses informations de celles vraies en analysant une masse non négligeable de données provenant de sources différentes. Pour plus de détails nous invitons le lecteur à voir le chapitre 3 état de l'art. En effet ces méthodes sont séquentielle et ne passent pas facilement à l'échelle en présence de grand volume de données. Vu ce problème une parallélisation de ces méthodes s'impose.

1.2 Problématique

La difficulté du mécanisme de vérification des faits, le caractère hétérogène des données et la taille des sources causent l'automatisation du processus de vérification des faits. Le processus manuel devient fastidieux et sujet à des erreurs à large échelle. Les algorithmes de vérification des faits sont séquentiels et passent difficilement à l'échelle lorsque la masse de donnée est très élevé. Le temps d'exécution des algorithmes de vérification de faits utilisés dans la pratique doit être largement amélioré de sorte que le résultat produit ne soit pas obsolète. Pour résoudre ce problème nous proposons une parallélisation ces algorithmes.

1.3 Objectifs de ce mémoire

Dans ce travail nous proposons à l'échelle de grands volume de données, une version parallèle des algorithmes de recherche de vérité en utilisant MapReduce sous Apache Spark. Les objectifs spécifiques sont :

— faire une étude approfondie de la parallélisation des algorithmes standards et séquentiels de découverte de vérité ;

— proposer une version parallélisée de l'algorithme de découverte de la vérité **TruthFinder** ; et

— proposer une validation sur des données synthétiques et réelles des performances de la version parallèle proposée par rapport à la version séquentielle, surtout la réduction drastique du temps de calcul.

1.4 Organisation du document

Le reste de ce document est ordonné comme suit. Nous introduisons, dans le chapitre 2, les concepts de base de la recherche de vérité ainsi que la notations utilisées par les algorithmes de découverte de vérité dans ce travail. Le chapitre 3 parlera des algorithmes de recherche de vérité et des algorithmes de recherche de vérité parallèles existants. Le chapitre 4 comportera une introduction au calcul parallèle, une étude approfondie sur les algorithme de recherche de vérité et la version parallèle de l'algorithme TruthFinder. Dans le chapitre 5 nous détaillons les expérimentations effectuées sur des données synthétiques et réelles de notre méthode et l'analyse des différents résultats. Nous concluons ce document dans le chapitre 6 et donnons quelques perspectives.

2. Préliminaires et Notations

Le problème de recherche de vérité est le processus qui consiste à trouver la valeur réelle d'un élément de données du monde réel (propriété ou attribut d'un objet) lorsque différentes sources de données fournissent des informations contradictoires à son sujet. Dans ce chapitre, nous allons présenter les préliminaires liés au problème de recherche de vérité. Dans un premier temps, nous allons définir les concepts clés.

2.1 Préliminaires

Nous définissons les différents concepts qui interviennent dans un problème de recherche de vérité tels que les concepts de source, d'objet, d'attribut, de valeur, d'observation, la confidence d'une valeur et la fiabilité d'une source avec des exemples à l'appui.

Objet (o) : correspond à une entité du monde réel caractérisée par un ou plusieurs attribut(s). On peut le voir comme un lot de questions lié à un domaine par exemple. En considérant l'exemple de la figure 2.1, un objet est une catégorie, par exemple la catégorie Géographie.

Attribut (a) : désigne une propriété d'un objet. Un objet peut avoir plusieurs attributs. En considérant l'exemple de la figure 2.1, un attribut est une question d'une rubrique, par exemple la question Q1 de la catégorie Géographie.

Valeur (v) : correspond à l'information associée à un attribut d'un objet. En d'autre terme une proposition de réponse ou affirmation concernant un attribut d'un objet provenant d'une source. En considérant l'exemple de la figure 2.2, une valeur est une réponse donnée par un postulant par rapport à une question d'une catégorie, par exemple **Afrique** est une valeur par rapport à la question Q1 de la catégorie Géographie.

Source (s) : c'est l'origine d'une observation ou de l'affirmation. Une source émet une proposition à propos de la valeur d'un attribut d'un objet. En considérant l'exemple de la figure

2.2, une source est représentée par un postulant, par exemple le postulant 3.

Observation (c) : c'est une tuple de quatre éléments qui constitue une valeur qu'une source propose pour l'attribut d'un objet. En considérant l'exemple de la figure 2.2, une observation est constituée du 4-tuple suivant : (postulant, catégorie, question, réponse). Ex : *(postulant 3, catégorie Géographie, Q3, Non)*. Q3 est la troisième question de la catégorie Géographie.

Confidence d'une valeur (C_v) : c'est le degré de véracité d'une valeur fournie. Elle représente la confiance accordée à une valeur d'attribut d'objet par rapport à la réalité. Son calcul dépend de chaque algorithme. En considérant l'exemple de la figure 2.2, on peut calculer la confidence d'une valeur en lui attribuant le nombre de fois qu'elle est soutenue par les sources ; ainsi avec *majority voting* les confidences des valeurs **Amérique du Sud** , **Non** et **python** sont **2**, **2**, et **2** respectivement.

Fiabilité d'une source (T_s) : c'est la capacité de la source à fournir des affirmations correctes pour des attributs d'objets donnés du monde réel. La valeur de la fiabilité d'une source peut être déduite de l'expertise de la source dans le domaine, mais dans la plupart des cas c'est une valeur qu'on ne connaît pas à l'avance. Ainsi elle est initialisée à une valeur par défaut en fonction du problème et ensuite mise à jour au cours de l'exécution de l'algorithme de recherche de vérité.

catégorie Géographie	Q1 - Dans quel continent est situé le Brésil ? Q2 - Quelle est la superficie de la Côte d'ivoire ? Q3 - Le Ghana fait-il partie de la CEDEOA ?
catégorie Informatique	Q1- Que signifie RAM ? Q2 - Quel est le langage de progammation le plus utilisé ? Q3 - Qu'affiche ce code python ? *print(6**2)*

TABLE 2.1 – Questions concours

postulant	catégorie	Q1	Q2	Q3
postulant 1	Géographie	Afrique	**322,463 km²**	Non
postulant 2	Géographie	**Amérique du Sud**	233 km²	**Oui**
postulant 3	Géographie	**Amérique du Sud**	320,463 km²	Non
postulant 1	Informatique	Random Access Manager	**Python**	12
postulant 2	Informatique	**Random Access Memory**	Java	**36**
postulant 3	Informatique	Random Allow Memory	**Python**	**36**

TABLE 2.2 – Réponses concours

Nous présentons le résultat de trois postulants (qui représentent les sources) à une compétition portant sur deux catégories (objets) et chaque catégorie contient trois questions (attributs) dans le tableau 2.1. Le tableau 2.2 présente les réponses que chaque postulant a donné par rapport à chaque question.

2.2 Notations

Dans la recherche de la vérité les notations usuellement utilisés sont résumées dans le tableau 2.3.

	Notation
O	ensemble d'objets
A	ensemble d'attributs
A_o	ensemble d'attributs de l'objet o
A_{o-s}	ensemble d'attributs de l'objet o auquel la source s a réagi
AO	ensemble de couple (attributs,objet)
P	ensemble des partitions de A
S	ensemble des sources
S_{a-o}	ensemble des sources réagissant à propos de l'attribut a de l'objet o
S_o	ensemble des sources réagissant à propos d'objet o (quelques soit l'attribut)
S_v	ensemble des sources fournissant la valeur v
$S_{v'}$	ensemble des sources fournissant des valeurs distinctes de la valeur v
V	ensemble des valeurs
V_{a-o}	ensemble des valeurs de l'attribut a de l'objet o
V_{AO_s}	ensemble des valeurs des attributs d'objet fournies par la source s
V_s	ensemble des valeurs fournies par la source s
C_v	probabilité de véracité d'une valeur d'un attribut d'un objet
T_s	niveau de fiabilité d'une source

TABLE 2.3 – Notations

3. État de l'art

La découverte de la vérité consiste a discerner la valeur réelle d'un élément de données lorsque différentes sources donnent des informations contradictoires. L'importance de traiter les conflits d'informations entre ces sources est la motivation de la recherche de la vérité. Dans cet Chapitre Ici nous faisons l'état de l'art des algorithmes de découverte de la vérité.

3.1 Classification des algorithmes de recherche de vérité

Le MajorityVoting est la méthode la plus triviale. C'est une méthodes simple de la découverte de la vérité qui consiste à rassembler toutes les valeurs candidates fournies par les sources pour un attribut particulier d'un objet et la valeur la plus fréquente est accordée a cet attribut. Les autres méthodes s'inspirent de cet algorithme comme nous l'avons dit plus haut. Nous pouvons regrouper les algorithmes de recherche de vérité en quatre (03) grandes familles en fonction de la méthodologie utilisée :

1. méthodes basées sur des modèles probabilistes bayésiens ;
2. méthodes basées sur l'optimisation et
3. les méthodes basées sur partitionnement de données.

3.1.1 Méthodes basées sur des modèles probabilistes bayésiens

Ces méthodes se basent dans la majeur partie des cas sur le vote pour le calcul de la confidence des valeurs ainsi que pour le calcule de la fiabilité des sources. Il faut noter que certaines méthodes y ajoutent la dépendance entre les sources et la similarité entre les valeurs qu'elles donnent par rapport a un attribut d'un objet. Comme algorithmes nous pouvons citer :

— truthFinder ;

— découverte de la vérité par corroboration des informations ;

— modèle de vérité latente ;

— découverte de la vérité par estimation de la vraisemblance maximale ;

— découverte de la vérité avec dépendance des sources et

— analyse de crédibilité latente

3.1.1.1 TruthFinder

Présenté par (Yin et al., 2008), **TruthFinder** est une approche bayésienne.La probabilité que l'attribut d'une entité ait une valeur particulière est une fonction de la fiabilité de la source fournissant la valeur ainsi que des valeurs présentées par d'autres sources. Le modèle est basé sur l'idée qu'une source est digne de confiance si elle fournit fréquemment une valeur correcte et qu'une valeur est souvent correcte pour un attribut si la source qui la fournit est digne de confiance. Comme l'algorithme est non supervisé, nous modélisons la fiabilité d'une source comme une fonction de la probabilité d'exactitude des attributs qu'elle fournit.

3.1.1.2 Modèle de vérité latente

Cette méthode utilise les réseaux bayésiens pour estimer la fiabilité de l'information. « Latent Truth Model »LTM est proposé dans (Zhao et al., 2012). Il se base sur deux hypothèses :

— les données doivent contenir un seul attribut avec une valeur atomique et

— la gestion de plusieurs valeurs vraies pour le même attribut.

Pour chaque source, LTM considère les probabilités à priori pour qu'elle soit un vrai positif ou un faux positif avec des erreurs négatives dans chaque cas représentant respectivement la sensibilité notée $(\alpha 1, 1, \alpha 1, 0)$ et la spécificité notée $(\alpha 0, 1, \alpha 0, 0)$. Enfin, les valeurs dont le degré de fiabilité est supérieur à 0.5 sont considérées comme vraies. Il faut noter que le LTM peut ne pas détecter des valeurs vraies pour certains attributs.

3.1.2 Méthodes basées sur l'optimisation

Les méthodes basées sur l'optimisation reposent sur la définition d'une fonction d'optimisation qui peut capturer les relations entre les qualités des sources et la véracité des affirmations, avec une méthode itérative pour calculer conjointement ces deux ensembles de paramètres. La fonction d'optimisation est généralement formulée comme suit :

$$argmin_{w_s, v^*_{o_a}} \sum_{o \in O} \sum_{s \in S} w_s d(v^s_{o_a}, v^*_{o_a}) \tag{3.1.1}$$

3.1.2.1 Recherche de la vérité semi-supervisée

Publié en 2011 par (Yin and Tan, 2011b) « Semi Supersived Truth Finding » SSTF est une approche semi-supervisée qui trouve les vraies valeurs à l'aide de données de vérité terrain. Ces données de vérité terrain, même en très petite quantité, peuvent grandement aider à identifier des sources de données fiables. Les expériences montrent que cette méthode atteint une plus grande précision que les approches existantes, et qu'elle peut être appliquée à des ensembles de données très volumineux lorsqu'elle est mise en œuvre avec MapReduce.

3.1.3 Les méthodes par partitionnement de données

Ici nous avons des algorithmes qui utilisent le partitionnement de données pour la recherche de vérité. Nous avons comme algorithmes :

1. Recherche de la vérité avec partitionnement des attributs
2. Découverte de la vérité basée sur le partitionnement efficace des données

3.1.3.1 Recherche de la vérité avec partitionnement des attributs

Proposée dans (Lamine Ba et al., 2015), c'est une méthode de recherche de vérité qui se base sur le partitionnement des attributs des objets. Son objectif est donc d'estimer une partition optimale de l'ensemble des attributs de telle sorte qu'en appliquant indépendamment n'importe quel algorithme de recherche de vérité de référence sur chaque sous ensemble de la partition, on maximisera la précision de cet algorithme. L'estimation de la partition optimale dans ce cas se base seulement sur la qualité de l'algorithme de recherche. Ils définissent la qualité (τ) d'un algorithme de recherche de vérité de différentes façons :

— le maximum des précisions des sources à la fin de l'exécution de l'algorithme :

$$\tau = \max_{s \in S} Accuracy(S) \tag{3.1.2}$$

— la moyenne des précisions des sources à la fin de l'exécution de l'algorithme :

$$\tau = \frac{1}{|S|} \sum_{s \in S} Accuracy(S) \tag{3.1.3}$$

— le rapport entre le nombre de valeurs d'attribut d'objet identifié (vrai positive) à la fin de l'exécution de l'algorithme et le produit entre le nombre d'objet et le nombre d'attribut :

$$\tau = \frac{\text{Nombre de vrai positive}}{|O| \times |A|} \tag{3.1.4}$$

Le « *ground truth* » c'est à dire les vraies valeurs des attributs d'objets doivent être disponibles (données par les experts en la matière), ce qui n'est pas réaliste vu l'objectif des algorithmes de recherche de vérité (cela suppose que la vérité n'existe pas encore). Ce

taux est donc utilisé pour la comparaison entre les partitions obtenues avec les fonctions objectives **max** et **avg** pour savoir laquelle donne un résultat proche de l'optimalité.

Une fonction poids notée θ est associée à chaque partition pour le choix de la partition optimale et est définie par :

$$\begin{array}{rcccl} \theta & : & P & \rightarrow & [0,1] \\ & & p & \mapsto & \theta(p) = \frac{1}{|p|} \times \sum_{X \in p} \tau(X) \end{array} \tag{3.1.5}$$

où $|p|$ représente le nombre d'ensembles dans la partition p et $\tau(X)$ la qualité (τ) de l'algorithme de recherche de vérité exécuté sur les données contenant les attributs dans l'ensemble X. La partition optimale est celle dont le poids est le plus élevé. Pour ce faire, il détermine donc toutes les partitions de l'ensemble des attributs P, calcule le poids de chaque partition (avec l'équation 3.1.5) et choisit celle qui a le poids le plus élevé. Le problème majeur est la détermination de toutes les partitions de l'ensemble des attributs lorsque le nombre d'attributs est élevé. Pour apporter une solution à ce problème, les auteurs (Lamine Ba et al., 2015) ont également proposé une approche d'échantillonnage aléatoire pour restreindre efficacement la recherche d'une partition optimale à un nombre limité de candidats parmi l'ensemble des partitions. Cependant, le choix de façon aléatoire ne garantit pas de retourner la partition optimale au même titre que l'approche brute force avec les fonctions **max** et **avg**.

3.1.3.2 Découverte de la vérité basée sur le partitionnement efficace des données

Proposé par (Osias Noël Nicodème Finagnon and Ba, 2021), TD-AC est basée sur le clustering de données en apprentissage automatique avec la *méthode des k-moyennes* combinée à *l'indice de silhouette* pour déterminer la valeur optimale de k, afin de détecter la partition optimale de l'ensemble des attributs. Une telle partition optimale maximise la précision du processus de recherche de la vérité sans avoir à explorer toutes les partitions possibles.

3.2 Recherche de vérité et algorithmes paralléles

Dans le cadre de leur processus d'extensibilité visant à traiter efficacement les données volumineuses, Ouyang et al. (2016) ont proposé **l'algorithme parallèle de découverte de la vérité** qui utilise le framework MapReduce. En effet cet algorithme décompose le problème de découverte de la vérité à grande échelle en plusieurs tâches de Map et de Reduce à petite échelle qui peuvent être exécutées en parallèle sur un cluster de processeurs. Cette approche a prouvé l'efficacité de la découverte de la vérité sur de grands ensembles de données. Cet algorithme est utilisé dans les applications de crowdsourcing. Le crowdsourcing est un processus qui consiste à obtenir le contenu, les informations ou les services nécessaires en sollicitant les contributions d'un grand groupe de personnes généralement indéterminées, plutôt que celles d'employés ou de fournisseurs traditionnels. Il devient de plus en plus populaire car il offre un moyen facile, rapide et économique de collecter un grand volume de données pour une variété d'applications, telles que l'étiquetage d'images, la description d'images, l'analyse de sentiments, la vérification de listes, le comptage d'objets, la traduction et la conception de logos. Cet algorithme parallèle utilise seulement des données quantitatives.

4. Contribution

Dans ce chapitre, nous parlerons de la parallélisation des différents algorithmes de recherche de vérité étudiés. D'abord nous introduisons la programmation parallèle et un exemple de calcul parallèle. Ensuite nous faisons une étude approfondie sur les algorithmes de recherche de vérité en les classifiant par algorithmes parallelisables et algorithmes non parallelisable. Enfin nous proposons une version parallèle de l'algorithme **TruthFinder.**

4.1 La programmation parallèle

4.1.1 Définition

Le calcul parallèle consiste en l'exécution simultanée d'une même tâche, partitionnée et adaptée afin de pouvoir être répartie entre plusieurs processeurs en vue de traiter plus rapidement des problèmes plus grands. Le calcul parallèle est très populaire de nos jours surtout avec l'existence de paradigmes de programmation parallèle et de plate-formes de calcule distribuées robustes tels que MapReduce et Spark. Nous signalons que tout les programmes ne sont pas parallélisables.

Prenons l'exemple de la somme de deux nombres binaires. Nous considérons l'additionneur Carry ripple et l'additionneur Carry look ahead.

4.1.2 Présentation de l'algorithme Carry Ripple

Carry ripple est un algorithme qui fait la somme arithmétique de deux nombres binaires. Pour compiler cet algorithme nous utilisons un additionneur que nous appellons Additionneur Carry Ripple. La figure 4.1 montre que cet additionneur comprend des additionneurs complets en cascade dans sa structure, de sorte que le portage sera généré à chaque étape complète de l'additionneur dans un circuit d'additionneur ripple-carry. Ces sorties de carry à chaque étape d'additionneur complet

sont transmises à son prochain additionneur complet et y sont appliquées en tant qu'entrée de portage. Ce processus se poursuit jusqu'à sa dernière étape complète de l'additionneur. Ainsi, chaque bit de sortie de portage est ondulé à l'étape suivante d'un additionneur complet.

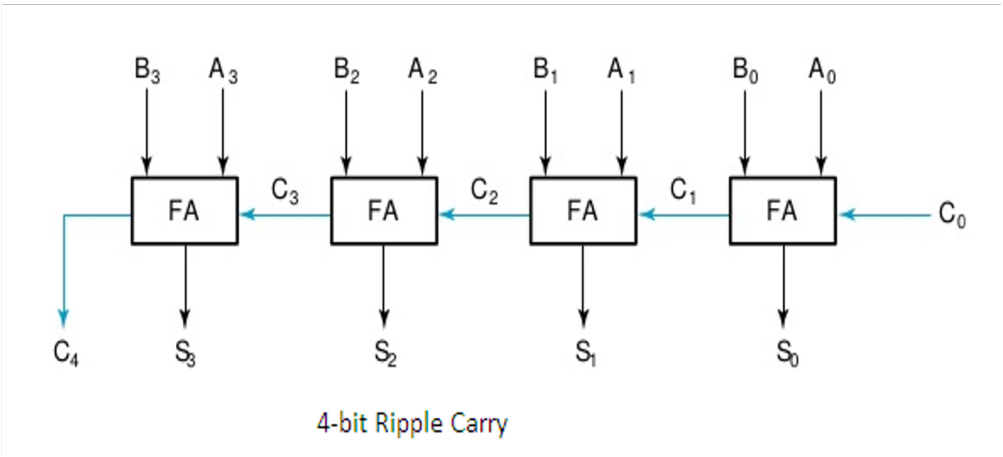

FIGURE 4.1 – Schéma de l'additionneur Carry Ripple

Prenons un exemple de deux séquences d'entrée A=0101 et B=1010. Ceux-ci représentent A_3 A_2 A_1 A_0 et B_3 B_2 B_1 B_0. Selon ce concept d'additionneur, le portage d'entrée est 0 lorsque A_0 et B_0 sont appliqués au 1er additionneur complet. La Somme et le carry seront générés selon les équations suivantes : S_i= $A_i + B_i + C_i$ et $C_i = A_iB_i + B_iC_{i-1} + C_{i-1}A_i$.

Ici $A_0 = 1; B_0 = 0; C_0 = 0$. Selon sa théorie, les équations de sortie pour la somme et le carry sont : S_0= $A_0 + B_0 + C_0$ et $C_0 = A_0B_0 + B_0C_0 + C_0A_0$.

Selon cette équation, pour le 1er additionneur complet $S_0 = 1$ et la sortie Carry, c'est-à-dire $C_0 = 0$. Même chose que pour les bits d'entrée suivants A_1 et B_1, sortie $S_1 = 1$ et $C_1 = 0$. Ici, le point important est que le deuxième additionneur complet de l'étape obtient le portage d'entrée, c'est-à-dire C_0 qui est le portage de sortie de l'additionneur complet de l'étape initiale. Comme cela on obtiendra la séquence de sortie finale $(S_3, S_2, S_1, S_0) = (1, 1, 1, 1)$ et le portage de sortie $C_3 = 0$. Il s'agit du processus d'ajout pour les séquences d'entrée 4 bits lorsqu'il est appliqué à cet additionneur de portage.

Dans les additionneurs de carry ripple, pour chaque bloc d'additionneur, les deux bits qui doivent

être ajoutés sont disponibles instantanément. Cependant, chaque bloc d'additionneur attend que le carry arrive de son bloc précédent. Ainsi, il n'est pas possible de générer la somme et le portage d'un bloc tant que le portage d'entrée n'est pas connu. La solution trouvée a ce problème est l'additionneur CARRY LOOK AHEAD (CLA).

4.1.3 Présentation de l'algorithme Carry Look Ahead

Carry Look-ahead Adder est le circuit d'additionneur plus rapide. Il réduit le délai de propagation, qui se produit pendant l'addition, en utilisant des circuits matériels plus complexes. Il est conçu en transformant le circuit d'additionneur Carry Ripple de telle sorte que la logique de portage de l'additionneur soit modifiée en logique à deux niveaux. La figure 4.2 montre que dans cet additionneur , l'entrée de portage à n'importe quel étage de l'additionneur est indépendante des bits de portage générés aux étapes indépendantes. Ici, la sortie de n'importe quelle étape dépend uniquement des bits qui sont ajoutés dans les étapes précédentes et de l'entrée de portage fournie à l'étape de début. Par conséquent, le circuit à n'importe quel stade n'a pas à attendre la génération du carry-bit de l'étape précédente et le bit de transport peut être évalué à tout moment.

Nous définissons deux variables <carry generate> G_i et <carry propagate> P_i. En utilisant les termes G_i et P_i, la somme S_i et le carry C_{i+1} sont données par :

$$P_i = A_i + B_i.$$

$$G_i = A_i B_i.$$

$$S_i = P_i + C_i$$

$$C_{i+1} = C_i.P_i + G_i.$$

Par conséquent, les bits de portage C_1, C_2, C_3 et C_4 peuvent être calculés comme suit :

$$C_1 = C_0P_0 + G_0.$$

$$C_2 = C_1P_1 + G_1 = (C_0P_0 + G_0)P_1 + G_1.$$

$$C_3 = C_2P_2 + G_2 = (C_1P_1 + G_1)P_2 + G_2.$$

$$C_4 = C_3P_3 + G_3 = C_0P_0P_1P + 2P_3 + P_3P_2P_1G_0 + P_3P_2G_1 + G_2P_3 + G3.$$

A partir des équations ci-dessus nous pouvons dire que :

— P_i et G_i sont utilisés pour faciliter le calcul du carry C_i a l'étage i, Pour ne pas que C_i dépend de la generation de C_{i-1}.

— C_i depend que de G_i , P_i et de C_{in} (le carry a l'entrée qui est toujours 0) pour être génère.

— C_{i-1} n'a pas besoin d'attendre C_i pour etre propage mais est propage en meme temps que les C_i precedents.

Prenons le cas de deux nombres $A = 0101$ et $B = 0110$

$$A_1 = 1; A_2 = 0; A_3 = 1; A_4 = 0$$

$$B_1 = 0; B_2 = 1; B_3 = 1; B_4 = 0$$

$$C_0 = 0$$

$$G_1 = A_1B_1 = 1 * 0 = 0$$

$$P_1 = A_1 + B_1 = 1 + 0 = 1$$

$$G_2 = A_2B_2 = 0 * 1 = 0$$

$$P_2 = A_2 + B_2 = 0 + 1 = 1$$

$$G_3 = A_3 B_3 = 1 * 1 = 1$$

$$P_3 = A_3 + B_3 = 1 + 1 = 10 = 0$$

$$G_4 = A_4 * B_4 = 0 * 0 = 0$$

$$P_4 = A_4 + B_4 = 0 + 0 = 0$$

$$C_1 = 0$$

$$C_2 = P_1.C_1 + G_1 = 1 * 0 + 0 = 0$$

$$C_3 = C_2.P_2 + G_2 = (C_1.P_1 + G_1).P_2 + G_2 = P_2 P_1 C_1 + P_2 G_1 + G_2 = 0$$

$$C_4 = C_3.P_3 + G_3$$

$$C_4 = (C_2.P_2 + G_2).P_3 + G_3 = P_2 P_1 C_1 P_3 + P_2 G_1 P_3 + G_2 P_3 + G_3 = 1$$

$$C_5 = C_4.P_4 + G_4$$

$$C_5 = C_1.P_1.P_2.P_3.P_4 + P_4.P_3.P_2.G_1 + P_4.P_3.G_2 + G_3.P_4 + G_4 = 0$$

$$Si = Pi + Ci$$

$$S_1 = P_1 + C_1 = 1 + 0 = 1$$

$$S_2 = P_2 + C_2 = 1 + 0 = 1$$

$$S_3 = P_3 + C_3 = 0 + 0 = 0$$

$$S_4 = P_4 + C_4 = 0 + 1 = 1$$

Nous pouvons conclure en disant l'additionneur Carry Ripple est la version séquentielle et l'additionneur Carry Look Ahead est la version séquentielle. Avec cet additionneur le calcule est rapide et fiable.

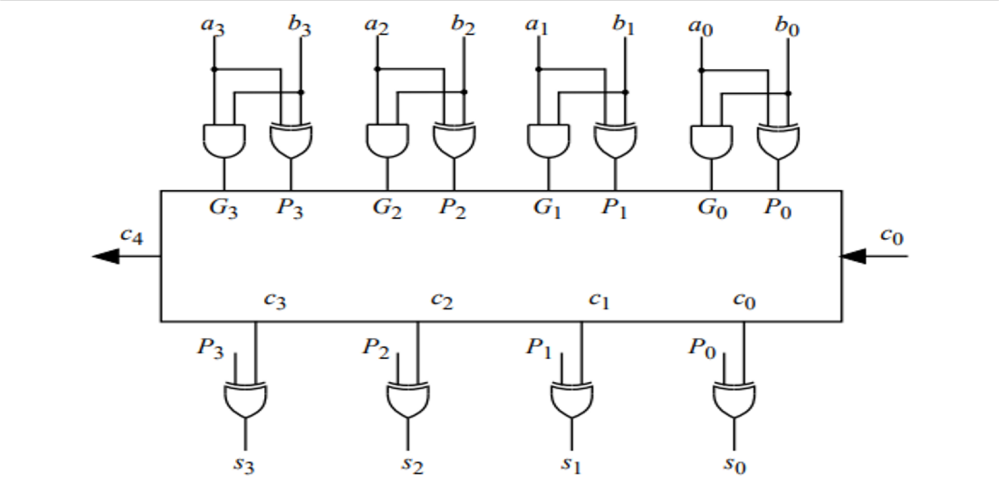

FIGURE 4.2 – Schéma de l'additionneur Carry Look Ahead

4.2 Étude de la parallélisation des algorithmes séquentiels

Dans cette section nous effectuons une étude approfondie des algorithmes de recherche de vérité concernant leur parallélisation en donnant les critères sur lesquels nous nous basons pour cette classification.

4.2.1 Algorithmes parallélisables vs. algorithmes non parallélisables

La conception d'un algorithme parallèle pour un problème donné est plus complexe que la conception d'un algorithme séquentiel du même problème. Cela semble difficile car elle prend en compte plusieurs facteurs tels que :

1. la partie du programme qui peut être traité en parallèle
2. la manière de distribuer les données
3. les dépendances des données
4. la répartition de charges entre les processeurs

5. les synchronisations entre les processeurs

Nous considérerons un algorithme parallélisable si :

1. ses différentes fonctions sont parallélisables
2. les fonctions sont des fonctions sommes
3. les fonctions sont des produits
4. la distribution des données se fait par partitionnement

Le tablerau 4.1 présente les algorithmes parallélisables et le tableau 4.2 présente les algorithmes non parallélisables.

4.2.1.1 Algorithmes de recherche de vérité Parallélisables

Algorithmes	Fonctions	Justification
MajorityVoting	$$vote(v) = \sum_{s \in S_v} \frac{w_s}{|V_s|} \quad (4.2.1)$$	Cette fonction est une fonction somme donc parallélisable.
TruthFinder (Yin et al., 2008)	$$T_s = \sum_{v \in V_s} \frac{C_v}{|V_s|} \quad (4.2.2)$$ $$\sigma_v = -\sum_{s \in S_v} \ln(1 - T_s) \quad (4.2.3)$$ $$\sigma_v^* = \sigma_v + \rho \sum_{v^* \in V_d} \sigma_{v^*}.sim(v, v^*) \quad (4.2.4)$$ $$C_v = \frac{1}{1 + e^{-\gamma \sigma_v^*}} \quad (4.2.5)$$	L'algorithme est composé de trois fonctions sommes qui peuvent être parallélisée chacune.
Recherche de la vérité avec partitionnement des attributs (Ba et al., 2015)	$$\tau = \frac{1}{|S|} \sum_{s \in S} Accuracy(S) \quad (4.2.6)$$	Cet algorithme utilise le partitionnement de données. Le partitionnement est une méthode qui facilite la programmation car nous pouvons faire une exécution sur chaque partition.
TD-AC (Osias Noël Nicodème Finagnon and Ba, 2021)	Il se comporte comme le TFAP en considérant les fonction des autres algorithmes.	Etant basé sur le partitionnement de données, TD-AC est une version améliorée de TFAP. Cet algorithme utilise une partition efficace des données donc parallélisable.
SSTF (Yin and Tan, 2011a)		Cet algorithme est parallélisable car en présence de grand volume de données, il s'utilise avec MapReduce .
LCA (Pasternack and Roth, 2013a)	$$C_v = \frac{C_v^*}{C_{dsum}} \quad (4.2.7)$$ $$T_s = \sum_{v \in V_s} \frac{C_v.W_{s,d}}{\sum_{d \in D} W_{s,d}} \quad (4.2.8)$$	Cet est algorithme parallélisable car il est constitué de fonctions parallélisables.
Depen (Dong et al., 2009)	$$C_v = C_v + t_{socre}.voteCount \quad (4.2.9)$$ $$T_s = \frac{1}{|V_s|} \sum_{v \in V_s} \frac{e^{C_v}}{\sum_{v' \in V_{Dv}} e^{C_{v'}}} \quad (4.2.10)$$	Cet algorithme est parallélisable car il est constitué de fonctions parallélisables.

TABLE 4.1 – Tableau des algorithmes de recherche de vérité parallelisables

4.2.1.2 Algorithmes de recherche de vérité non parallélisables

Algorithmes	Fonctions	Justification
Cosine (Galland et al., 2010)	$C_v = \frac{\sum_{s \in S_v} (T_s^i)^3 - \sum_{s \in S_d s \notin S_v} (T_s^i)^3}{\sum_{s \in S_d} (T_s^i)^3}$ (4.2.11) $T_s^i = (1-n)T_s^{i-1} + n\frac{\sum_{v \in V_s} C_v - \sum_{v \in V_{Ds} - V_s} C_v}{(\lvert V_{Ds} \rvert \sum_{v \in V_{Ds}} C_v^2)^1/2}$ (4.2.12)	Cet algorithme n'est pas parallélisable car il est composé de fonction n'nont parallélisables.
2-Estimates (Galland et al., 2010)	$C_v = \frac{\sum_{s \in S_v} (1 - T_s) + \sum_{s \in S_{\bar{v}}} (T_s)}{\lvert S_d \rvert}$ (4.2.13) $T_s = \frac{\sum_{v \in V_s} (1 - C_v) + \sum_{s \in S_{\bar{v}}} C_v}{\lvert V_{Ds} \rvert}$ (4.2.14)	Cet algorithme n'est pas parallelisable car ses fonctions sont composées de sous fonctions. Pendant la parallélisation les sous fonctions dépendront de chacune ce qui ralentira le processus.
3-Estimates (Galland et al., 2010)	$C_v = \frac{\sum_{s \in S_v} (1 - T_s \epsilon_v) + \sum_{s \in S_{\bar{v}}} T_s \epsilon_v}{\lvert S_d \rvert}$ (4.2.15) $T_s = \frac{\sum_{v \in V_s \wedge \epsilon_v \neq 0} (1 - C_v)/\epsilon_v + \sum_{d \in D_s} (\sum_{s \in S_{\bar{v}} \wedge \epsilon_v \neq 0} C_v/\epsilon_v)}{\lvert v \in V_{Ds}/\epsilon_v \neq 0 \rvert}$ (4.2.16)	Cet algorithme n'est pas parallelisable car ses fonctions sont composées de sous fonctions. Pendant la parallélisation les sous fonctions dépendront de chacune ce qui ralentira le processus.
(MLE) (Wang et al., 2012)	$C_v = \frac{a_v \beta_1}{a_v \beta_1 + b_v (1 - \beta_1)}$ (4.2.17) $a(s) = \frac{\sum_{v \in V_s} C_v}{C_{sum} + C_v}$ (4.2.18) $b(s) = \lvert V_s \rvert - \frac{\sum_{v \in V_s} C_v}{\lvert V \rvert - C_{sum}}$ (4.2.19)	Cet algorithme n'est pas parallelisable car ses fonctions sont constituées de sous fonctions.

TABLE 4.2 – Tableau des algorithmes de recherche de vérité non parallelisable

La parallélisation des algorithmes de recherche de vérité se font en utilisant les mêmes technologies. Notre étude portera sur la parallélisation de l'algorithme de recherche de vérité **TruthFinder**.

4.3 Proposition de la version parallèle de TruthFinder

Dans cette section nous proposons la version parallèle de l'algorithme **TruthFinder** en utilisant le paradigme MapReduce (Map et Reduce).

4.3.1 MapReduce

Initialement utilisé par Google, MapReduce est un modèle de programmation qui a gagné en popularité à cause de sa capacité à diviser et à traiter de grands volumes de données en parallèle, obtenant des résultats plus rapides. Il le fait de manière fiable et tolérante aux pannes.La particularité de MapReduce est qu'il comprend deux fonctions Map et Reduce.Ils sont excécutés les uns après les autres.

— La fonction Map prend en entrée les données sous forme de <clé,valeur> et les traite en produisant un autre ensemble de données sous forme de paires <clé,valeur> intermédiaire en sortie.

— La fonction Reduce prend en entrée la sortie de la fonction Map et produit des paires <clé,valeur> en sortie.

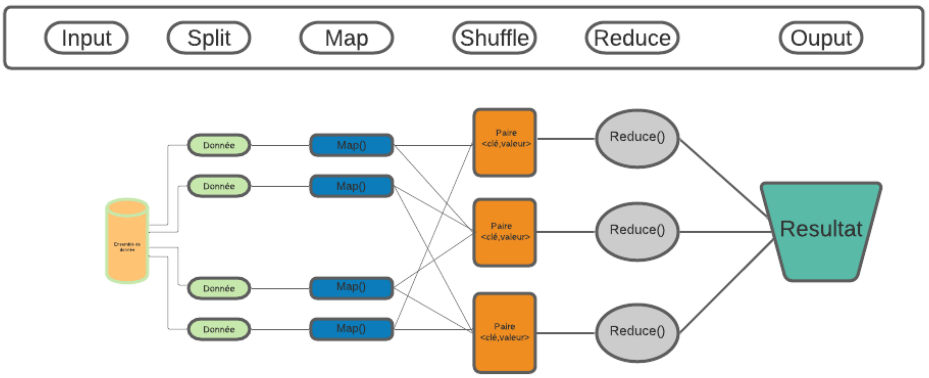

FIGURE 4.3 – Schéma de fonctionnement de MapReduce

4.3.2 Détails de l'algorithme TruthFinder

Publié en 2008 par (Yin et al., 2008), **TruthFinder** est un algorithme basé sur l'analyse bayésienne qui s'appuie sur l'honnêteté de la source d'information et suit l'heuristique selon laquelle si les valeurs données sont essentiellement vraies pour de nombreux cas, alors elles seront aussi probablement vraies pour d'autres cas.Il calcule la fiabilité de l'information donnée par une source et la probabilité qu'une information donnée par une source s soit fiable est T_s et celle pour qu'elle soit non fiable est $(1 - T_s)$. Si l'information est fournie par plusieurs sources, alors sa probabilité de ne pas être fiable est $\prod_{s \in S_v}(1 - T_s)$.la fiabilité de la source dans **TruthFinder** est (équation 4.3.1) :

$$T_s = \sum_{v \in V_s} \frac{C_v}{|V_s|} \tag{4.3.1}$$

Nous avons le score de confiance d'une valeur qui est donné par (équation 4.3.2) :

$$\sigma_v = -\sum_{s \in S_v} \ln(1 - T_s) \tag{4.3.2}$$

Pour éviter l'affluence par valeur inférieure lorsqu'il y a une quantité très faible de vérité le logarithme est utilisé . Ce score est ajusté de sorte qu'il élimine l'effet que les valeurs similaires d'un même attribut peuvent avoir les unes sur les autres par (équation 4.3.3) :

$$\sigma_v^* = \sigma_v + \rho \sum_{v^* \in V_d} \sigma_{v^*}.sim(v, v^*) \tag{4.3.3}$$

ρ est un paramètre de contrôle du poids de cet effet entre les valeurs et $sim(v, v^*)$ est la fonction de similarité entre les valeurs v et v^*. La confidence finale de la valeur est calculée avec une fonction logistique pour être positive comme suit (équation 4.3.4) :

$$C_v = \frac{1}{1 + e^{-\gamma \sigma_v^*}} \tag{4.3.4}$$

Le paramètre γ est un facteur d'amortissement qui compense les effets lorsque des sources ayant des valeurs similaires sont indépendantes. La plus grande valeur de C_v est considérée comme vraie et les autres sont considérées comme fausses. Il faut noter qu'un seuil de convergence est donné à l'algorithme. Puisque **TruthFinder** calcule la similarité entre les valeurs, il peut être considérablement affecté par le nombre élevé de valeurs distinctes à comparer, ce qui explique des performances relativement plus faibles lorsque le nombre de conflits est élevé.

L'algorithme **TruthFinder** est composé de fonctions sommes. Pour sa parallélisation, nous parallélisons les différentes fonctions qui sont :

1. Fonction de calcul de la confiance d'une valeur
2. Fonction de calcul de la confidence d'une valeur utilisant une fonction logistique
3. Fonction de fiabilité d'une source

4.3.2.1 Parallélisation de la fonction de calcul de la confidence d'une valeur

Pour la parallélisation de cette fonction de l'algorithme, nous utiliserons l'ensemble des données en entrée qui subiront des transformations avec le framework MapReduce.Nous aurons des étapes Map et des étapes Reduce qui sont présentées dans la figure 4.4 et l'algorithme 1.

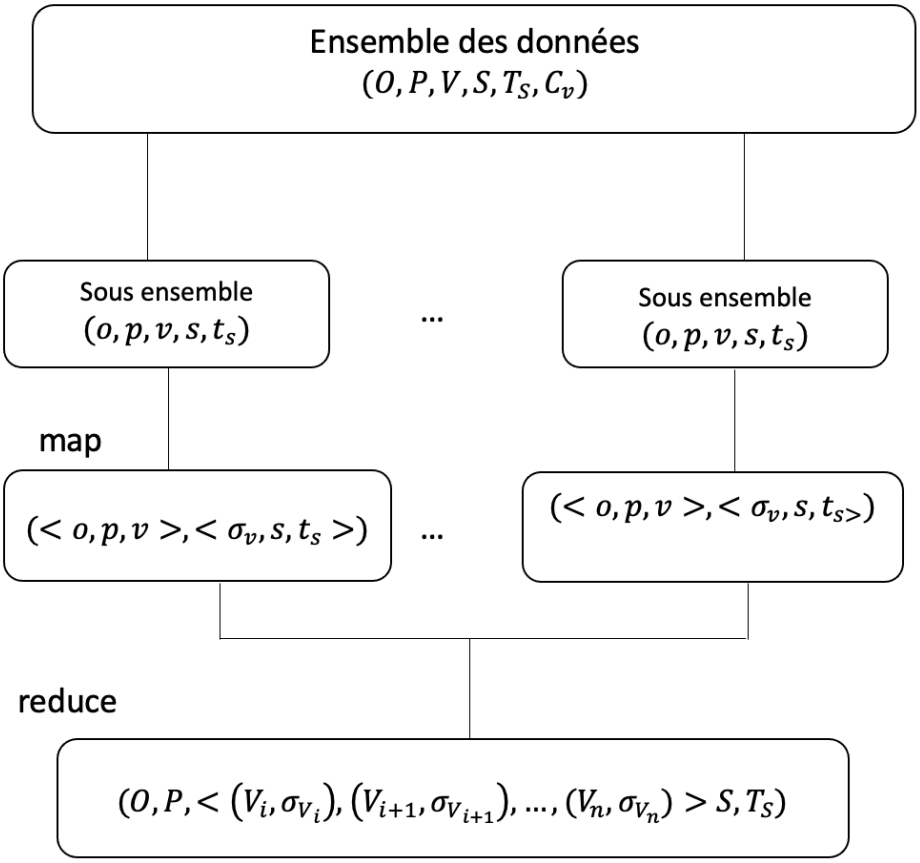

FIGURE 4.4 – Schéma de parallélisation de la fonction de calcul de la confidence d'une valeur

Algorithme 1 Fonction de calcul de la confidence d'une valeur :(S,O,A,V,C$_v$,T_s)

Pré-conditions : $\forall s \in S : T_s \leftarrow 0.8$
Pré-conditions : $\forall v \in V : C_v \leftarrow 0$
1: Écriture de la fonction MAP
2: **Pour tout** $(o, p, v, s, t_s) \in (O, P, V, S, T_S)$ (en parallele) **faire**
3: $\sigma'_v \leftarrow -ln(1 - t_s)$
4: **Fin Pour**
5: Retourne un couple $(< o, p, v >, < \sigma'_v, s, t_s >)$
6: Écriture de la fonction REDUCE
7: **Pour tout** $(< o, p, v >, < \sigma_v, s, t_s >) \in (< O, P, V >, < \sigma_V, S, T_S >)$ (en parallèle) **faire**
8: $\sigma_v \leftarrow \sum_{s \in S_v} \sigma'_v$
9: **Fin Pour**
10: Retourne $(O, P, < (V_i, \sigma_{V_i}), ..., (V_n, \sigma_{V_n}) >, S, T_S)$

4.3.2.2 Fonction de calcul de la confidence d'une valeur utilisant une fonction logistique

Pour la parallélisation de celle-ci nous prenons en entrée la sortie de la fonction de la confidence d'une valeur. L'entrée suit des transformations et donne à la sortie la fonction de calcul de la confidence d'une valeur utilisant une fonction logistique. Nous calculons la similarité entre les valeurs et la fonction d'ajustement en faisant les transformation. Ces étapes sont présentées dans la figure 4.5 et l'algorithme 2.

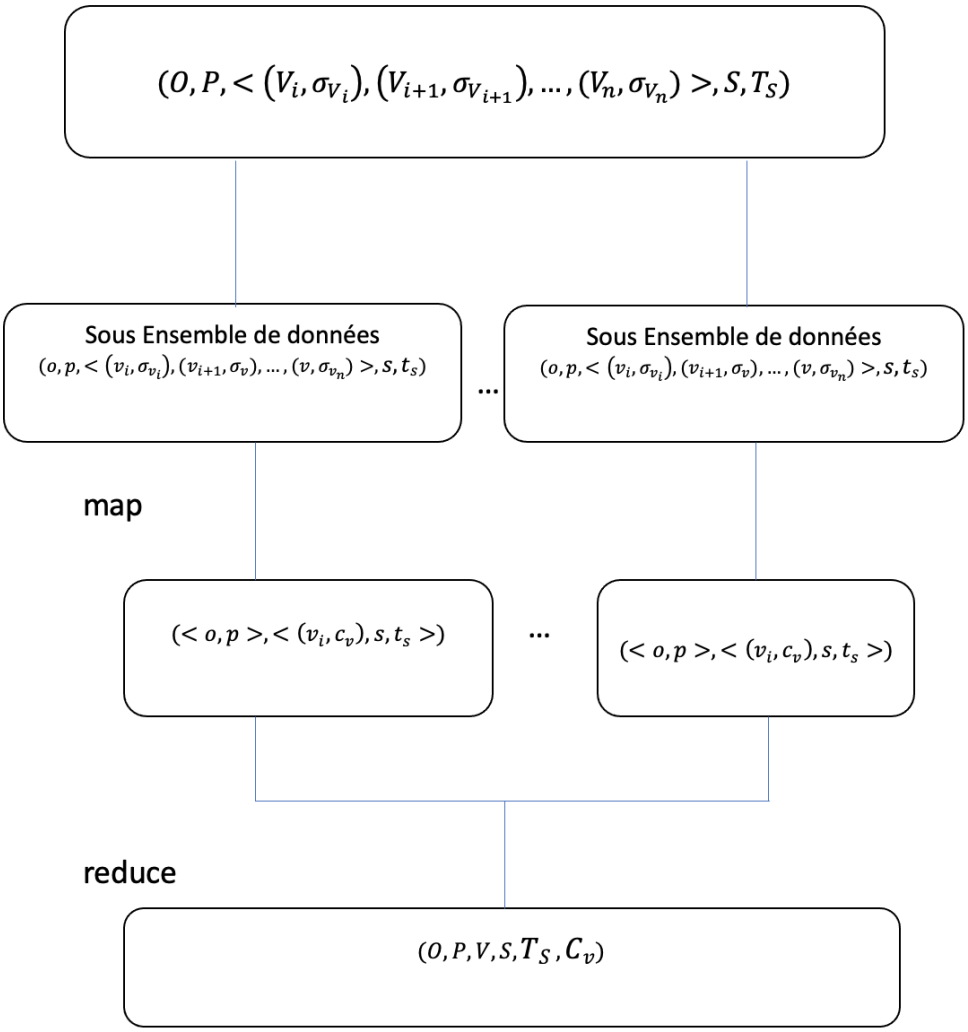

FIGURE 4.5 – Schéma de parallélisation de la fonction de calcul de la confidence d'une valeur utilisant une fonction logistique

Algorithme 2 Fonction de calcul de la confidence d'une utilisant une fonction logistique valeur :(S,O,P,V,C$_v$, T_s,σ_v, ρ, γ)

Pré-conditions : $\gamma \leftarrow 0.5$,$\rho \leftarrow 0.7$
Pré-conditions : $\forall v \in V : c_v \leftarrow \sigma_v$
1: *Écriture de la fonction MAP*
2: **Pour tout** $(o, p, <(v_i, \sigma_{v_i}>, s, t_s) \in (O, P, <(V_i, \sigma_{v_i}>, S, T_S)$ (en parallèle) **faire**
3: $c_v \leftarrow \rho\sigma'_v \times sim(v, v')$
4: $C_v \leftarrow 1/(1 + \exp(-\gamma \times c_v))$
5: **Fin Pour**
6: Retourne un couple $(< o, p >, < v, c_v, s, t_s >)$
7: *Écriture de la fonction REDUCE*
8: **Pour tout** $(< o, p >, < v, c_v, s, t_s >) \in (< O, P >, < V, C_V, S, T_S >)$ (en parallèle) **faire**
9: $C_v \leftarrow \sum_{s \in S_v} c_v$
10: **Fin Pour**
11: retourne (O, P, V, T_S, C_V)

4.3.2.3 Fonction de fiabilité d'une source

Prenant en entrée la sortie de la Fonction de calcul de la confidence d'une valeur utilisant une fonction logistique,les données subissent des transformations pour donner en sortie la fonction de calcul de la fiabilité d'une source. Nous donnons les étapes dans la figure 4.6 et l'algorithme 3

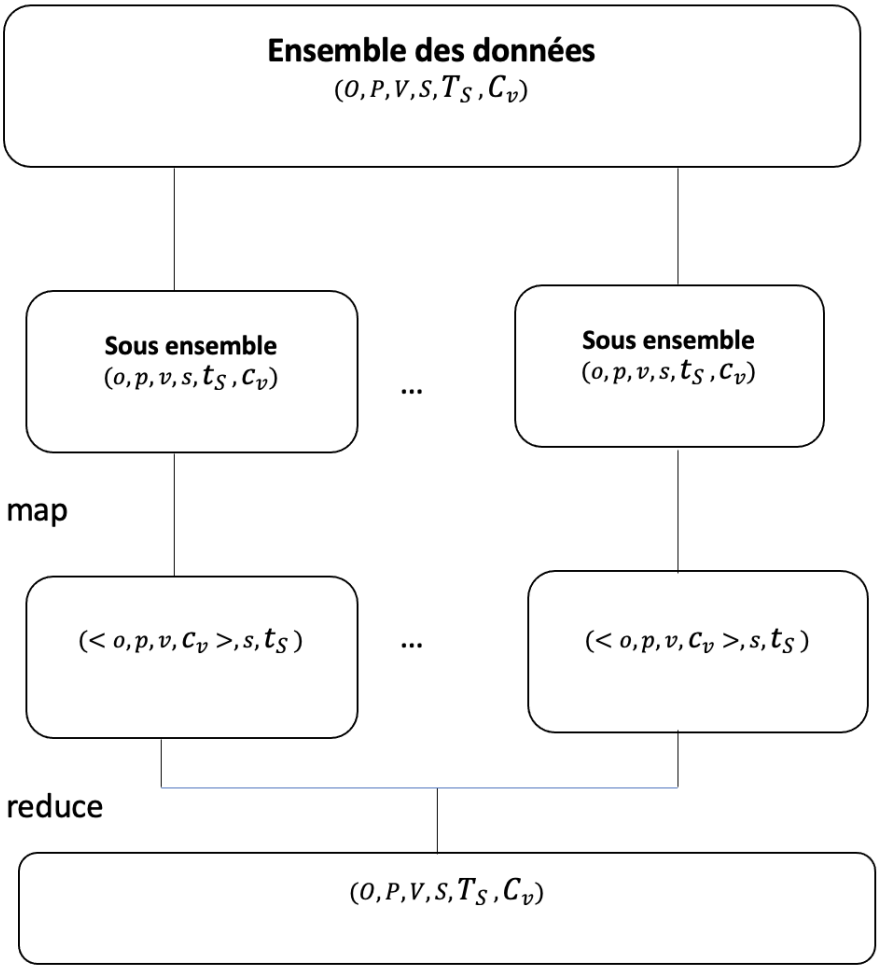

FIGURE 4.6 – Schéma de parallélisation de la fonction de calcul de la fiabilité d'une source

Algorithme 3 Fiabilité source :(S,O,P,V,C$_v$, T_s)

Écriture de la fonction MAP
Pour tout $(s, o, p, v, c_v, t_s) \in (S, O, A, V, C_v, T_s)$ (en parallèle) **faire**
  $t_s \leftarrow c_v/|V_s|$
Fin Pour
Retourne un couple $(< o, p, v, c_v >, < s, t_s >)$
Écriture de la fonction REDUCE
Pour tout $(< o, p, v, c_v >, < s, t_s >) \in (< O, P, V, C_v >, < S, T_S >)$ (en parallèle) **faire**
  $T_s \leftarrow \sum_{v \in V_s} t_s$
Fin Pour
Retourne (S, O, A, V, C_v, T_s)

Algorithme 4 TRUTHFINDER :(S,O,P,V,,T$_s, \delta$)

Pré-conditions : $\delta \leftarrow 0.00001$

1: **tant que** l'algorithme ne converge pas **faire**
2: appel à la fonction de confidence d'une valeur (algorithme 1)
3: appel à la fonction de confidence d'une valeur utilisant une fonction logistique (algorithme 2)
4: appel à la fonction de fiabilité d'une source (algorithme 3)
5: **Fin tant que**

Ici nous présentons l'algorithme TruthFinder avec ses différentes fonctions. L'algorithme ne converge pas c'est a dire que le facteur de convergence δ est inférieur a la similarité entre les vecteur de fiabilité des source après chaque itération.

5. Validation de notre approche

Dans ce chapitre, nous démontrons la performance de notre approche sur différents jeux de données, en montrant l'efficacité de la version parallèle. Nous montrons aussi que son temps d'exécution est largement satisfaisant par rapport à celui de la version séquentielle.

En premier lieu, nous donnons les détails de l'implémentation de l'algorithme de recherche de la vérité **TruthFinder**. Notons que cet algorithme fait partir de la famille des méthodes basées sur des modèles probabilistes bayésiens et est une reference pour les autres algorithmes. Ensuite, nous détaillons les expérimentations effectuées sur des jeux de données synthétiques et réelles sous diverses configurations. Nous faisons aussi une analyse comparative des métriques de performance, la précision, le recall, le F1-score, l'accuracy, le temps d'exécution et l'utilisation de la mémoire des deux versions.

5.1 Implémentation de algorithme

Cet algorithme a été implémenté en utilisant le langage de programmation *Python* (Site Web Python) et *MapReduce* pour le modèle de parallélisme via *Apache Spark* (SparkSiteWeb) avec la librairie *Pyspark*.

5.1.1 Apache Spark

Définition Apache Spark

Apache Spark est un framework de traitements Big Data open source construit pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation. Apache Spark a originellement été développé par AMPLab, de l'Université UC Berkeley, en 2009 et passé open source sous forme de projet Apache en 2010. Spark est écrit en Scala et exécuté sur la machine

virtuelle Java (JVM). Spark propose un framework complet et unifié pour répondre aux besoins de traitements Big Data pour divers jeux de données, divers par leur nature (texte, graphe, etc.) aussi bien que par le type de source (batch ou flux temps-réel). Ensuite, Spark permet à des applications sur clusters Hadoop d'être exécutées jusqu'à 100 fois plus vite en mémoire, 10 fois plus vite sur disque.

Les fonctionnalités de Spark

Spark apporte des améliorations à MapReduce grâce à des étapes de shuffle moins coûteuses. Avec le stockage en mémoire et un traitement proche du temps-réel, la performance peut être plusieurs fois plus rapide et présente plusieurs avantages que d'autres technologies big data.Le moteur d'exécution est conçu pour travailler aussi bien en mémoire que sur disque. Les Résilient Distributed Datasets (RDD) est la principale innovation de Spark.Nous pouvons le voir comme une table dans une base de données qui peut comporter tout type de données.Les RDD permettent de réarranger les calculs et d'optimiser le traitement.

L'écosystème de Spark

Spark supporte les opérations de Map, Reduce et les requêtes SQL. On peut aussi faire le streaming de données, le machine learning et le traitement orienté graphe. La figure 5.1 montre l'écosystème de Spark.

— **Spark Streaming** : cette librairie peut être utilisé pour traitement en temps-réel des données en flux. Il s'appuie sur un mode de traitement en "micro batch" et utilise pour les données temps-réel DStream, c'est-à-dire une série de RDD.

— **Spark SQL** : Celui ci permet d'exposer les jeux de données Spark via API JDBC et d'exécuter des requêtes de type SQL en utilisant les outils BI et de visualisation traditionnels. Elle permet d'extraire, de transformer et de charger des données sous différents formats (JSON, Parquet, base de données) et de les exposer pour des requêtes ad-hoc.

— **Spark MLlib** : MLlib est une librarie de machine learning qui contient tous les algorithmes et utilitaires d'apprentissage classiques, comme la classification, la régression, le clustering,

le filtrage collaboratif, la réduction de dimensions, en plus des primitives d'optimisation sous-jacentes.

— **Spark GraphX** : GraphX est la nouvelle API pour les traitements de graphes et de parallélisation de graphes. GraphX étend les RDD de Spark en introduisant le Resilient Distributed Dataset Graph, un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arrêtes.

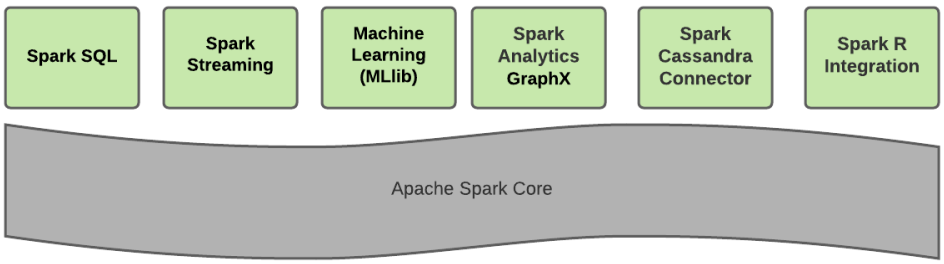

FIGURE 5.1 – Spark Framework Librairie

5.1.2 Python

Python est un langage de programmation open source qui offre des librairies de manipulation de données, de visualisation de données (Matplotlib, Numpy, pandas,Scikit-learn, etc). La programmation en Python implique moins de lignes de code que les autres langages de programmation disponible. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exception.

5.1.3 PySpark

Le langage de programmation de Apache Spark est Scala.Afin de prendre en charge la collaboration d'Apache Spark et Python,PySpark a été publié.C'est en fait une API Python pour Spark. De plus, PySpark facilite la connection aux jeux de données distribués résilients (RDD) dans Apache

Spark.

FIGURE 5.2 – Pyspark

Nous avons implémenté l'algorithme en utilisant les outils présentés ci-dessus.

5.2 Environnement de test, paramétrages et métriques de performance

5.2.1 Environnement de test et paramétrages

Nous utilisons un ordinateur portatif de 256 Go de disque dure, un processeur 3.1 GHz Dual-Core Intel Core i5, 16 Go de mémoire RAM pour les expérimentations. Les hyper-paramètres de l'algorithme et leur choix sont représentés dans le tableau 5.1 suivant pour les différents tests de données.

Algorithmes	Hyper-paramètre	Description	Critère et choix
TruthFinder	ρ	Paramètre de soutien de poids entre les valeurs	$\rho \in [0, 1]$
	γ	Facteur d'amortissement de compensation lorsque les sources ayant des valeurs similaires sont réellement dépendantes	$\gamma \in [0, 1]$
	δ	Paramètre de convergence de l'algorithme	Il doit être proche de 0 nous avons utilisé dans notre document 0.001 à 0.00001

TABLE 5.1 – Hyper-paramètres de l'algorithme et choix de leurs valeurs pour nos tests

5.2.2 Métriques de mesures de performance

Le problème de recherche de vérité se pose comme un problème de classification binaire parce qu'il consiste a dire si une valeur donnée par une source est vrai ou fausse. Nous affectons **1** si la valeur donnée est vraie et affectons **0** si la valeur est fausse. Nous utilisons les critères de performance comme le temps d'exécution ; la mémoire consommée ; la précision ; l'accuracy ; le recall et le f1-score pour l'évaluation des performance des différentes versions de notre algorithme.

La matrice de confusion est la matrice sur laquelle se basent les quatre derniers critères. C'est un tableau qui présente différentes prévisions et résultats de tests, en les comparant avec des valeurs réelles.

		Valeur vrai	
		Positive	Négative
Valeur prédite	Positive	vrais positifs(VP)	faux positifs (FP)
	Négative	faux négatifs (FN)	vrais négatifs(VN)

TABLE 5.2 – Matrice de confusion

— VP : lorsque la valeur prédite est pareil que la valeur réelle qui est 1.

— VN : lorsque la valeur prédite est pareil que la valeur réelle qui est 0.

— FP : lorsque la valeur prédite est 1 et que la valeur réelle est 0.

— FN : lorsque la valeur prédite est 0 et que la valeur réelle est 1.

Nous parlons de l'efficacité d'un algorithme lorsqu'il restreint les faux positifs et les faux négatifs tout en augmentant les vrais positifs et les vrais négatifs qui constitue les observations correctement prédites.

L'accuracy : est la proportion de prédictions correctes (à la fois les vrais positifs et les vrais négatifs) parmi le nombre total de cas examinés. La formule pour quantifier l'accuracy est la suivante :

$$accuracy = \frac{VP + VN}{VP + FP + FN + VN} \tag{5.2.1}$$

La précision : C'est le rapport entre les observations positives correctement prédites et le total des observations positives prédites.

$$precision = \frac{VP}{VP + FP} \tag{5.2.2}$$

Le recall : correspond au rapport entre les observations positives correctement prédites et toutes les observations de la classe réelle.

$$recall = \frac{VP}{VP + FN} \tag{5.2.3}$$

Le F1-score : représente la moyenne pondérée de la précision et du recall. Par conséquent, ce score tient compte à la fois des faux positifs et des faux négatifs. Intuitivement, il n'est pas aussi facile à comprendre que la précision, mais le F1-score est généralement plus utile que la précision, surtout si on a une distribution de classe inégale. La précision fonctionne mieux si les faux positifs et les faux négatifs ont un coût similaire. Si le coût des faux positifs et des faux négatifs est très différent, il est préférable d'examiner à la fois la précision et le rappel.

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \tag{5.2.4}$$

Nous évaluons aussi le temps d'exécution de chaque algorithme, c'est à dire le temps qu'il prend pour retourner le résultat de sa prédiction ainsi que la mémoire utilisée .

5.3 Expérimentation sur les données synthétiques

Dans cette section, nous détaillons les résultats de nos expérimentations sur les jeux de données synthétiques. Nous générons divers jeux de données de différentes tailles avec une même configuration dont les détailles sont présentés dans les tableaux 5.3 et 5.4. Nous débutons cette section en présentant notre générateur de données synthétiques et les divers jeux de données générés.

5.3.1 Description des données

Le recours aux données synthétiques (DS) donne la garantie d'avoir le « ground truth » associé pour une évaluation des performances de algorithme ; ce qui n'est pas forcément toujours le cas avec les données réelles. Nous utilisons un générateur de données synthétiques qui permet de simuler une grande variété de scénarios dans lesquels les sources présentent les mêmes comportements en termes de couverture, de taux d'erreur, de niveau de fiabilité, d'informations contradictoires, etc. Pour ce faire, le générateur suit les étapes suivantes :

1. Initialisation de l'ensemble des attributs (A), l'ensemble des objets (O) et l'ensemble des sources S ($na = |A|$, $no = |O|$, $ns = |S|$).
2. la taille des données est donnée par $td = no * ns * na$
3. Choix aléatoire d'une partition P de A.
4. Attribuer une source de façon aléatoire à chaque sous-ensemble de P.
5. Pour chaque sous-ensemble X_1 de P, sa source correspondante S aura une précision élevée sur ce sous-ensemble et une précision faible sur les autres X_2 dans P ($X_2 \neq X_1$) en utilisant deux lois de distribution uniforme de moyenne respective m_1 et m_2.
6. Prendre une partie du reste des sources S'' (la moitié par exemple) pour lesquelles on fixe une précision un peu élevée sur les attributs dans X_1, en fonction de la manière dont la précision de S sur X_1 est répartie sur l'ensemble des objets.

7. Éviter que chacune des sources choisie dans $S_{''}$ n'ait ni une précision locale significative, ni globale significative. Deux différents attributs dans un sous-ensemble peuvent contribuer à la précision locale de ce dernier.

8. Le jeu de données de la vérité de base ou « ground truth » est aussi généré à la fin.

Nous avons implémenté le générateur sous Python.

	Object	Property	Value	Source
0	Object1	Property4	373418	Source1
1	Object1	Property2	224546	Source1
2	Object2	Property4	484579	Source1
3	Object2	Property2	390812	Source1
4	Object3	Property4	314886	Source1
...	...	...	...	...
999995	Object1996	Property8	748696	Source48
999996	Object1997	Property8	773471	Source48
999997	Object1998	Property8	594826	Source48
999998	Object1999	Property8	702357	Source48
999999	Object2000	Property8	800929	Source48

(a) Jeu de données synthétique

	Object	Property	Value
0	Object1	Property1	187624
1	Object1	Property2	224546
2	Object1	Property3	396265
3	Object1	Property4	373418
4	Object1	Property5	210163
...	...	...	...
19995	Object2000	Property6	435825
19996	Object2000	Property7	195253
19997	Object2000	Property8	288447
19998	Object2000	Property9	369764
19999	Object2000	Property10	437146

(b) Ground truth associé

FIGURE 5.3 – Exemple de données synthétiques générées avec le ground truth associé

Les figures 5.3a et 5.3b montrent respectivement un extrait de données synthétiques générées et le « ground truth » correspondant obtenu grâce au générateur.

Configurations	DS
m_1	1,0
m_2	0,0
m_3	0,8

TABLE 5.3 – Valeur moyenne des précisions pour la configurations des données synthétiques

Informations	DS1	DS2	DS3
na	6	10	10
no	1000	1000	2000
ns	10	10	50
td	60.000	100.000	1.000.000

TABLE 5.4 – Information concernant les données synthétiques utilisées

Nous allons maintenant présenter les résultats des expérimentations sur les jeux de données synthétiques générés.

5.3.2 Présentation des expérimentations

Nous avons testé les différentes versions de l'algorithme sur les trois différents jeux de données synthétiques et avons mesuré leurs performances.

5.3.2.1 Evaluation de la correction de la version parallèle

Ici nous évaluons la correction de la version parallèle en comparant ses performances avec celles de la version séquentielle. Les tableaux 5.5, 5.6 et 5.7 présentent respectivement les performances des différentes versions sur DS1, DS2 et DS3. Nous avons aussi reporté dans la figures 5.4 l'étude comparative des précisions des versions parallèle et séquentielle de l'algorithme **TruthFinder**.

Données	Versions	Recall	Précision	Accuracy	F1-score	Temps(s)	Nbr Iter
DS1	**Séquentielle**	0.84	0.79	0.89	0.81	2700	2
	Parallèle	0.84	0.79	0.89	0.81	7,8	2

TABLE 5.5 – Performance des versions de l'algorithmes sur le jeu de données synthétiques DS1

Données	Versions	Recall	Précision	Accuracy	F1-score	Temps(s)	Nbr Iter
DS2	**Séquentielle**	0.93	0.88	0.92	0.90	9219	3
	Parallèle	0.93	0.88	0.92	0.90	20,79	3

TABLE 5.6 – Performance des versions séquentielle et parallèle de l'algorithmes sur le jeu de données synthétiques DS2

Données	Versions	Recall	Précision	Accuracy	F1-score	Temps(s)	Nbr Iter
DS3	**Séquentielle**	0.96	0.91	0.95	0.93	494049	2
	Parallèle	0.96	0.91	0.95	0.93	252	2

TABLE 5.7 – Performance des versions séquentielle et parallèle de l'algorithme sur le jeu de données synthétiques DS3

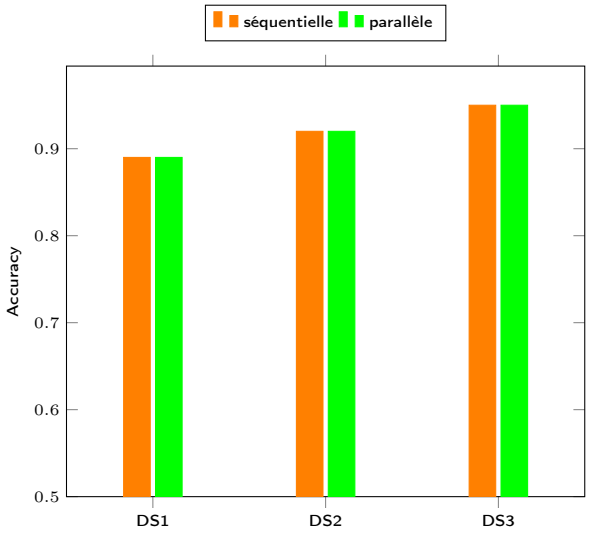

FIGURE 5.4 – Comparaison de la précision des Versions parallèle et séquentielle sur les données réelles Stock and Vols

5.3.2.2 Évaluation du temps d'exécution

Nous mesurons ici le coût en temps des versions séquentielles et parallèles de algorithme **Truth-Finder** que nous avons implémenté. Nous présentons les différents temps d'exécution exprimés en seconde (s) sur les différents jeux de données synthétiques DS1, DS2, DS3 dans le tableau 5.8 et dans la figure 5.5 une etude comparative des différentes versions. D'après les résultats, nous avons une réduction du temps d'exécution de 99,71% pour le jeux de donnée DS1 ; de 99,77% pour le jeux de données DS2 et de 99,99% pour le jeux de données DS3

Données	Versions	Temps(s)
DS1	Séquentielle	2700
	Parallèle	7,8
DS2	Séquentielle	9219
	Parallèle	20,79
DS3	Séquentielle	494049
	Parallèle	252

TABLE 5.8 – Temps d'exécution des versions séquentielle et parallèle de algorithme **TruthFinder** sur les jeux de données synthétiques DS1, DS2 et DS3

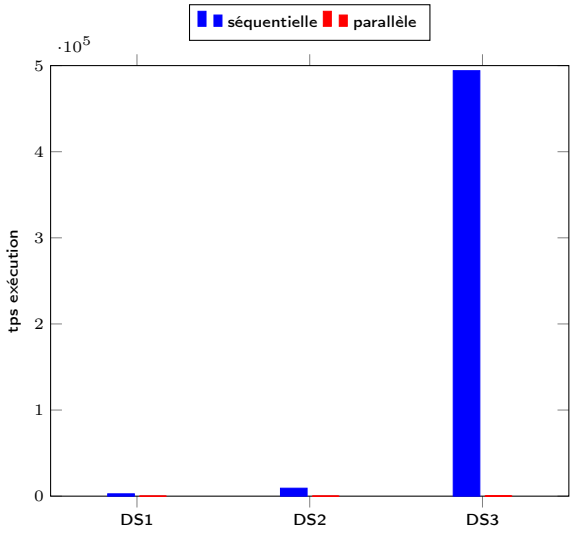

FIGURE 5.5 – Comparaison du temps d'exécution des Versions parallèle et séquentielle sur les données synthétiques

5.3.2.3 Evaluation de la mémoire consommée

Ici, nous évaluons le coût de la mémoire consommée par les différentes versions de l'algorithme dans le même environnement de test. Le tableau 5.9 présente les différents résultats des tests effectués. Nous remarquons que la mémoire consommée est réduite de 98,23% pour DS1 ; de 97,95% pour DS2 et de 99,66% pour DS3 par la version parallèle de l'algorithme.

Données	Versions	Mémoire(Go)
DS1	Séquentielle	0.0502
	Parallèle	0.0009
DS2	Séquentielle	0.0720
	Parallèle	0.0015
DS3	Séquentielle	0.7207
	Parallèle	0.0024

TABLE 5.9 – Consommation en mémoire des versions séquentielle et parallèle de algorithme **TruthFinder** sur les jeux de données synthétiques DS1, DS2 et DS3

5.4 Expérimentation sur des données réelles

Nous présentons dans cette section l'évaluation comparative des Versions parallèle et séquentielle de notre algorithme sur des données réelles. Le but est la validation pour les applications pratiques. À cet effet, nous considérons et utilisons les jeux de données réelles suivants :

— le jeu de donnée **Stocks** qui provient de l'article de Li et al. (Li et al., 2012)

— le jeu de données **Vols** de (Li et al., 2012).

5.4.1 Description des données

La description de chacun des jeux de données réelles se fait ci-dessous.

Stocks. Ce jeu de données contient des informations dans le domaine des stocks. Ces données ont été extraites à partir de 55 sources Web grâce à des extracteurs suivant les résultats de recherche avec les mots clés « stock price quotes » et « AAPL quotes » sur les moteurs *Google* et *Yahoo*. Ces sources comprennent certains agrégateurs financiers populaires tels que Yahoo Finance, Google Finance et MSN Money, des sites Web boursiers officiels comme le NASDAQ et des sites Web d'informations financiers tels que Bloomberg et Market Watch. La collecte de ces donées a été effectuée en 2011. Ce jeu est composé de 100 objets, ayant chacun 15 attributs pour un total de 56992 observations.

Vols. Ces données proviennent de sources différentes contenant 3 sites Web de compagnies aériennes (AA, UA, Continental), 8 sites Web d'aéroports (tels que SFO, DEN), et 27 sites Web de tiers notamment Orbitz et Travelocity. Les données ont été collectées en décembre 2011 et la partie utilisée dans ce travail contient 8771 observations avec 100 objets ayant chacun 6 attributs.

Dans le monde réel, les sources ne réagissent pas souvent par rapport à tous les attributs d'un objet. Cela fait que nous avons très souvent des données manquantes qui pourraient impacter les performances des algorithmes de recherche de vérité. Nous avons donc estimé le Taux de Couverture des Données (TCD) en pourcentage de chaque jeu de données réelles grâce à la formule suivante :

$$TCD = \left(1 - \frac{\sum_{o \in O}(|S_o| \times |A_o| - \sum_{s \in S_o} |A_{o-s}|)}{\sum_{o \in O}(|S_o| \times |A_o|)}\right) \times 100 \tag{5.4.1}$$

Données	Stocks	Vols
Nombre de sources	55	37
Nombre d'objets	100	100
Nombre d'attributs	15	6
Nombre d'observations	56992	8771
Taux de Couverture des Données (%)	75	66

TABLE 5.10 – Caractéristiques des jeux de données réelles

Le tableau 5.10 résume les caractéristiques des deux jeux de données réelles (Stocks et Vols) après pré-traitement.

5.4.2 Présentation des expérimentations

Nous avons testé et mesuré les performances de chaque version de l'algorithme **TruthFinder**, sur les différents jeux de données réelles présentés précédemment.

5.4.2.1 Evaluation de la correction de la version parallèle

Ici nous évaluons la correction de la version parallèle en comparant ses performances avec celles de la version séquentielle. Les tableau 5.11 et 5.12 présente les mesures de performance (précision, recall, accuracy, F1-score, temps d'exécution et nombre d'itération) des différentes versions à la sortie de ces tests. Nous avons aussi reporté dans le figure 5.6 l'étude comparative des précisions des versions parallèle et séquentielle de l'algorithme **TruthFinder**.

Données	Versions	Précision	Recall	Accuracy	F1-mesure	Temps(s)	Nbr Iter
Stocks	Séquentielle	0.84	0.27	0.47	0.41	5224	3
	Parallèle	0.76	0.38	0.49	0.51	8.40	3

TABLE 5.11 – Performances des différentes versions sur le jeux de données Stock

Données	Versions	Précision	Recall	Accuracy	F1-mesure	Temps(s)	Nbr Iter
Vols	Séquentielle	0.82	0.70	0.74	0.76	48	3
	parallèle	0.84	0.75	0.77	0.79	2.43	3

TABLE 5.12 – Performances des différentes versions sur le jeux de données Vols

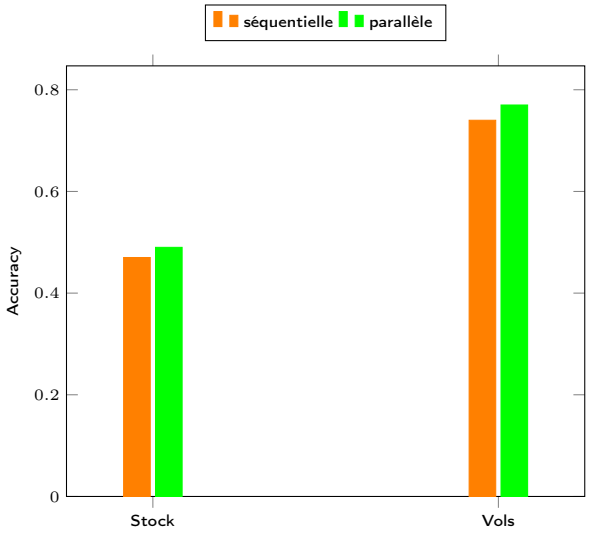

FIGURE 5.6 – Comparaison de la précision des Versions parallèle et séquentielle sur les données réelles Stock and Vols

5.4.2.2 Évaluation du temps d'exécution

Dans cette partie, Nous mesurons le coût en temps des versions séquentielle et parallèle de algorithme **TruthFinder** que nous avons implémenté. Comme dit plus haut la versions parallèle a été implémentée avec la librairie PySpark en utilisant le framework Apache Spark et exécutées dans le même environnement de test que la version séquentielle. Le tableau 5.13, présente les différents temps d'exécution exprimés en seconde (s) sur les jeux de données réelles Stock et Vols. Nous faisons aussi une étude comparative de la précision dans la figure 5.7

Concernant les résultats pour le jeux de données Vols la réduction du temps est de 94,43% et pour le jeux de données Stocks est de 99,83%. C'est-à-dire que la version parallèle est de meilleure en terme de temps d'exécution.

Données	Versions	Temps(s)
Vols	Séquentielle	48
	Parallèle	2.43
Stocks	Séquentielle	5227
	Parallèle	8.40

TABLE 5.13 – Temps d'exécution des versions séquentielle et parallèle de algorithme **TruthFinder** sur les jeux de données réelles Stocks et Vols

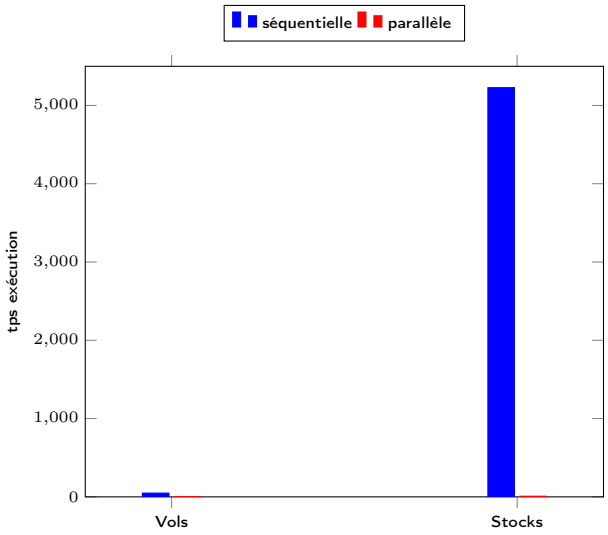

FIGURE 5.7 – Comparaison du temps d'exécution des Versions parallèle et séquentielle sur les données réelles

5.4.2.3 Evaluation de la mémoire consommée

Nous évaluons ici la mémoire consommée par chaque version de notre algorithme sur les jeux de donnée réelles. Les résultats sont présentés dans le tableau 5.14. D'après ces résultats nous constatons une réduction de la consommation en mémoire de 90,90% pour le jeux de données Vols et de 92,66% pour le jeux de données.

Données	Versions	Mémoire(Go)
Vols	**Séquentielle**	0.0010
	Parallèle	0.0001
Stocks	**Séquentielle**	0.0101
	Parallèle	0.0008

TABLE 5.14 – Consommation en mémoire des versions séquentielle et parallèle de algorithme **TruthFinder** sur les jeux de données réelles Stocks et Vols

5.5 Analyse des résultats et discussions

Les sections précédentes détaillent les expérimentations effectuées sur les différents jeux de données avec des configurations variées pour estimer et comparer les performances de la version proposée dans ce travail et celle de la version séquentielle. Nous effectuons les tests sur 2 catégories de données tels que les jeux de données synthétiques et les jeux de données réelles. Nous avons différentes configurations pour chaque jeux de données (3 pour les données synthétiques et 2 pour les données réelles) ont été considérées dans le but de prendre en compte l'hypothèse de base de ce travail. Nous notons une grande diversité en termes de caractéristiques (nombre de sources, d'objets et d'observations, taux de couverture des données, etc dans les jeux de données utilisés. Nous proposons une analyse plus fine de ces résultats dans ce qui suit pour une meilleure compréhension de la version parallèle par rapport à la version existante.

5.5.1 Comparaison entre la version parallèle et la version séquentielle

Dans ce travail, la version parallèle a été implémentée avec la librairie PySpark en utilisant le framework MapReduce et exécuté dans le même environnement de test que la version séquentielle. Nous remarquons que les deux versions de l'algorithme ont les mêmes performance (Accuracy, Precision, Recall, F1-score) quelque soit le volume du jeux de données. Nous pouvons dire que la version parallèle est aussi performantes que la version séquentielle concernant les différentes métriques. Concernant le temps d'exécution en seconde (s) et l'utilisation de la mémoire, la version séquentielle sont **très élevés** par rapport au temps d'exécution et l'utilisation de la mémoire de la version parallèle. La version parallèle s'exécute de manière très rapide par rapport à la version et utilise moins de mémoire. Ce qui prouve l'efficacité de la version parallèle en terme de temps d'exécution.

6. Conclusion et perspectives

Au terme de notre travail qui porte sur la parallélisation des algorithmes de recherche de vérité a l'ère de grande données. L'objectif ultime de notre travail était de proposer à l'échelle de grands volume de données une version parallèle des algorithmes de recherche de vérité en utilisant MapReduce sous Hadoop. En plus Proposer une validation sur les données synthétiques et réelles en prouvant sa rapidité et son efficacité en fonction du temps d'exécution et de l'utilisation de la mémoire. Pour ce fait nous avons proposé une classification des algorithmes de recherche de vérité en justifiant si un algorithme est parallélisable ou non parallélisable. Ainsi nous avons proposé une version parallélisée de l'algorithme de vérité **TruthFinder** et fait une comparaison avec sa version séquentielle. Nous évaluons les performances de chaque versions sur les données synthétiques de différentes tailles (60000, 100000 et 1000000) et sur les jeux de données réelles en terme de temps d'exécution, de l'utilisation de la mémoire, de précision, accuracy etc. Enfin nous avons montrer que la version parallèle de l'algorithme s'exécute très rapidement que la version séquentielle de l'algorithme et utilise moins de mémoire.

Cependant, nous faisons une exécution avec un système à un seul noeud dans l'espoir d'avoir de bon résultats qui est la réduction du temps d'exécution à 99% mais avec un système à plusieurs noeuds, nous pensons que la réduction du temps d'exécution sera plus exponentielle. Malheureusement cette tache n'a pas pu être traitée par manque de ressources. En guise de perspective, nous envisageons faire une exécution avec un système à plusieurs noeuds en présence d'une taille de données plus grande.

7. Annexe

Algorithme 5 Algorithme TruthFinder : TruthFinder(S,O,A,V,n,c,$\alpha, \rho, \gamma, \delta$)

Pré-conditions : $\forall s \in S : T_s \leftarrow 0.8$

répéter
  Pour tout $o \in O$ **faire**
    Pour tout $a \in A_o$ **faire**
      Pour tout $v \in V_{a-o}$ **faire**
        $\sigma_v \leftarrow -\sum_{s \in S_v} \ln(1 - T_s)$
        $\sigma_v^* \leftarrow \sigma_v + \rho \sum_{v' \in V_{a-o}} \sigma'_v \times sim(v, v')$
        $C_v \leftarrow 1/(1 + \exp(-\gamma \sigma_v^*))$
      Fin Pour
    Fin Pour
  Fin Pour
  Pour tout $s \in S$ **faire**
    $T_s \leftarrow \sum_{v \in V_s} C_v / |V_s|$
  Fin Pour
jusqu'à convergence(T_s, δ)
Pour tout $o \in O$ **faire**
  Pour tout $a \in A_o$ **faire**
    $vraiValeur(a - o) \leftarrow argmax_{v \in V_{a-o}}(C_v)$
  Fin Pour
Fin Pour

References

Aarnio, T. Parallel data processing with mapreduce. In *TKK T-110.5190, Seminar on Internetworking*, 2009.

Ba, M. L., Horincar, R., Senellart, P., and Wu, H. Truth Finding with Attribute Partitioning. In *WebDB*, pages 27–33, Melbourne, Australia, May 2015. URL https://hal-imt.archives-ouvertes.fr/hal-01178403.

Berti-Équille, L. Truth discovery, 2018.

circuit today. carry ripple adder. adder, https://www.circuitstoday.com/ripple-carry-adder/, Accessed in Aug 2018.

Dong, X. L., Berti-Equille, L., and Srivastava, D. Integrating conflicting data : The role of source dependence. *Proc. VLDB Endow.*, 2(1) :550–561, aug 2009. ISSN 2150-8097. doi : 10.14778/1687627.1687690. URL https://doi.org/10.14778/1687627.1687690.

elprocus. carry look ahead adder. adder, https://www.elprocus.com/carry-look-ahead-adder/, Accessed 2013 - 2022.

Galland, A., Abiteboul, S., Marian, A., and Senellart, P. Corroborating information from disagreeing views. In *WSDM 2010 - Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, WSDM 2010 - Proceedings of the 3rd ACM International Conference on Web Search and Data Mining, pages 131–140, 2010. ISBN 9781605588896. doi : 10.1145/1718487.1718504. 3rd ACM International Conference on Web Search and Data Mining, WSDM 2010 ; Conference date : 03-02-2010 Through 06-02-2010.

gatevidyalay. carry ripple adder. adder, https://www.gatevidyalay.com/ripple-carry-adder/, Accessed 2020.

Lamine Ba, M., Horincar, R., Senellart, P., and Wu, H. Truth finding with attribute partitioning. In *Proceedings of the 18th International Workshop on Web and Databases*, WebDB'15, page 27–33, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336277. doi : 10.1145/2767109.2767118. URL https://doi.org/10.1145/2767109.2767118.

Li, X., Dong, X. L., Lyons, K., Meng, W., and Srivastava, D. Truth finding on the deep web : Is the problem solved ? 6(2) :97–108, Dec. 2012. ISSN 2150-8097. doi : 10.14778/2535568.2448943. URL https://doi.org/10.14778/2535568.2448943.

Li, Y., Gao, J., Meng, C., Li, Q., Su, L., Zhao, B., Fan, W., and Han, J. A survey on truth discovery. *SIGKDD Explor. Newsl.*, 17(2) :1–16, feb 2016. ISSN 1931-0145. doi : 10.1145/2897350.2897352. URL https://doi.org/10.1145/2897350.2897352.

Osias Noël Nicodème Finagnon, T. and Ba, M. Td-ac : Efficient data partitioning based truth discovery. 03 2021. doi : 10.5441/002/edbt.2021.37.

Ouyang, R. W., Kaplan, L. M., Toniolo, A., Srivastava, M., and Norman, T. J. Parallel and streaming truth discovery in large-scale quantitative crowdsourcing. *IEEE transactions on parallel and distributed systems*, 27(10) :2984–2997, 2016.

Pasternack, J. and Roth, D. Latent credibility analysis. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, page 1009–1020, New York, NY, USA, 2013a. Association for Computing Machinery. ISBN 9781450320351. doi : 10.1145/2488388.2488476. URL https://doi.org/10.1145/2488388.2488476.

Pasternack, J. and Roth, D. Latent credibility analysis. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1009–1020, 2013b.

Site Web Infoq. Traitements big data avec apache spark - 1ère partie : Introduction. Spark, https://www.infoq.com/fr/articles/apache-spark-introduction/, Accessed in April 2020.

Site Web Python. Python. Python, https://www.python.org/, Accessed in May 2020.

SparkSiteWeb. Apache spark is a unified analytics engine for large-scale data processing. Spark, https://spark.apache.org/, Accessed in April 2020.

Waguih, D. A. and Berti-Equille, L. Truth discovery algorithms : An experimental evaluation, 2014.

Wang, D., Kaplan, L., Le, H., and Abdelzaher, T. On truth discovery in social sensing : A maximum likelihood estimation approach. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*, IPSN '12, page 233–244, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312271. doi : 10.1145/2185677.2185737. URL https://doi.org/10.1145/2185677.2185737.

Wang, Y., Ma, F., Su, L., and Gao, J. Discovering truths from distributed data. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 505–514, 2017. doi : 10.1109/ICDM.2017.60.

WatElectronics.com. carry l.a adder. CLAadder, https://www.watelectronics.com/carry-lookahead-adder/, Accessed in December 30, 2021.

Yin, X. and Tan, W. Semi-supervised truth discovery. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 217–226, New York, NY, USA, 2011a. Association for Computing Machinery. ISBN 9781450306324. doi : 10.1145/1963405.1963439. URL https://doi.org/10.1145/1963405.1963439.

Yin, X. and Tan, W. Semi-supervised truth discovery. In *Proceedings of the 20th international conference on World wide web*, pages 217–226, 2011b.

Yin, X., Han, J., and Yu, P. Truth discovery with multiple conflicting information providers on the web. *Knowledge and Data Engineering, IEEE Transactions on*, 20 :796 – 808, 07 2008. doi : 10.1109/TKDE.2007.190745.

Zhao, B., Rubinstein, B. I., Gemmell, J., and Han, J. A bayesian approach to discovering truth from conflicting sources for data integration. *arXiv preprint arXiv :1203.0058*, 2012.