جامعة سيدي محمد بن عبد الله بفاس
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

N° d'ordre :19/2023

Centre des Etudes Doctorales : Sciences et Techniques et Sciences Médicales
Formation Doctorale : Sciences de l'Ingénieur, Sciences Physiques, Mathématiques et Informatique
Discipline : Informatique
Spécialité : Informatique
Laboratoire : Systèmes Intelligents et Applications

Thèse pour l'obtention de DOCTORAT

Modélisation des performances des travaux MapReduce géo-distribués à l'aide du Deep Learning

Préparée par : Mr BERGUI Mohammed

Soutenue le 18 mars 2023 devant le jury :

Nom et Prénom	Grade	Etablissement	Qualité
MRABTI Fatiha	PES	Faculté des Sciences et Techniques de Fès	Président
ABDELHAMID Zouhair	PH	Faculté des Sciences et Techniques de Tanger	Rapporteur
EN-NAHNAHI Noureddine	PH	Faculté des Sciences Dhar El Mehraz de Fès	Rapporteur
ABOUNAIMA Med Chaouki	PH	Faculté des Sciences et Techniques de Fès	Rapporteur
ZAHI Azeddine	PES	Faculté des Sciences et Techniques de Fès	Examinateur
NAJAH Said	PH	Faculté des Sciences et Techniques de Fès	Directeurs de thèse

Résumé de la thèse

L'analyse de données géo-distribuées a suscité un grand intérêt ces dernières années en raison du besoin croissant de dériver des connaissances à partir de données géo-distribuées. Bien que les applications de traitement en cluster, telles que MapReduce et Spark, aient été largement déployées dans les centres de données pour prendre en charge les applications commerciales et la recherche scientifique, elles ne parviennent pas à répondre aux exigences de performance de la plupart des applications. De plus, elles entraînent un gaspillage des ressources en raison des différences inhérentes entre les environnements, notamment la nature éparse, hautement hétérogène et dynamique des ressources distribuées : puissance de calcul et bande passante du réseau.

Ce type d'analyse géo-distribuée implique le transfert de données sur les liens du réseau étendu (WAN) entre différents centres de données. La nature de ces liens est très limitée et hétérogène, ce qui rend le transfert de données lent et coûteux. Dans cette thèse, nous passons en revue les systèmes de traitement de big data géo-distribués qui prennent en compte la bande passante du WAN et nous fournissons les avantages et les inconvénients de la plupart de ces systèmes. En outre, nous les classons en fonction de la technique de traitement et du framework sur lesquels ils sont basés et nous les comparons en fonction de plusieurs caractéristiques. Nous identifions les travaux futurs, tels que l'utilisation de l'apprentissage automatique pour améliorer la planification des tâches.

Puisque l'une des fonctions les plus critiques de Hadoop est la gestion des ressources. Une gestion plus efficace sera obtenue si l'estimation du temps d'exécution des travaux est faite avec précision. Ainsi, dans cette thèse, nous proposons d'utiliser l'apprentissage automatique pour prédire le temps d'exécution des travaux MapReduce géo-distribué. Pour cela, nous proposons une nouvelle base de données de traces de travaux MapReduce version dans le cloud avec une bande passante réseau limitée. Nous présentons la méthodologie et le dispositif expérimental que nous avons adoptés pour créer la base de données, puis nous décrivons le processus de génération et de collecte de cette base.

Par la suite, nous étudions l'anatomie de MapReduce ainsi que les caractéristiques qui impactent les performances. Nous proposons ensuite un réseau de neurones profond pour prédire le temps d'exécution des travaux MapReduce. Le résultat est prometteur car il montre que l'apprentissage profond peut être appliqué comme solution à ce problème et, avec suffisamment de données, peut atteindre une grande précision.

Mots-clés : Analyse de big data géo-distribuée, hadoop, mapreduce, cloud computing, bande passante WAN, modélisation des performances, apprentissage profond.

Abstract

There has been a lot of interest in geo-distributed data analytics in recent years due to the increasing need to derive insights from geo-distributed data. Although cluster computing applications, such as MapReduce and Spark, have been widely deployed in data centres to support business applications and scientific research, they are not designed to perform jobs across geo-distributed data centres and fail to achieve the performance requirements of most applications and lead to resource waste due to the inherent differences between environments, including the sparse, highly heterogeneous, and dynamic nature of the distributed resources: computing power and network bandwidth.

This kind of geo-distributed analysis (GDA) implies transferring data over the wide area network (WAN) links between different processing data centres. The nature of these WAN links is highly constrained and heterogeneous, making the transfer of data over the WAN slow and expensive. In this thesis, we review geo-distributed big data processing frameworks with WAN bandwidth awareness and provide the advantages and disadvantages of most of these frameworks. In addition, we classify them according to the processing technique and the framework they are based on and compare them according to several characteristics. We identify open issues and future work, such as the use of machine learning to improve scheduling and task placement.

Since one of the most critical functions of Hadoop is job and resource management. More efficient management will be achieved if the estimation and prediction of the execution time of a job are done accurately. Thus, in this thesis, we propose to use machine learning to predict the execution time of MapReduce jobs in a Hadoop geo-distributed cluster. For this reason, we propose a new Hadoop MapReduce version 2 job trace database in a cloud environment with limited network bandwidth. We present the methodology and experimental setup we adopted to create the database and then describe the process of generating and collecting this database.

Subsequently, we study the Hadoop architecture and anatomy of MapReduce and the features that impact its performance at each stage. We then propose a deep neural network DNN to predict the execution time of MapReduce jobs based on our database. We then propose a deep neural network DNN to predict the execution time of MapReduce jobs based on our database. The result is promising as it shows that deep learning can be applied as a solution to this problem and, with enough data, can achieve high accuracy.

Keywords: Geo-distributed big data analytics, hadoop, mapreduce, cloud computing, WAN bandwidth, performance modeling, deep learning

Dedication

To my beloved family.

Acknowledgements

Many people have earned my gratitude for their contribution to my time as a PhD candidate. First of all, I would like to thank my doctoral advisers, Prof. Said Najah and Dr. Nikola S. Nikolov, for their guidance and support over the last six years. They introduced me to research and patiently taught me how to think critically, approach a problem, and improve my communication skill. Such invaluable lessons will be very helpful in my future career.

Besides my advisers, I would also like to thank Prof. Fatiha Mrabti, Prof. Zouhair Abdelhamid, Prof. Noureddine En-Nahnahi, Prof. Med Chaouki Abounaima and Prof. Azeddine Zahi for serving as my thesis and preliminary examination committees and providing constructive feedback throughout my degree progress. I am also grateful to Prof. Fatiha Mrabti and Prof. Arsalane Zerghili who supported and encouraged me to pursue a doctoral degree.

I would also like to thank my friends and colleagues at the Faculty of Sciences and Technologies in Fez and the University of Limerick. In particular, I am grateful to Brahim Ait Skourt, Dr. Soufiane Hourri, Ilham El Ouariachi, Dr. Fatima Zahrae Zenkouar and Brian Halpin for making the lab a family-like working environment. The informal discussions with them always proved to be very enlightening and fruitful. I will always cherish the memories I made with them in pursuing our PhD careers together.

Finally and most importantly, I would like to thank my family. I am extremely grateful to my parents, Driss Bergui and Nadia Ouadahi, for their unconditional love, faith, support, and sacrifice. I will forever be indebted to them. I am also genuinely thankful to my sister, Imane Bergui M.D, who has always been very supportive. Everything that I have achieved in life would have been impossible without my family.

Remerciements

De nombreuses personnes ont mérité ma gratitude pour leur contribution à mon parcours de Doctorant. Tout d'abord, je tiens à remercier mes directeurs de thèse, le professeur Said Najah et le docteur Nikola S. Nikolov, pour leurs conseils et leur soutien au cours des six dernières années. Ils m'ont initié à la recherche et m'ont patiemment appris comment réfléchir de manière critique, à aborder un problème et à améliorer ma capacité de communication. Ces leçons inestimables me seront très utiles dans ma future carrière.

Outre mes directeurs de thèse, je voudrais également remercier l'ensemble des Professeurs membres du jury, le Prof. Fatiha Mrabti, le Prof. Zouhair Abdelhamid, le Prof. Noureddine En-Nahnahi, le Prof. Med Chaouki Abounaima et le Prof. Azeddine Zahi pour l'intérêt qu'ils ont porté à ma thèse et de m'avoir fait part de leurs commentaires constructifs. Je suis également reconnaissant envers les Professeurs Fatiha Mrabti et Arsalane Zerghili qui m'ont soutenue et encouragé à poursuivre un cursus de doctorat.

Je tiens également à remercier mes amis et collègues de la Faculté des Sciences et Technologies de Fès et de l'Université de Limerick. En particulier, je suis reconnaissant à Brahim Ait Skourt, Dr. Soufiane Hourri, Ilham El Ouariachi, Dr. Fatima Zahrae Zenkouar et Biran Halpin pour avoir fait du laboratoire un environnement de travail familial. Les discussions informelles avec eux se sont toujours révélées très instructives et fructueuses. Je chérirai toujours les souvenirs que j'ai partagé avec eux dans la poursuite de nos carrières de doctorat ensemble.

Enfin, et surtout, je tiens à remercier ma famille. Je suis extrêmement reconnaissant à mes parents, Driss Bergui et Nadia Ouadahi, pour leur amour inconditionnel, leur foi, leur soutien et leurs sacrifices, je leur serai à jamais redevable. Je suis également très reconnaissant à ma sœur, Docteur Imane Bergui, qui m'a toujours soutenu. Tout ce que j'ai accompli dans la vie aurait été impossible sans eux.

Abstract

There has been a lot of interest in geo-distributed data analytics in recent years due to the increasing need to derive insights from geo-distributed data. Although cluster computing applications, such as MapReduce and Spark, have been widely deployed in data centres to support business applications and scientific research, they are not designed to perform jobs across geo-distributed data centres and fail to achieve the performance requirements of most applications and lead to resource waste due to the inherent differences between environments, including the sparse, highly heterogeneous, and dynamic nature of the distributed resources: computing power and network bandwidth.

This kind of geo-distributed analysis (GDA) implies transferring data over the wide area network (WAN) links between different processing data centres. The nature of these WAN links is highly constrained and heterogeneous, making the transfer of data over the WAN slow and expensive. In this thesis, we review geo-distributed big data processing frameworks with WAN bandwidth awareness and provide the advantages and disadvantages of most of these frameworks. In addition, we classify them according to the processing technique and the framework they are based on and compare them according to several characteristics. We identify open issues and future work, such as the use of machine learning to improve scheduling and task placement.

Since one of the most critical functions of Hadoop is job and resource management. More efficient management will be achieved if the estimation and prediction of the execution time of a job are done accurately. Thus, in this thesis, we propose to use machine learning to predict the execution time of MapReduce jobs in a Hadoop geo-distributed cluster. For this reason, we propose a new Hadoop MapReduce version 2 job trace database in a cloud environment with limited network bandwidth. We present the methodology and experimental setup we adopted to create the database and then describe the process of generating and collecting this database.

Subsequently, we study the Hadoop architecture and anatomy of MapReduce and the features that impact its performance at each stage. We then propose a deep neural network DNN to predict the execution time of MapReduce jobs based on our database. We then propose a deep neural network DNN to predict the execution time of MapReduce jobs based on our database. The result is promising as it shows that deep learning can be applied as a solution to this problem and, with enough data, can achieve high accuracy.

Résumé

L'analyse de données géo-distribuées a suscité un grand intérêt ces dernières années en raison du besoin croissant de dériver des connaissances à partir de ces données. Bien que les applications de traitement en cluster, telles que MapReduce et Spark, aient été largement déployées dans les centres de données pour prendre en charge les applications commerciales et la recherche scientifique, elles ne parviennent pas à répondre aux exigences de performance de la plupart des applications. De plus, elles entraînent un gaspillage des ressources en raison des différences inhérentes entre les environnements, notamment la nature éparse, hautement hétérogène et dynamique des ressources distribuées : puissance de calcul et bande passante du réseau.

Ce type d'analyse géo-distribuée implique le transfert de données sur les liens du réseau étendu (WAN) entre différents centres de données. La nature de ces liens est très limitée et hétérogène, ce qui rend le transfert de données lent et coûteux. Dans cette thèse, nous passons en revue les systèmes de traitement de big data géo-distribués qui prennent en compte la bande passante du WAN et nous fournissons les avantages et les inconvénients de la plupart de ces systèmes. En outre, nous les classons en fonction de la technique de traitement et du framework sur lesquels ils sont basés et nous les comparons en fonction de plusieurs caractéristiques. Nous identifions les travaux futurs, tels que l'utilisation de l'apprentissage automatique pour améliorer la planification des tâches.

Puisque l'une des fonctions les plus critiques de Hadoop est la gestion des ressources, une gestion plus efficace sera obtenue si l'estimation du temps d'exécution des travaux est faite avec précision. Ainsi, dans cette thèse, nous proposons d'utiliser l'apprentissage automatique pour prédire le temps d'exécution des travaux MapReduce géo-distribués. Pour cela, nous proposons une nouvelle base de données de traces de travaux MapReduce version dans le cloud avec une bande passante réseau limitée, nous présentons la méthodologie et le dispositif expérimental que nous avons adoptés pour créer la base de données, puis nous décrivons le processus de génération et de collecte de cette base.

Par la suite, nous étudions l'anatomie de MapReduce ainsi que les caractéristiques qui impactent la performance, nous proposons ensuite un réseau de neurones profond pour prédire le temps d'exécution des travaux MapReduce. Le résultat est prometteur car il montre que l'apprentissage profond peut être appliqué comme solution à ce problème et avec suffisamment de données peut atteindre une grande précision.

Sommaire

Liste des Figures **x**

Liste des Tableaux **xi**

Liste des Abréviations **xii**

1 Introduction **1**
- 1.1 Contexte . . . 1
- 1.2 Questions de recherche . . . 4
- 1.3 Contributions . . . 5
- 1.4 Déroulement et structure de la thèse . . . 6

2 Notions essentielles de l'écosystème Hadoop **7**
- 2.1 Introduction . . . 7
- 2.2 MapReduce . . . 8
- 2.3 Apache Hadoop . . . 9
  - 2.3.1 Architecture de Hadoop . . . 10
  - 2.3.2 Hadoop Distributed File System . . . 10
  - 2.3.3 MapReduce . . . 11
  - 2.3.4 Yet Another Resource Negotiator . . . 12
- 2.4 Apache Hive . . . 14
- 2.5 Apache Spark . . . 16
  - 2.5.1 Resilient Distributed Dataset . . . 16
  - 2.5.2 Spark Streaming . . . 16
- 2.6 Apache Flink . . . 17
- 2.7 Conclusion . . . 17

3 Les frameworks big data géo-distribués tenant compte de la bande passante **18**
- 3.1 Introduction . . . 19
- 3.2 Méthodologie . . . 21
- 3.3 Motivations . . . 22
  - 3.3.1 Support des Applications Géo-distribuées . . . 22

3.3.2 Défaillance ou Panne du Centre de Données . . . 23
3.3.3 Contraintes réglementaires . . . 23
3.4 Défis . . . 23
3.4.1 Contraintes liées au réseau étendu WAN . . . 24
3.4.2 Hétérogénéité des Clusters . . . 24
3.4.3 Tailles de données hétérogènes . . . 24
3.5 Traitement Big Data Géo-distribué . . . 25
3.5.1 Frameworks de traitement par lots géo-distribués . . . 26
3.5.1.1 Frameworks basés sur MapReduce . . . 26
3.5.1.2 Frameworks basés sur Spark . . . 28
3.5.1.3 Frameworks de traitement de type SQL . . . 35
3.5.1.4 Autres frameworks de traitement par lots . . . 38
3.5.2 Frameworks de traitement de flux géo-distribués . . . 40
3.5.2.1 Frameworks de traitement par micro-lots . . . 40
3.5.2.2 Frameworks de traitement de flux natifs . . . 41
3.6 Conclusion . . . 45

4 Base de données Hadoop pour l'estimation des tâches dans le cloud avec une bande passante limitée 47
4.1 Introduction . . . 47
4.2 Méthodologie . . . 48
4.3 Dispositif expérimental . . . 49
4.4 La base de données des traces des travaux Hadoop MapReduce . . . 50
4.4.1 Génération de données synthétiques . . . 50
4.4.1.1 Génération de données dans HDFS. . . . 51
4.4.1.2 Description des travaux Hadoop. . . . 52
4.4.2 Collecte des données . . . 53
4.5 Conclusion . . . 57

5 Prédiction du temps d'exécution a l'aide du Deep Learning 58
5.1 Introduction . . . 58
5.2 Travaux connexes . . . 59
5.2.1 MapReduce 1 . . . 60
5.2.1.1 Grey Box . . . 60
5.2.1.2 Black Box . . . 61
5.2.2 MapReduce 2 . . . 62
5.2.2.1 Grey Box . . . 62
5.2.2.2 Black Box . . . 62
5.3 Contexte et motivation . . . 63
5.4 Construction du modèle . . . 65

5.4.1 Pré-traitement des données 65
5.4.1.1 Évaluation de la qualité des données 65
5.4.1.2 Nettoyage des données 66
5.4.1.3 Sélection des caractéristiques 66
5.4.2 Modèle d'apprentissage profond 66
5.5 Résultats et discussion 69
5.6 Conclusion 70

6 Conclusions et Perspectives **71**
6.1 Conclusions 71
6.2 Perspectives 73

Bibliographie **75**

Liste des Figures

2.1 Flux de données logique du modèle de programmation MapReduce. 8
2.2 Architecture de HDFS. 11
2.3 Exécution de l'algorithme MapReduce. 12
2.4 Architecture générale de YARN. 13
2.5 Principaux composants de l'architecture Hive. 15

3.1 Modèle de traitement centralisé vs. décentralisé des données géo-distribuées. 20
3.2 Architecture centralisée vs. décentralisée des systèmes géo-distribués. 25
3.3 L'architecture du système GeoDis. 27
3.4 Le cycle de vie d'un travail dans Houtu. 33
3.5 Programme DAG avec différentes stratégies de répartition. 35
3.6 Vue d'ensemble de Turbo. 38
3.7 L'architecture du système Sana. 43

4.1 Modèle de données TPCx-BB. 51

5.1 Pipeline de traitement MapReduce. 63
5.2 L'architecture du réseau de neurone profond (DNN) utilisé dans la méthode proposée. 69

Liste des Tableaux

3.1 Comparaison des techniques de traitement des données géo-distribuées 29

4.1 Les différents types de VM pour les déploiements de clusters Hadoop 50
4.2 Noms et tailles des tables de la base de données du benchmark . . . 52
4.3 La répartition des différents types de requêtes et des types de données 53
4.4 Caractéristiques des applications, cluster et YARN collectées 54
4.5 Caractéristiques des travaux collectés 55
4.6 Caractéristiques des compteurs de travaux collectés 56
4.7 Caractéristiques des tâches collectées 57

5.1 Liste des caractéristiques sélectionnées. 67
5.2 Précision de notre DNN pour différentes quantités de travaux. . . . 70

Liste des Abréviations

ADMM . . . . Alternating Direction Method of Multipliers

AF . . . . . Adaptive Feedback

API . . . . . Application Programming Interface

ARIA . . . . . Automatic Resource Inference and Allocation

ARIMA . . . . AutoRegressive Integrated Moving Average

AWS . . . . . Amazon Web Services

CD . . . . . Centre de Données

CNN . . . . . Convolutional Neural Network

CPU . . . . . Central Processing Unit

CSV . . . . . Comma-Separated Values

DAG . . . . . Directed Acyclic Graph

DC . . . . . Data Centre

DN . . . . . Data Node

DNN . . . . . Deep Neural Network

E/S . . . . . Entrée / Sortie

EC2 . . . . . Elastic Compute 2

FiFo . . . . . First in First out

GBRT . . . . . Gradient Boosting Regression Tree

GCP . . . . . Google Cloud Platform

GDA . . . . . Geographically Distributed big data Analytics

HDFS . . . . . Hadoop Distributed File System

HDM-MC . . Hierarchically Distributed Data Matrix - Multi Cluster

HTTP . . . . . Hypertext Transfer Protocol

I/O . . . . . Input / Output

ILP . . . . . Integer Linear Programming

JDBC . . . . . Java DataBase Connectivity

JM . . . . . Job Manager

JSON . . . . . JavaScript Object Notation

JVM . . . . . Java Virtual Machine

LAN . . . . . Local Area Network

LASSO . . . . Least Absolute Shrinkage and Selection Operator

LP . . . . . Linear Programming

LWLR . . . . . Locally Weighted Linear Regression

MAE . . . . . Mean Absolute Error

MDRF . . . . Maximum Data Reduction First

MDRRF . . . Maximum Data Reduction Rate First

MIP . . . . . Mixed Integer Programming

ML . . . . . Machine Learning

MR . . . . . MapReduce

NLP . . . . . Natural Language Processing

NM . . . . . Node Manager

NN . . . . . Name Node

ODBC . . . . . Open DataBase Connectivity

OLAP . . . . . Online Analytical Processing

PDGF . . . . . Parallel Data Generation Framework

QEP . . . . . Query Execution Plan

RAM . . . . . Random Acess Memory

RDD . . . . . Resilient Distributed Dataset

ReLU . . . . . Rectified Linear Unit

REST . . . . . REprensentational State Transfer

RM . . . . . Resource Manager

RNN . . . . . Recurrent Neural Network

SCTF . . . . . Shortest Completion Time First

SlidHMM . . . Sliding Hidden Markov Model

SQL . . . . . Structured Query Language

SRPT . . . . . Shortest Remaining Processing Time

SSH . . . . . Secure Shell Protocol

SVM . . . . . Support Vector Machine

TM . . . . . Task Manager

TPC . . . . . Transaction Processing Performance

UDF . . . . . User Defined Function

UDTF . . . . . User Defined Table generating Function

vCPU . . . . . virtual Central Processing Unit

VM . . . . . Virtual Machine

WAN . . . . . Wide Area Network

XML . . . . . eXtensible Markup Language

YARN . . . . . Yet Another Resource Negotiator

1 Introduction

Sommaire

1.1 Contexte 1
1.2 Questions de recherche 4
1.3 Contributions 5
1.4 Déroulement et structure de la thèse 6

1.1 Contexte

Nous vivons à l'ère du Big Data. Aujourd'hui, une énorme quantité de données est générée partout en raison des progrès de l'Internet et des technologies de communication et des intérêts des personnes utilisant des smartphones, des médias sociaux, l'Internet des objets, des dispositifs de détection, des services en ligne et bien d'autres. De même, en raison de l'amélioration des applications de données et de la large diffusion des logiciels, plusieurs organisations gouvernementales et commerciales telles que les institutions financières, les organisations de santé, les départements d'éducation et de recherche, les secteurs de l'énergie, les secteurs de la vente au détail, les sciences de la vie et les départements environnementaux produisent tous une énorme quantité de données chaque jour. Par exemple, International Data Corporation (IDC) a signalé que 62,2 ZB (zettabytes) de données de l'univers ont été crées, capturées, copiées et consommées dans le monde en 2010

et que ce chiffre atteindra 180 ZB en 2025[1]. Ces énormes bases de données n'incluent pas seulement des données structurées, mais plus de 75 % des bases de données comprennent des données brutes, semi-structurées et non structuré[1]. Cette quantité massive de données sous différents formats peut être considérée comme du Big Data.

Bien que la définition du Big Data ait fait l'objet d'un débat, un accord des "5V", est progressivement adopté sur la base des caractéristiques du Big Data [1–3]. La définition des "5V" est largement reconnue car elle présente la signification et la nécessité des Big Data. Les caractéristiques des Big Data basées sur la définition des "5V" peuvent être détaillées comme suit.

- La variété représente les différents types de données collectées par les réseaux sociaux et les dispositifs terminaux, tels que les capteurs, les dispositifs portables, les smartphones, les moniteurs. L'organisation des données est plus compliquée, et le contenu des données est plus varié. De plus, les types de données sont généralement multiplexés, y compris l'image, le texte, la vidéo, l'audio, les logs et ainsi de suite [4].

- La vélocité représente la vitesse élevée à laquelle les données sont générées, transférées et traitées. Pour certaines applications, l'arrivée et le traitement des données peuvent être effectués par lots, d'autres applications, comme les programmes d'analyse, nécessitent un traitement en temps réel et en continu [5].

- Le volume représente la quantité de tous les types de données qui sont générées à partir de différentes sources. Avec le développement des technologies de cloud computing, l'énorme quantité de données produites par le stockage en cloud, les services mobiles et l'Internet des objets, fait également exploser le volume des données.

- La valeur est la caractéristique la plus importante du big data. Elle représente le processus d'extraction des valeurs importantes cachées dans ces énormes bases de données de différents types.

- La véracité fait référence à la manière d'améliorer la qualité des données dans les applications big data.

[1]Worldwide IDC Global DataSphere Forecast, 2022-2026: Enterprise Organizations Driving Most of the Data Growth [Online]. Disponible: https://www.idc.com/getdoc.jsp?containerId=US49018922.

Le traitement de larges volumes de données, souvent appelé big data analytics, est l'une des tâches les plus importantes dont ont besoin la plupart des sociétés, qu'il s'agisse d'entreprises bien établies ou de start-ups. À titre d'exemple, les entreprises doivent analyser les logs des activités des clients, faire des recommandations basées sur l'historique de la navigation ou des achats des utilisateurs, et diffuser des publicités à ceux qui pourraient être les plus intéressés par celles-ci.

Les centres de données modernes sont déployés dans le monde entier, de manière géographiquement distribuée, pour traiter de grands volumes de données de manière distribuée à l'aide des frameworks de traitement parallèle des données, tels qu'Apache Hadoop et Spark. Traditionnellement, ces frameworks de traitement parallèle des données sont conçus pour traiter des données au sein d'un même centre de données, où les tâches s'exécutent généralement dans le même cluster, et les données à traiter sont stockées localement dans le système Hadoop Distributed File System (HDFS). Cependant, comme le volume des données augmente, les stocker dans le même centre de données n'est plus envisageable, de ce fait elles doivent naturellement être distribuées dans plusieurs centres de données. Ceci est d'autant plus motivé par le fait que les données à traiter, telles que les logs d'activité des utilisateurs, sont générées de manière géographiquement distribuée. Il est plus efficace de stocker les données là où elles sont générées

Les Framework de traitement des données massives tels que MapReduce [6], Hadoop[2], Spark[3] et Dryad [7] ont été conçus pour analyser efficacement de grandes bases de données. Le problème est que tous ces Frameworks supposent le déploiement dans un seul centre de données, où le réseau est généralement disponible et homogène. La solution la plus simple consiste à regrouper toutes les données sur un seul site, cependant l'analyse de ces données peut être impossible en raison de contraintes de confidentialité et de réglementation, de plus il a été prouvé qu'il s'agissait d'une utilisation coûteuse et inutile de la bande passante du réseau étendu [8, 9].Dans un scénario géo-distribué, les données sont générées à différents endroits dans le monde et stockées dans plusieurs centres de données. Un Framework de traitement de données géo-distribuée doit assigner les traitements là où les données sont localisées et ensuite regrouper les résultats de ces traitements sur un seul site pour exécuter le traitement final.

En bref, un système d'analyse de donnés massives géographiquement distribuées (GDA) doit (1) exécuter de manière transparente les tâches des utilisateurs sur différents sites comme s'il s'agissait d'une tâche locale, (2) prendre en charge les

[2]Apache Hadoop [Online]. Disponible: http://hadoop.apache.org/.
[3]Apache Spark [Online]. Disponible: http://spark.apache.org/.

Framework et langages de traitement de données massives existants, (3) ne permettre le déplacement que pour les données pertinentes pour le résultat final, (4) gérer les pannes et les défaillances des tâches, des nœuds, des racks et des centre de données.

La nécessité d'utiliser une telle infrastructure introduit de nouveaux défis dans le processus d'analyse des données en raison des limitations de la bande passante de la communication entre les centres de données.

Dans cette thèse, nous discutons des défis et passons en revue les derniers frameworks et ordonnanceurs d'analyse de données géo-distribuées tenant compte de la bande passante du réseau étendu. En outre, nous introduisons une nouvelle base de données et un modèle d'apprentissage profond pour améliorer les performances des systèmes d'analyse de données géo-distribuées.

1.2 Questions de recherche

Notre objectif principal dans cette thèse est d'étudier et d'améliorer les performances des frameworks d'analyse de données géo-distribuées. Ainsi, dans cette section, nous relevons les questions liées à cette étude dont les réponses constituent l'essentiel du travail que nous présentons dans ce document:

1. Avant de penser à l'amélioration des performances des frameworks d'analyse de big data géo-distribués, nous devons tout d'abord définir ce qu'est un framework d'analyse de big data géo-distribué ? pourquoi avons-nous besoin de tels systèmes ? quelles sont les principales différences entre les frameworks locaux et géo-distribués ? Quelles sont les principales motivations et les défis de la conception de tels frameworks ? Quelles sont les architectures et les solutions proposées dans la littérature ? A partir de la littérature, quelles sont les questions ouvertes et les domaines de recherche que nous pouvons étudier afin d'améliorer les performances de ces systèmes ?

2. À la suite de notre étude, nous avons identifié l'apprentissage automatique comme une solution pour améliorer les performances des systèmes d'analyse de données géo-distribuées en prédisant le temps d'exécution des travaux. Cela nous a conduit à proposer une nouvelle base de données car nous n'avons pas réussi à trouver de base de données publique disponible. Par conséquent, afin de générer cette base de données, nous avons été amenés à répondre aux questions suivantes :

    - Quelle pile de frameworks allons-nous mettre en place ?

- Comment alimenter le cluster avec des données et exécuter des travaux afin de générer des traces ?
- Quelles sont les caractéristiques des travaux que nous pourrions collecter et la manière de les collecter ?

3. Dans cette thèse, nous nous concentrons sur l'optimisation des performances des travaux MapReduce dans un environnement géo-distribué. L'un des principaux défis de l'optimisation des travaux MapReduce est la planification. La planification repose sur une estimation optimale et précise du temps d'exécution des travaux. Les réseaux de neurones profond (DNN) sont l'un des algorithmes de pointe qui ont eu un impact énorme sur l'apprentissage profond. Par conséquent, nous devons répondre aux questions suivantes : quelles sont les caractéristiques qui ont un impact sur le temps d'exécution des travaux MR ? Est-ce que les DNNs peuvent prédire avec précision le temps d'exécution des travaux MR en utilisant la base de données que nous avons proposée ?

1.3 Contributions

Dans cette thèse, nous présentons la liste des contributions par chapitre/article :

➥ Les contributions du **chapitre 3** (Bergui, M., Najah, S. & Nikolov, N.S, A survey on bandwidth-aware geo-distributed frameworks for big-data analytics, 2021) [10] :

- Dans ce travail, nous avons fourni une vue d'ensemble des frameworks les plus utilisés pour l'analyse des big data et discuté des défis que pose la conception de systèmes efficaces de traitement des données géo-distribuées. Nous avons également étudié et comparé des nouveaux systèmes capables d'exécuter des analyses géo-distribuées, tout en gérant l'hétérogénéité et en tenant compte de la bande passante du réseau étendu. De plus, nous avons identifié les problèmes restants et les directions à suivre pour les développements futurs.

➥ Les contributions du **chapitre 4** (Bergui, M., Nikolov, N.S,. & Najah, S, Hadoop dataset for job estimation in the cloud with limited bandwidth, 2023) [11] :

- Dans ce travail, nous présentons notre nouvelle base de données de traces de tâches MapReduce dans un environnement en cloud avec une bande passante réseau limitée, nous décrivons le processus de génération et de collecte de la base de données en utilisant notre profileur de travaux qui est basé sur les API REST fournies par le framework Hadoop.

➥ Les contributions du **chapitre 5** (Bergui, M., Hourri, S,. Nikolov, N.S,. & Najah, S, Hadoop Job Performance Modeling with Network Limitations: A Deep Learning Approach 2023)[En cours de soumission]:

- Dans ce travail, nous avons proposé un réseau de neurones profonds pour prédire le temps d'exécution des travaux MapReduce en utilisant notre base de donnée présenté dans la contribution précédente.

1.4 Déroulement et structure de la thèse

Le reste de cette thèse est organisé comme suit :

Dans le **chapitre 2**, nous fournissons une vue d'ensemble du modèle de programmation MapReduce et nous donnons un aperçu plus détaillé des composants du framework de calcul Apache Hadoop MapReduce, nous présentons également certains frameworks qui font partie de l'éco-système Hadoop.

Dans le **chapitre 3**, nous passons en revue les frameworks d'analyse de données géo-distribuées qui prennent en compte la bande passante du réseau WAN dans la description du problème, Nous fournissons également une analyse détaillée de chaque framework, avec ses avantages et ses inconvénients et nous les comparons et les classons en fonction de leurs caractéristiques communes. Nous terminons en identifiant les domaines de recherche potentiels et les directions de développement future.

Dans le **chapitre 4**, nous proposons une nouvelle base de données de traces de travaux Hadoop MapReduce version 2 dans un environnement cloud avec une bande passante réseau limitée. Nous présentons la méthodologie et le dispositif expérimental que nous avons adopté pour créer la base de données, puis nous décrivons le processus de génération et de collecte de cette base de données.

Dans le **chapitre 5**, nous présentons un réseau de neurones profonds pour prédire le temps d'exécution des travaux MapReduce, basé sur la base de données présentée dans le chapitre précédent. Nous étudions l'anatomie de MapReduce et les caractéristiques qui impact ses performances à chaque étape, puis nous décrivons le pré-traitement des données, la sélection des caractéristiques et le modèle de deep learning proposé.

Dans le **chapitre 6**, nous présentons une conclusion générale de la thèse ainsi que les limitations, perspectives et les travaux futurs à envisager.

2

Notions essentielles de l'écosystème Hadoop

Sommaire

2.1 Introduction 7
2.2 MapReduce 8
2.3 Apache Hadoop 9
2.3.1 Architecture de Hadoop 10
2.3.2 Hadoop Distributed File System 10
2.3.3 MapReduce 11
2.3.4 Yet Another Resource Negotiator 12
2.4 Apache Hive 14
2.5 Apache Spark 16
2.5.1 Resilient Distributed Dataset 16
2.5.2 Spark Streaming 16
2.6 Apache Flink 17
2.7 Conclusion 17

2.1 Introduction

Apache Hadoop est un framework open source utilisé pour stocker et traiter efficacement de grands volumes de données allant du gigaoctet au pétaoctet. Au lieu d'utiliser un seul gros ordinateur pour stocker et traiter les données, Hadoop permet de regrouper plusieurs ordinateurs pour analyser plus rapidement des datasets massifs en parallèle . Ce chapitre fournit d'abord une vue d'ensemble du

modèle de programmation MapReduce et de ses systèmes d'implémentation. Il présente ensuite l'architecture du framework Apache Hadoop et donne un aperçu détaillé de ses composants tel que Hadoop Distributed File System et Yet Another Resource Negotiator. Ce chapitre décrit également certains systèmes inclus dans l'éco-système Hadoop.

2.2 MapReduce

MapReduce est un modèle de programmation introduit par Google en 2004 pour le traitement parallèle de grands bases de données sur un groupe de machines de manière évolutive et sans défaillance [6]. Dans le modèle de programmation MapReduce, le traitement est exprimé sous la forme d'une fonction map et d'une fonction reduce. La fonction Map traite un bloc de données sous la forme d'une paire (clé, valeur) et produit une sortie sous la forme d'une liste de paire (clé, valeur). Les valeurs intermédiaires sont regroupées en fonction de la même clé, puis transmises à la fonction reduce. La fonction de réduction prend la clé intermédiaire avec ses valeurs associées et les traite pour produire une nouvelle liste de valeurs comme sortie finale. La fonction map et la fonction reduce sont exécutées indépendamment en utilisant les ressources allouées, ce qui permet un parallélisme élevé.

Le diagramme logique de flux de données du modèle de programmation MapReduce est illustré à la figure 2.1, qui présente un ensemble de données météorologiques avec diverses températures de l'air pour différentes années. Le programme doit trouver la température de l'air la plus élevée pour chaque année. L'année est représentée comme la clé et la température de l'air est représentée comme la valeur dans la base de données [12].

Le modèle de programmation MapReduce a été initialement conçu pour le traitement de données sur le Web, mais il est aujourd'hui adapté à d'autres domaines d'applications à forte intensité de données, comme le Machine Learning, la simulation scientifique, l'analyse des données de santé et l'exploration de données sur le Web. Les caractéristiques intéressantes du modèle de programmation MapReduce sont

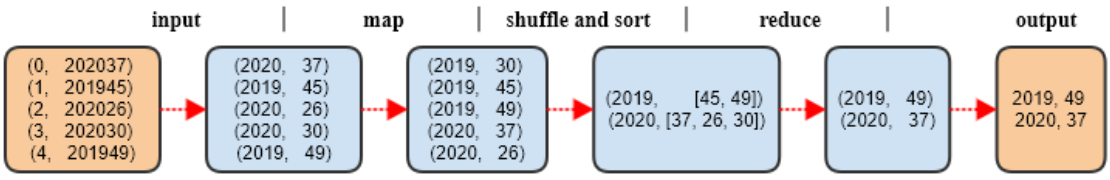


Figure 2.1: Flux de données logique du modèle de programmation MapReduce.

la tolérance aux pannes, la simplicité et la scalabilité. MapReduce est un modèle de calcul hautement évolutif qui permet d'utiliser des centaines d'ordinateurs de commodité peu coûteux comme plate-forme de calcul efficace pour le traitement distribué et parallèle. Ce modèle permet de détecter et de traiter automatiquement les défaillances de nœuds sans que cela n'affecte le processus de traitement. Il s'agit d'un modèle de programmation simple car il permet aux développeurs d'applications de fournir uniquement une implémentation séquentielle de la logique d'application exprimée dans un style fonctionnel (c'est-à-dire la fonction map et la fonction reduce) et le système d'exécution s'occupe des détails de parallélisme de bas niveau, c'est-à-dire le partitionnement de la base de données d'entrée, l'ordonnancement de l'exécution du programme sur plusieurs nœuds d'un cluster d'ordinateurs, le traitement des défaillances de nœuds et la gestion des communications entre nœuds. Par conséquent, le modèle de programmation MapReduce diminue les difficultés de la programmation parallèle, de sorte que les programmeurs peuvent facilement réaliser des tâches complexes.

Le modèle de programmation MapReduce a fait l'objet d'un certain nombre d'implémentations telles que Hadoop [12], Dryad [7], Phoenix [13], Mars [14] et Sector/Sphere [15].Parmi les systèmes d'implémentation du modèle de programmation MapReduce mentionnés, le framework Hadoop est la plateforme MapReduce la plus utilisée en raison de sa nature open source. Les détails du framework Hadoop sont présentés dans la section suivante.

2.3 Apache Hadoop

Hadoop [12] est une implémentation open source du modèle de programmation MapReduce et est devenu la principale plate-forme de calcul pour l'analyse des données volumineuses. Apache Hadoop est un framework développé en Java qui distribue et parallélise le traitement de données massives sur un cluster d'ordinateurs en utilisant le modèle de programmation MapReduce. Il est devenu le framework le plus populaire pour l'analyse de données massives et est utilisé par de nombreuses organisations[1] telles que Yahoo, Facebook, YouTube, Twitter, Google, LinkedIn pour traiter et analyser leurs énormes quantités de données. Aujourd'hui, les fournisseurs de services d'analyse de données volumineuses les plus populaires, tels que IBM, Oracle, Microsoft, Dell, Cloudera et Hortonworks, soit proposent des

[1]Powered by Apache Hadoop [Online]. Disponible: https://cwiki.apache.org/confluence/display/HADOOP2/poweredby.

produits liés à Hadoop tels que IBM BigInsights[2] ou fournissent une assistance aux utilisateurs sur Hadoop MapReduce (Cloudera[3], HortonWorks[4]).

Le framework Hadoop MapReduce est hautement évolutif et peut passer d'une seule machine à des dizaines de milliers de machines, chacune d'entre elles offrant un traitement et un stockage de données locaux. La taille du cluster Hadoop peut diminuer ou augmenter de façon dynamique en fonction de la charge de travail. Le framework Hadoop MapReduce a été développé en partant de l'hypothèse fondamentale que les pannes de machine sont courantes dans les clusters et qu'elles doivent être traitées automatiquement au niveau logiciel par le framework. Par conséquent, des techniques de tolérance aux pannes et de gestion automatique des pannes de machine sont incluses dans le framework.

2.3.1 Architecture de Hadoop

Hadoop suit l'architecture maître-esclave pour stocker et traiter efficacement de grandes quantités de données. Les nœuds maîtres assignent des tâches aux nœuds esclaves. Les nœuds esclaves sont responsables du stockage des données et de l'exécution des calculs/traitements. Les nœuds maîtres sont chargés de stocker les métadonnées et de gérer les ressources dans le cluster. L'architecture Hadoop comprend trois couches, La *couche de stockage* (HDFS), la *couche de gestion des ressources* (YARN), la *couche de traitement* (MapReduce).

2.3.2 Hadoop Distributed File System

HDFS (Hadoop Distributed File System) est le système de fichiers distribués d'Hadoop, qui fonctionne sur du matériel de commodité peu coûteux [16]. Il s'agit de la couche de stockage. Quand une base de données est copiée dans HDFS, ce dernier divise la base de données en blocs de taille égale, prépare plusieurs répliques de chaque bloc et les distribue dans un cluster d'esclaves (Data Node) comme des unités indépendantes. Les répliques du bloc sont stockées sur différents nœuds et racks. Par défaut, la taille des blocs est de 128 MB et le nombre de répliques est de 3, mais peuvent être configurée en fonction des besoins. Les répliques multiples assurent la tolérance aux pannes et la disponibilité des données. Par

[2]IBM BigInsights for Apache Hadoop [Online]. Disponible: https://www.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/8/877/ENUSZP15-0178/index.html&lang=en&request_locale=en.

[3]Cloudera Data Platform (CDP) - Cloudera [Online]. Disponible: https://www.cloudera.com/products/cloudera-data-platform.html.

[4]Hortonworks Data Platform (HDP) - Cloudera [Online]. Disponible: https://www.cloudera.com/products/hdp.html.

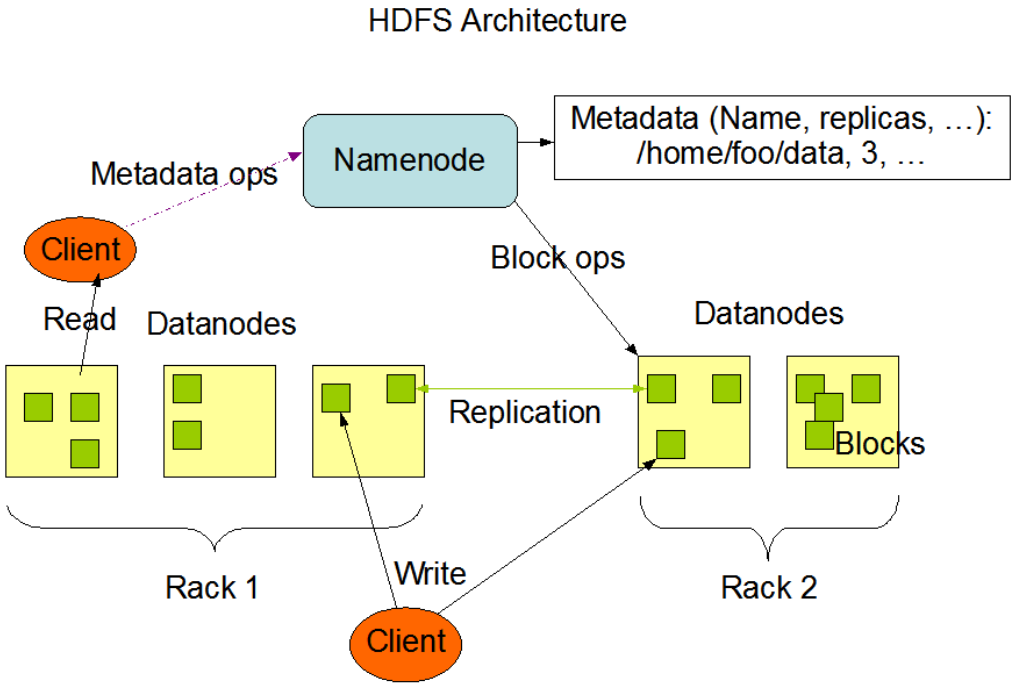


Figure 2.2: Architecture de HDFS.

exemple, si un nœud tombe en panne, les blocs de données qui y sont stockés seront disponibles sur d'autres nœuds.

Tout comme Hadoop, HDFS suit également l'architecture maître-esclave. Il comprend deux démons : NameNode et DataNode comme le montre la Figure 2.2. Le NameNode est le démon maître qui fonctionne sur le nœud maître. Les DataNodes sont les démons esclaves qui fonctionnent sur les nœuds esclaves[1].

- **NameNode**: stocke les métadonnées du système de fichiers, c'est-à-dire les noms des fichiers, les informations sur les blocs d'un fichier, l'emplacement des blocs, les permissions, etc. Il gère les Datanodes.
- **DataNode**: sont les nœuds esclaves qui stockent les données sous forme de blocs. Il répond aux demandes de lecture/écriture du client en fonction des instructions du NameNode.

2.3.3 MapReduce

MapReduce est un moteur de traitement faisant partie du framework Hadoop MapReduce. Un travail Hadoop MapReduce est une unité de traitement qui s'exécute sur plusieurs nœuds de calcul et qui se déroule en plusieurs phases, à

[1]https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

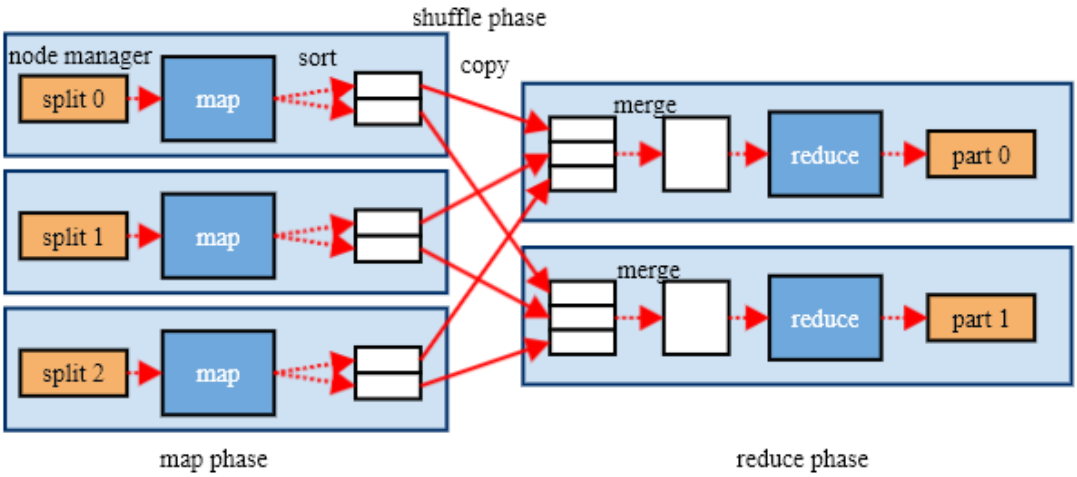


Figure 2.3: Exécution de l'algorithme MapReduce.

savoir la phase map et la phase reduce. Les détails de ces phases sont disponibles dans la section 5.3.

Dans la phase *map*, chaque tâche map traite un bloc de données d'entrée qui est généralement stocké dans HDFS. Les données d'entrée sont généralement divisées en blocs de taille prédéfinie (par exemple 64 MB ou 128 MB) et distribuées sur les nœuds du cluster. Les tâches map lisent les blocs de données et appliquent la fonction map définie par l'utilisateur. La sortie d'une tâche map (données intermédiaires) produite par la fonction map est collectée dans la mémoire physique. Elle est écrite périodiquement de la mémoire physique vers le disque local d'un nœud de traitement qui exécute la tâche map. Les emplacements de la sortie map sont transmis au nœud maître qui est responsable de la transmission à un nœud qui exécute la tâche reduce.

Dans la phase *reduce*, les tâches reduce lisent la sortie map des emplacements distants, les trient selon la clé de sortie map, de sorte que toutes les valeurs de la même clé soient regroupées. Ensuite, les tâches reduce itèrent à travers les fichiers de sortie map triés et transmettent une clé intermédiaire unique et les valeurs associées à la fonction reduce. La sortie de la fonction reduce est écrite dans HDFS comme sortie finale. Le nombre de fichiers de sortie finaux dépend du nombre de tâches reduce lancées. Le flux d'exécution des tâches Hadoop est illustré à la figure 2.3.

2.3.4 Yet Another Resource Negotiator

YARN (Yet Another Resource Negotiator) est le composant de gestion de cluster de la pile Hadoop. YARN est responsable de la coordination et de la gestion des ressources et de l'ordonnancement des travaux dans Hadoop [17]. YARN permet

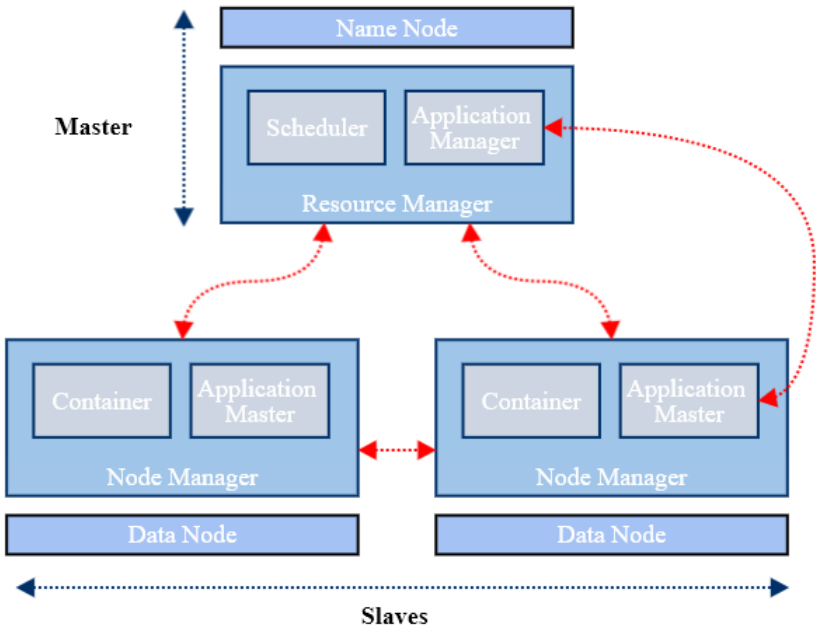


Figure 2.4: Architecture générale de YARN.

l'utilisation de différents moteurs de traitement de données tels que Hive et Spark sur des données stockées dans HDFS, le tout sur le même cluster Hadoop.

YARN est conçu dans l'idée de séparer les fonctionnalités de planification des tâches et de gestion des ressources dans des démons distincts. L'idée de base est d'avoir un ResourceManager global et un ApplicationMaster par application, l'application pouvant être un travail unique ou un DAG (Directed Acyclic Graph) de travaux. YARN se compose de ResourceManager, NodeManager, et ApplicationMaster par application comme illustré dans la figure 2.4.

- ResourceManager: gère les ressources entre toutes les applications du cluster et a deux composants principaux qui sont Scheduler et ApplicationManager.
  - Scheduler: alloue les ressources aux différentes applications fonctionnant dans le cluster, en tenant compte des capacités, des files d'attente, etc. Il s'agit d'un simple ordonnanceur. Il ne surveille ni ne suit l'état de l'application. L'ordonnanceur ne garantit pas le redémarrage des tâches qui ont échoué en raison d'une défaillance de l'application ou du matériel. Il effectue l'ordonnancement en fonction des besoins en ressources des applications.
  - ApplicationManager: responsable de l'acceptation des soumissions de tâches. Il négocie le premier conteneur pour l'exécution de l'Application Master spécifique à l'application. Il fournit un service pour redémarrer

le conteneur ApplicationMaster en cas d'échec. L'ApplicationMaster est responsable de la négociation des conteneurs auprès du Scheduler. Il suit et surveille leur état et leur progression.

- NodeManager: fonctionne sur les nœuds esclaves. Il est responsable des conteneurs, surveille l'utilisation des ressources de la machine, c'est-à-dire l'utilisation du CPU, de la mémoire, du disque, du réseau, et en rend compte au ResourceManager ou au Scheduler.

- ApplicationMaster: responsable de la négociation des ressources auprès du ResourceManager. Il travaille avec le(s) NodeManager(s) pour exécuter et surveiller les tâches.

2.4 Apache Hive

Apache Hive est un entrepôt de données open-source pour Hadoop. Un entrepôt de données fonctionne comme un dépôt central où les informations sont collectées à partir d'une ou plusieurs sources de données [18]. Il collecte des données provenant de sources diverses et hétérogènes dans le but principal de supporter l'analyse et l'interrogation avec un langage proche du SQL.

Apache Hive traduit les programmes écrits en HiveQL en un ou plusieurs travaux qui seront ensuite exécutés sur des moteurs d'exécution tels que MapReduce, Apache Tez ou Apache Spark [19]. Par la suite, Apache Hive organise les données dans un tableau pour le fichier HDFS afin d'exécuter les travaux sur un cluster pour produire le résultat. Une vue d'ensemble de la configuration de Hive/Hadoop est présentée dans la Figure 2.5.

Les principaux composants de Hive sont les suivants:

- Meta Store: Les métadonnées sont constituées de différentes données sur les tables comme leur emplacement, le schéma, les informations sur les partitions, ce qui aide à surveiller la progression des données distribuées dans le cluster.

- Driver: Lors de l'exécution de l'instruction en langage de requête Hive, le driver reçoit l'instruction et la contrôle pendant tout le cycle d'exécution. Parallèlement à l'exécution de l'instruction, le driver stocke également les métadonnées générées par l'exécution. Il crée également des sessions pour surveiller la progression et le cycle de vie des différentes exécutions. Après la fin de l'opération de réduction par le job Map Reduce, le driver collecte toutes les données et les résultats de la requête.

- Compiler: Il est utilisé pour traduire le langage de requête Hive en entrée MapReduce. Il invoque une méthode qui exécute les étapes et les tâches nécessaires pour lire la sortie HiveQL selon les besoins du MapReduce.

- Optimiser: La tâche principale de l'optimiseur est d'améliorer l'efficacité et l'évolutivité, en créant une tâche tout en transformant les données avant l'opération de réduction. Il effectue également des transformations comme l'agrégation, la conversion de pipeline par une jointure unique pour des jointures multiples.

- Executor: La tâche principale de l'exécuteur est d'interagir avec le gestionnaire de tâches Hadoop pour la planification des tâches prêtes à être exécutées.

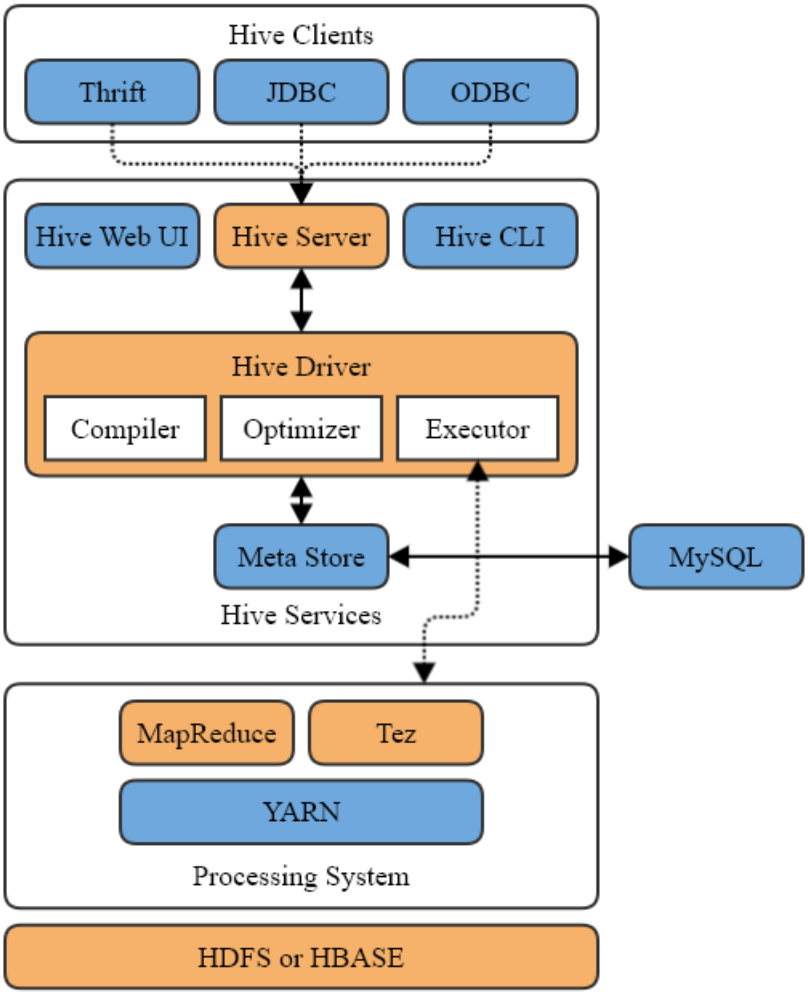


Figure 2.5: Principaux composants de l'architecture Hive.

- Hive Client: Hive offre un support pour les applications écrites dans n'importe quel langage de programmation comme C++, Python, Java, etc. en utilisant les pilotes JDBC, ODBC et Thrift, pour exécuter des requêtes sur Hive.

2.5 Apache Spark

Apache Spark[5] est une plateforme de traitement en cluster basée sur Hadoop MapReduce et qui étend le modèle pour prendre en charge d'avantage de types de traitements de données comme les requêtes interactives. Contrairement à Hadoop MapReduce qui doit stocker les résultats de chaque tâche sur le disque, Spark stocke les résultats en mémoire, ce qui augmente la vitesse de traitement des données.

2.5.1 Resilient Distributed Dataset

Resilient Distributed Dataset (RDD) est une abstraction distribuée en mémoire permettant aux développeurs de réaliser des traitements parallèles en mémoire sur un cluster de manière totalement tolérante aux pannes. RDD a été créé pour répondre au problème que posent les algorithmes itératifs et les traitements interactifs à MapReduce [20].

Du point de vue formel, le RDD est une "collection" (au sens Scala du terme) d'éléments partitionnés et distribués sur les nœuds du cluster et uniquement accessible en mode lecture seule.

Il fonctionne comme une abstraction de partage de données dans un cluster. Lorsque l'on mentionne "distribué" dans la définition du RDD, on parle en fait de "partagé", atomique, car il utilise la mémoire cache pour faire persister les données en RAM en vue de leur réutilisation, ce qui évite la réplication des données sur disque, nécessaire dans Hadoop pour assurer la disponibilité du cluster. C'est grâce à ce mécanisme qu'il est capable de fournir une haute disponibilité et une tolérance aux pannes au cluster.

2.5.2 Spark Streaming

Apache Spark Streaming est un système de traitement de flux de données extensible et tolérant aux pannes qui prend en charge de manière native les charges de travail par lots et en flux. Spark Streaming est une extension de l'API Spark de base qui permet le traitement en temps réel de données provenant de diverses sources, notamment (mais pas exclusivement) Kafka, Flume et Amazon Kinesis. Ces données

[5]Apache Spark [Online]. Disponible: http://spark.apache.org/.

traitées peuvent être transmises à des systèmes de fichiers, des bases de données et des tableaux de bord en temps réel.

Son abstraction principale est un flux discrétisé (Discretized Stream) ou, en bref, un DStream, qui représente un flux de données divisé en petits lots [21]. Les DStreams sont construits sur les RDDs, l'abstraction de données de base de Spark. Cela permet à Spark Streaming de s'intégrer de manière transparente à tous les autres composants de Spark, comme MLlib et Spark SQL.

2.6 Apache Flink

Flink est un Framework open source qui traite les données de flux comme de véritables flux (flux natifs), les enregistrements sont instantanément acheminés à travers les opérateurs dès qu'ils arrivent [22]. Flink fournit une gestion des pannes, un haut débit et un mode de compatibilité qui permet l'utilisation de code Apache Storm[6] et MapReduce existant et non modifié sur le moteur Flink.

2.7 Conclusion

Dans ce chapitre nous avons donné un aperçu général du modèle de programmation MapReduce. Nous avons présenté ensuite l'architecture du Framework Apache Hadoop et donné un aperçu détaillé des principaux composants qui le constitue. Ce chapitre décrit également certains des systèmes inclus dans l'écosystème Hadoop.

Aussi, nous nous sommes principalement concentrés sur Apache Hadoop et ses composants, car c'est sur eux que nous avons basé notre travail et nos contributions.

[6] Apache Storm [Online]. Disponible: http://storm.apache.org/.

3

Les frameworks big data géo-distribués tenant compte de la bande passante

Sommaire

3.1 Introduction 19
3.2 Méthodologie 21
3.3 Motivations 22
3.3.1 Support des Applications Géo-distribuées 22
3.3.2 Défaillance ou Panne du Centre de Données 23
3.3.3 Contraintes réglementaires 23
3.4 Défis 23
3.4.1 Contraintes liées au réseau étendu WAN 24
3.4.2 Hétérogénéité des Clusters 24
3.4.3 Tailles de données hétérogènes 24
3.5 Traitement Big Data Géo-distribué 25
3.5.1 Frameworks de traitement par lots géo-distribués 26
3.5.1.1 Frameworks basés sur MapReduce 26
3.5.1.2 Frameworks basés sur Spark 28
3.5.1.3 Frameworks de traitement de type SQL 35
3.5.1.4 Autres frameworks de traitement par lots . . . 38
3.5.2 Frameworks de traitement de flux géo-distribués 40
3.5.2.1 Frameworks de traitement par micro-lots . . . 40
3.5.2.2 Frameworks de traitement de flux natifs 41
3.6 Conclusion 45

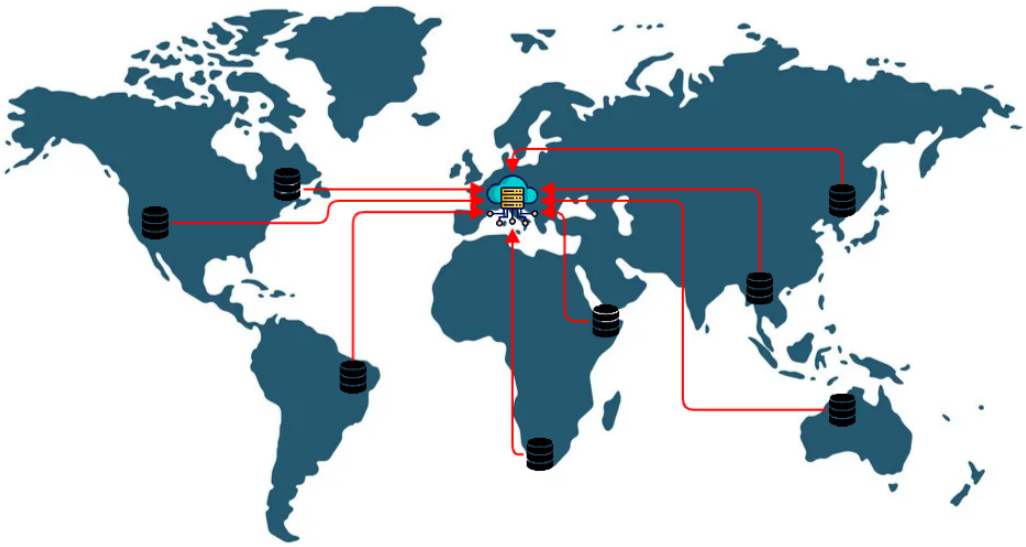
3.1 Introduction

Traditionnellement, l'analyse des données est effectuée de manière centralisée au sein d'un cluster ou d'un centre de données, comprenant un grand nombre de machines de calcul connectées par un réseau à haut débit. Plusieurs frameworks d'analyse de données et de calcul ainsi que des prototypes de recherche ont été développés dans cet environnement pour le traitement par lots [6, 18, 23] et le traitement par flux [21, 22, 24, 25]. Cependant, l'utilisation de ces frameworks de calcul en cluster pour analyser de façon collective des données géo-distribuées n'est pas triviale.

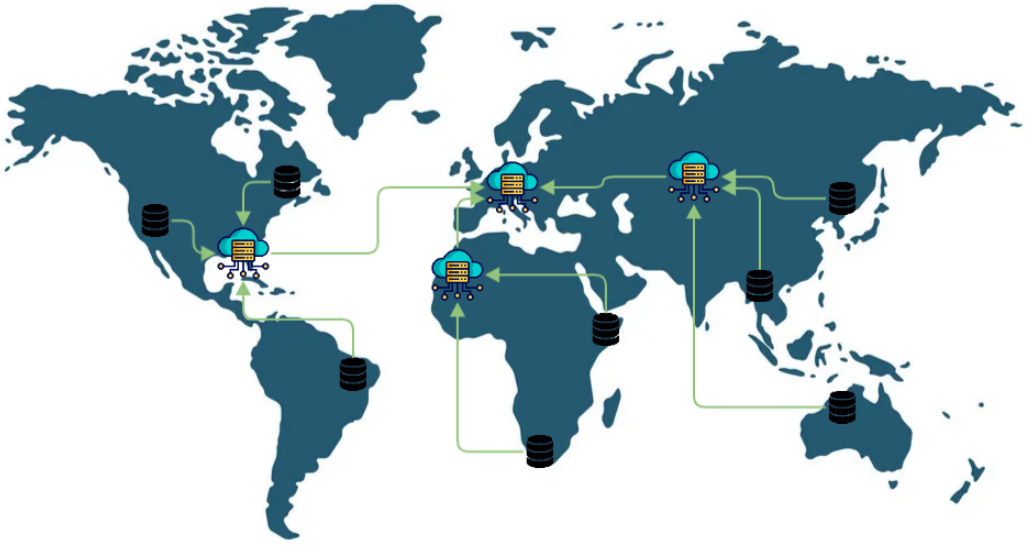
Par exemple, la plupart des frameworks d'analyse de données supposent que toutes les données d'entrée et les machines de calcul sont situées sur le même site et que toute communication de données entre les machines se fait sur un réseau local (LAN) à haut débit. Cependant, une telle hypothèse n'est pas valable pour le traitement de données géo-distribuées car les données ne sont pas forcément situées au même endroit que le cluster de calcul et la transmission de données géo-distribuées à un cluster de calcul centralisé se fait sur un réseau étendu (WAN) à bande passante relativement faible à haute latence et hétérogène.

Une approche possible pour analyser des données géo-distribuées est d'abord de transmettre toutes les données à un seul cluster, puis de les traiter toutes ensemble à l'aide d'un framework d'analyse de données en cluster existant. Nous appelons cette approche un modèle de traitement centralisé (figure 3.1b). Malheureusement, la transmission d'une grande quantité de données sur le réseau étendu (WAN) entraîne généralement un délai de transmission très élevé en raison de la limitation de la bande passante du WAN. Les études ont montré que la bande passante du réseau étendu d'aujourd'hui est nettement inférieure à celle du réseau local de plusieurs ordres de grandeur [26, 27]. En outre, l'internet public qui est souvent utilisé par les clouds privés et les infrastructures périphériques a une bande passante encore plus limitée. Ainsi, le modèle de traitement centralisé limite généralement la rapidité des résultats et n'est donc pas pratique pour de nombreuses applications d'analyse de données [8, 9, 26, 28, 29].

Les efforts récents ont proposé de nouveaux frameworks et techniques de planification basés sur Hadoop, MapReduce et Spark qui permettent l'analyse de données sur plusieurs centres de données (Figure 3.1b) [8, 30–35]. Cependant, ces frameworks ne sont pas optimisés pour l'hétérogénéité et les limitations de la bande passante inter-sites [36, 37]. De plus, la plupart des travaux supposent que les sites ont des capacités de calcul homogènes et disponibles, ce qui n'est pas en conformité avec la réalité [38].



(a) Approche centralisée.



(b) Approche décentralisée (géo-distribuée).

Figure 3.1: Modèle de traitement centralisé vs. décentralisé des données géo-distribuées.

Dans ce chapitre, nous passons en revue les frameworks de traitement des données géo-distribuées qui tiennent compte de la bande passante du réseau étendu puisque le coût et la performance d'un travail géo-distribuée dépendent de la bande passante du réseau WAN et de la quantité des données échanges entre les sites [39], nous présentons les avantages et les inconvénients de la plupart d'entre eux. Nous les classons également en fonction de la technique de traitement (batching, micro-batching, streaming natif) et du framework de big data sur lequel ils sont basés (MapReduce, Spark, Flink) et les comparons en fonction de plusieurs caractéristiques telles que la localisation des données, le support multi-clusters et le type d'architecture. Comme résultat, nous avons pu identifier les problèmes restants pour des travaux futurs potentiels, comme le besoin de mécanismes de confidentialité, de sécurité et d'authentification qui manquent dans tous les frameworks, une architecture décentralisée qui offre la flexibilité et la tolérance aux pannes dans les systèmes GDA et l'utilisation de l'apprentissage automatique pour améliorer l'ordonnancement et le placement des tâches.

Dans la section Méthodologie, nous décrivons la méthodologie que nous avons adoptée dans notre étude. Dans la section Motivations, nous fournisons les raisons et les motivations derrière la conception de frameworks big data géo-distribués. Dans la section Défis, nous décrivons les défis auxquels sont confrontés les systèmes de traitement de données géo-distribuées. Dans la section Traitement Big Data Géo-distribué, nous passons en revue, classons et comparons les différents frameworks GDA tenant compte de la bande passante. Enfin, la section Conclusion conclut le chapitre.

3.2 Méthodologie

Dans cette étude, nous prenons en compte que les articles répondant aux critères suivants :

- Les articles doivent inclure l'hétérogénéité de la bande passante inter-DC tout en améliorant la performance des applications, le temps d'exécution des requêtes ou en minimisant les transferts de données inter-DC.
- Les articles qui se concentrent uniquement sur les mécanismes d'ordonnancement ou d'allocation des ressources géo-distribuées sont exclus, par exemple Flutter [35], WANanalytics [8], Pixida [40], Awan [41].
- Les articles qui se concentrent uniquement sur un type particulier de données sont exclus, par exemple HPS+ [42].

- Les articles qui proposent un système GDA basé sur un service disponible uniquement auprès d'un fournisseur de cloud computing spécifique sont exclus, par exemple Yugong [43].
- Les frameworks qui distribuent les données sur différents DC avant le calcul sont exclus, par exemple Resilin [44], Photon [45]. Les données doivent déjà être distribuées avant le calcul.
- Les articles sur l'apprentissage automatique géo-distribué sont exclus, par exemple Gaia [26].
- Les articles publiés avant 2014 sont exclus.

Parmi les études similaires précédentes dans ce domaine, on peut citer *Dolev et al.* [46] et *Ji et al.* [47]. Dans cette étude, nous étendons le champ d'application des études précédentes en tenant compte des avancées récentes dans le domaine du traitement géo-distribué des données et nous nous concentrons en particulier sur les frameworks big data géo-distribués tenant compte de la bande passante des réseaux étendus. Cela reflète le point de vue spécifique que nous adoptons sur les motivations (voir section Motivations) et les défis (voir section Défis) du traitement géo-distribué des données ainsi que sur les caractéristiques de catégorisation des systèmes géo-distribués (voir section Traitement Big Data Géo-distribué).

3.3 Motivations

Compte tenu de la taille des centres de données existants, certains estiment que l'analyse des données géo-distribuées est inutile. Dans cette section, nous énumérons les raisons qui motivent la conception de frameworks de big data géo-distribués.

3.3.1 Support des Applications Géo-distribuées

Les organisations opérant dans le monde entier déploient des applications dans des centres de données géo-distribuées pour répondre aux besoins des clients et aux exigences de latence. Ainsi, d'énormes volumes de données sont générés à ces endroits géo-distribués et la collecte de toutes les données à un seul endroit pour le traitement a été prouvée comme étant coûteuse et un gaspillage en termes de ressources [8, 9]. D'où la nécessité de frameworks de big data capables d'affecter les traitements là où se trouvent les données, puis de regrouper uniquement les sorties pertinentes de ces traitements sur un seul site pour exécuter le traitement final.

3.3.2 Défaillance ou Panne du Centre de Données

La défaillance d'un centre de données est un événement inhabituel ; cependant, lorsqu'elle se produit, elle peut entraîner une interruption de service pour les organisations qui hébergent des applications dans ce centre. La plupart des frameworks de big data gèrent déjà la tolérance aux pannes de disque, de nœud ou de rack en répliquant les données entre les nœuds, mais la réplication des données ne s'étend pas en dehors du centre de données. Il est donc nécessaire de concevoir des systèmes qui introduisent de nouvelles architectures et de nouvelles techniques de réplication des tâches, des travaux et des données (voir section 3.5) afin de gérer les défaillances des centres de données.

3.3.3 Contraintes réglementaires

Récemment, les gouvernements ont augmenté les restrictions sur le transfert et le stockage des données[1] [28], ce qui rend la solution consistant à regrouper de manière centralisée toutes les données à un seul emplacement avant le traitement inapplicable. Le transfert de données brutes sensibles ou confidentielles (données de santé) à l'intérieur du pays est encore acceptable ; en revanche, le transfert de données brutes sensibles à l'extérieur du pays peut constituer une atteinte à la vie privée. Par conséquent, il est obligatoire de concevoir des systèmes de traitement géo-distribués qui préservent la confidentialité des données en évitant le transfert de données brutes et en transférant uniquement les données intermédiaires nécessaires.

3.4 Défis

Regrouper toutes les données en un seul endroit pour les analyser s'est avéré coûteux et limitant le temps d'analyse [8, 9]. La meilleure approche consiste à laisser les données "sur place" et à répartir les tâches d'un travail entre les différents clusters. Cependant, cette approche est confrontée à de nombreux défis, comme nous l'expliquons dans la suite de cette section.

[1] European Commission press release. Commission to pursue role as honest broker in future global negotiations on internet governance [Online]. Disponible: https://ec.europa.eu/commission/presscorner/detail/en/IP_14_142.

3.4.1 Contraintes liées au réseau étendu WAN

L'hétérogénéité des ressources du réseau est une caractéristique importante dans l'analyse des données géo-distribuées. La bande passante du réseau étendu est très limitée par rapport à celle du réseau local (LAN). Par exemple, *Zhang et al.* ont constaté que la bande passante intra-centre est d'environ 820 Mbps, tandis que la bande passante inter-centre est d'environ 100 Mbps [27].

En outre, la bande passante disponible peut varier considérablement d'un centre de données à l'autre en raison de différences dans le matériel réseau et/ou du trafic des autres applications non analytiques exécutées dans le même cluster. Par exemple, *Viswanathan et al.* constatent la variation entre des paires de dix régions Amazon EC2, ainsi qu'entre des centres de données gérés par Microsoft [29]. La fluctuation de la bande passante entre les centres peut entraîner un temps de transmission des données imprévisible, ce qui a un impact significatif sur les performances des traitements géo-distribués [48, 49]. Ainsi, la bande passante du réseau étendu est une contrainte importante dans l'analyse des données géo-distribuées.

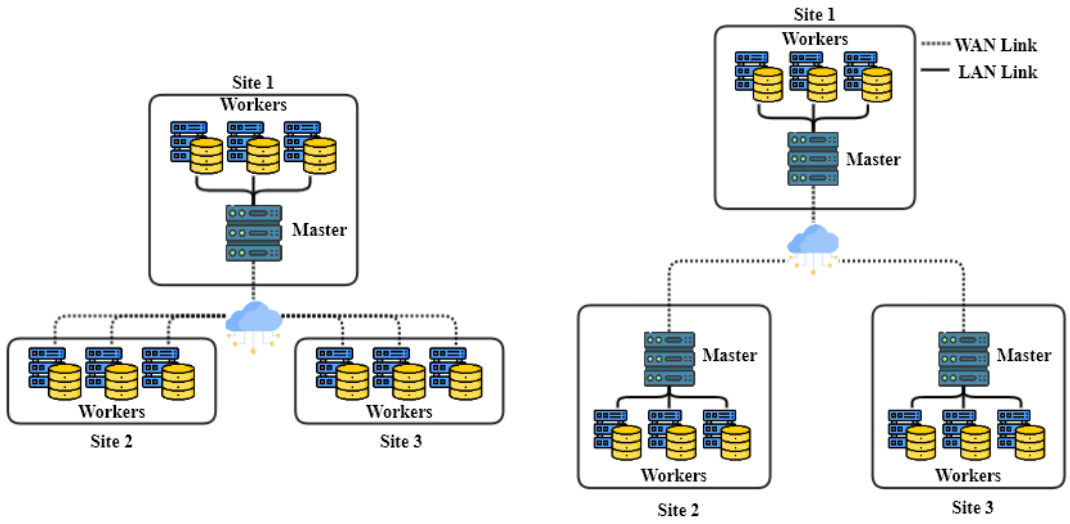
3.4.2 Hétérogénéité des Clusters

Comme nous l'avons mentionné dans la section 3.1, la plupart des travaux récents supposent que les centres ont des capacités de traitement homogènes et disponibles, ce qui n'est pas en accord avec la réalité [38]. Les clusters sont bâtis avec différents taux d'investissement et différentes exigences de capacité à différents moments. *Hung et al.* constatent que les ressources informatiques varient jusqu'à deux fois entre des centaines de sites [38]. Les clusters partagent souvent les ressources avec des applications non analytiques (les services des clients), ce qui limite les ressources de traitement disponibles pour les travaux d'analyse de données et accentue l'hétérogénéité [9, 50].

De plus, il a été prouvé que la mémoire peut devenir un point de congestion lors de l'exécution de requêtes d'analyse de données à grande échelle [51].

3.4.3 Tailles de données hétérogènes

Dans un réseau de capteurs globalement distribué, la taille des données générées dépend de la fréquence et du nombre de capteurs. Au fil du temps, la distribution des données varie considérablement et n'est pas constante. De plus, pour un travail géo-distribué, les données disponibles sur les différents sites pour effectuer les analyses peut ne pas être l'ensemble des données distribuées. Il est donc problématique d'équilibrer et de fournir aux sites des ressources informatiques proportionnelles à la taille des données générées.



(a) Approche centralisée. **(b)** Approche décentralisée.

Figure 3.2: Architecture centralisée vs. décentralisée des systèmes géo-distribués.

3.5 Traitement Big Data Géo-distribué

Dans cette section, nous examinons les frameworks big data géo-distribués qui prennent en compte la bande passante dans le scénario où les données sont déjà distribuées sur plusieurs sites avant le traitement.

Nous passons en revue plusieurs frameworks basés sur MapReduce, Spark et Flink, et nous les comparons sur la base de plusieurs caractéristiques telles que la gestion des ressources, la tolérance aux pannes, la localisation des données et la prise en charge de plusieurs clusters (voir tableau 3.1). Ensuite, nous les classons en deux catégories d'architectures, comme suit :

- *Architecture centralisée:* Un maître unique situé sur l'un des sites contrôle les ressources de tous les travailleurs de tous les sites, comme le montre la Fig. 3.2a.

- *Architecture décentralisée:* Chaque site possède un maître qui contrôle ses propres travailleurs. Chaque site peut exécuter des tâches traditionnelles en cluster isolé, mais aussi collaborer et partager des ressources de calcul et de données pour prendre en charge des tâches géo-distribuées, comme le montre la Fig. 3.2b.

3.5.1 Frameworks de traitement par lots géo-distribués

3.5.1.1 Frameworks basés sur MapReduce

Medusa est une plateforme basée sur MapReduce qui permet le traitement géo-distribué sans aucune modification du framework Hadoop et supporte trois scénarios de défaillance : accidentelle, malveillante, et les pannes de cloud [52]. Medusa démarre f+1 copies d'un travail MapReduce dans différents clouds. Il valide le calcul en comparant les résultats des travaux répliqués et traite les défaillances accidentelles en réexécutant le travail défectueux dans le même cloud. Pour les défaillances malveillantes ou les pannes de cloud, le système réexécute le travail défectueux dans un autre cloud. Medusa peut identifier le cloud compromis dans le scénario de défaillance malveillante.

Chrysaor est similaire à Medusa mais réplique les tâches au lieu des travaux [53]. Un proxy compare chaque réplique de la sortie de toutes les tâches map et reduce pour identifier les tâches défectueuses et les réexécute immédiatement, au lieu d'attendre la fin de l'exécution du travail. Chrysaor lance f+1 copies de la tâche défectueuse dans le même cloud lorsqu'il s'agit de défaillances accidentelles. Pour les défaillances malveillantes ou les pannes de cloud, Chrysaor exécute f+1 copies de la tâche défectueuse dans un autre cloud lorsqu'il s'agit d'une défaillance dans une tâche de map. Chrysaor doit réexécuter la totalité du travail si une défaillance malveillante ou une panne du cloud se produit pendant la phase de reduce.

Avantages: **Medusa** et **Chrysaor** planifient les tâches répliquées sur plusieurs clouds [54], le meilleur cloud (pour Medusa) ou les meilleurs clouds (pour Chrysaor) en fonction de la puissance de calcul (nombre de cœurs de CPU, vitesse d'horloge, mémoire totale) et de la bande passante, ce qui réduit le temps d'exécution des travaux.

GeoDis est le premier ordonnanceur tenant compte de la localité et du réseau qui prend en compte à la fois la localité des données (avec la réplication) et la migration des données pour optimiser le temps d'exécution des travaux à forte intensité de données sur des systèmes géo-distribués [55]. L'ordonnanceur global centralisé gère une file d'attente FIFO (First-In First-Out) pour tous les travaux soumis et décide où placer les tâches et à quelle réplique accéder. Dans chaque centre, un ordonnanceur local maintient et rapporte à l'ordonnanceur global la progression de la file d'attente des tâches locales (voir Figure 3.3). Le problème de placement des tâches et d'accès aux données est décrit comme un programme

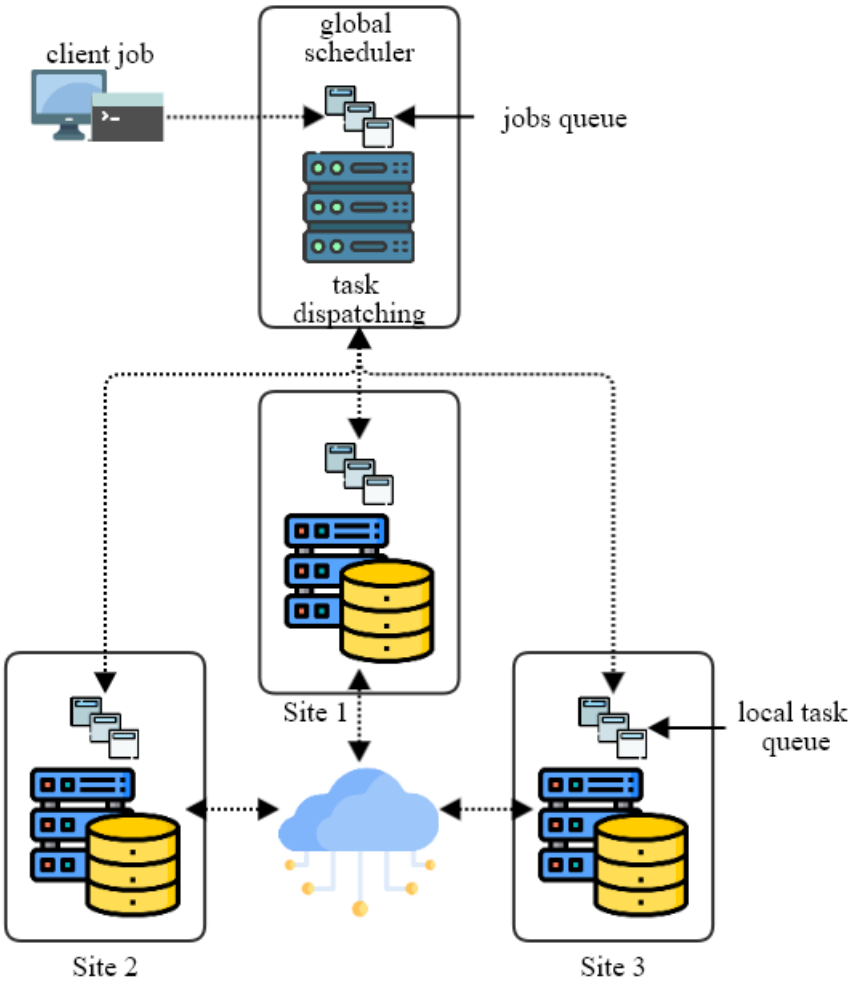


Figure 3.3: L'architecture du système GeoDis.

linéaire (LP) et résolu à l'aide du solveur GLPK[2]. L'algorithme heuristique en ligne proposé favorise la localisation des données à la migration des données lorsque cela est possible, mais dans le cas où les données doivent être transférées à partir d'un centre de données distant, l'algorithme sélectionnera la réplique de données du centre de données avec la liaison la plus rapide.

GeoDis peut réduire le temps d'exécution des travaux de traitement de 44 % par rapport aux algorithmes de la littérature et demeure à 91 % plus proche de la solution optimale du solveur LP [55].

Li et al.(1) [56] ont proposé un framework MapReduce géo-distribué qui minimise le trafic entre les centres de données en considérant conjointement le déplacement

[2]Makhorin Andrew. GLPK (GNU Linear Programming Kit) [Online]. Disponible: https://www.gnu.org/software/glpk/.

des données d'entrée et le placement des tâches. Les données d'entrée peuvent être déplacées par les tâches MapReduce exécutées dans des centres de données distants si le trafic inter-centres total est réduit. Le problème du déplacement des données et du placement des tâches est formulé comme un problème d'optimisation non linéaire et résolu à l'aide d'une technique de linéarisation pour remplacer les contraintes non linéaires par des contraintes linéaires. Une approche d'approximation par la relaxation de l'une des contraintes est utilisée pour obtenir le délai prévu de traitement des tâches. La conception du système comporte trois éléments principaux :

- *Extracteur de paramètres:* Estime les paramètres tels que la bande passante entre les clusters et les informations sur le rapport OI en analysant l'historique d'exécution de travaux similaires, puis envoie les paramètres à l'optimiseur.
- *Optimiseur:* Exécute un algorithme qui détermine le déplacement des données d'entrée et le placement des tâches en fonction des paramètres estimés. L'algorithme minimise le trafic inter-clusters induit par le travail MapReduce.
- *Chargeur de données et assignateur de tâches:* Les données d'entrée sont récupérées par le chargeur de données en fonction des choix de placement des tâches effectués par l'optimiseur. Pour chaque fraction de données chargée, l'assignateur de tâches lance une tâche map et des tâches reduce.

3.5.1.2 Frameworks basés sur Spark

Lube est un framework géo-distribué qui réduit les temps de réponse des requêtes en détectant les congestions au moment de l'exécution [51]. Lube surveille les paramètres de performance (CPU, mémoire, réseau et disque) en temps réel et utilise Autoregressive Integrated Moving Average (ARIMA) [67] ou Sliding Hidden Markov Model (SlidHMM) [68] pour détecter les congestions de ressources au moment de l'exécution. L'algorithme d'ordonnancement tient compte de la localisation des données et de la gravité de la congestion pour affecter les tâches aux nœuds de travail. L'algorithme late-binding de Sparrow [69] est utilisé pour éviter les faux positifs lors de la détection des congestions en retenant une tâche pendant une courte période avant de la soumettre à un nœud de travail.

Lube est le premier framework qui utilise des techniques d'apprentissage automatique pour détecter les congestions pendant l'exécution. Lube est implémenté sur Spark et atteint une précision de 90 % pour la détection des congestions et réduit le temps de réponse des requêtes de 33 % lorsqu'il est déployé sur 9 régions Amazon EC2 [51].

Table 3.1: Comparaison des techniques de traitement des données géo-distribuées

Frameworks	Resource management	Data locality	Multi-query support	Cluster architecture	Multi-cluster support	Relevant intermediate data finding	Fault-tolerance support	Heterogeneous resources	Inter-DC traffic reduction	Machine learning use	Cost-awareness
Batch processing MapReduce-based frameworks											
Medusa [52]				C			✓				
Chrysaor [53]				C			✓				
GeoDis [55]		✓		C				✓			
Li et al.(1) [56]		✓		C					✓		
Batch processing Spark-based frameworks											
LinePro [57]		✓		C							
Houtu [27]	✓		✓	D	✓		✓	✓			
Tetrium [38]	✓		✓	C		✓		✓	✓		
Lube [51]		✓		C				✓		✓	
Harmony [58]		✓		C				✓	✓		
Kimchi [59]				C			✓	✓	✓		✓
Other batch processing frameworks											
HDM-MC [60]				D	✓			✓			
Geo-distributed SQL-style processing systems											
Turbo [61]		✓		C		✓			✓	✓	
Bohr [62]		✓		C		✓			✓		
Micro-batch processing frameworks											
Iridium [9]		✓		C		✓			✓		
Li et al.(2) [63]			✓	C					✓	✓	
Native stream processing frameworks											
Sana [64]			✓	C		✓	✓		✓		
WASP [65]	✓		✓	C			✓	✓	✓		
JetStream [66]		✓		C			✓		✓		

Notations. C: Centralised architecture. D: Decentralised architecture.

Avantages: Les auteurs affirment que Lube est le premier travail qui utilise l'apprentissage automatique pour détecter les congestions au moment de l'exécution.

Désavantages: Les auteurs ne mentionnent aucune base de données pour l'entraînement ou le test du modèle d'apprentissage automatique.

LinePro est un algorithme de traitement des données géo-distribuées qui prend en considération l'état du réseau, basé sur Spark [57]. L'algorithme tire parti de la localité des données pour réduire le coût de transfert en programmant les traitements qui sont prêts à être exécutés plutôt que de programmer tous les traitements en même temps. Le problème du déplacement des traitements est décrit comme un programme linéaire en nombres entiers (ILP) et résolu à l'aide du solveur Gurobi[3]. Les principaux composants du modèle de mouvement de traitement sont :

- *TaskDefinition:* un objet contenant les attributs suivants : stageId, shuffleId, taskbinary, parts, locations et cost matrix. La matrice des coûts est utilisée pour produire les emplacements finaux d'une tâche reduce. Les emplacements contiennent les nœuds vers lesquels le traitement peut être déplacé. shuffleid définit si la tâche appartient à la phase reduce ou non.

- *MapOutTracker:* suit et fournit les emplacements des données et la taille de sortie des tâches map à DAGScheduler.

- *DAGScheduler:* cette fonction soumet les ensembles de tâches manquants qui doivent être exécutés et utilise MapOutTracker et la dépendance RDD pour produire TaskDefinition de chaque tâche.

- *TaskSchedulerImpl:* calcule l'emplacement du traitement. L'emplacement du traitement de map est fourni par la dépendance RDD, tandis que l'emplacement du traitement reduce est fourni par l'ILP via la matrice des coûts et TaskDefinition.

- *TasksetManager:* lorsque chaque nœud est saturé et ne peut pas exécuter un ensemble de tâches en une seule étape, la matrice des coûts est utilisée pour construire une liste de coûts triée pour chaque nœud. Le nœud peut choisir le coût le plus faible dans la liste des tâches lorsque des ressources de traitement sont disponibles.

[3]Gurobi Optimizer - Gurobi [Online]. Disponible: https://www.gurobi.com/products/gurobi-optimizer/.

LinePro améliore les performances du traitement des données géo-distribuées de 22% par rapport à l'ordonnanceur par défaut de Spark lorsqu'il est déployé sur 3 centres de données [57].

Désavantages: LinePro est peu performant si la distribution des bases de données n'est pas équilibrée entre les centres de données.

Tetrium est un système géo-distribué d'allocation de ressources multiples conçu au-dessus de Spark, qui prend en compte la bande passante et les ressources de calcul hétérogènes pour planifier les travaux, ordonner et placer les tâches map et reduce [38]. Tetrium se compose de deux gestionnaires, comme suit.

- *Global manager:* situé sur un seul site pour coordonner les travaux d'analyse entre les sites, ajuste les charges de travail et conserve la trace des emplacements des données.

- *Site manager:* situé sur chaque site pour exécuter les travaux assignés, contrôle les ressources locales et notifie au gestionnaire global les changements de disponibilité des ressources.

Pour planifier des tâches multiples et concurrentes sur un cluster géo-distribué, le problème est simplifié en résolvant d'abord la planification d'un seul travail puis en étendant la solution à des travaux multiples. Un programme linéaire est décrit pour optimiser le temps d'exécution d'un seul travail tout en considérant les slots de traitement disponibles et la bande passante comme des contraintes. Ensuite, le SRPT (Shortest Remaining Processing Time) utilise la solution donnée par le LP de chaque travail pour planifier et organiser les travaux concurrents géo-distribués [38].

Tetrium [38] améliore le temps de réponse moyen des travaux jusqu'à 78% par rapport aux techniques de localisation existantes [70], jusqu'à 55% par rapport à Iridium [9], et 33% par rapport à Tetris [71] lorsqu'il est déployé dans huit régions Amazon EC2 sur cinq continents en exécutant les benchmarks TPC-DS[4] et Berkeley Big Data Benchmark[5].

Avantages: Tetrium est le premier effort d'ordonnancement multi-ressources pour les travaux d'analyse de données sur des clusters géo-distribués qui prennent en compte la distribution des données, les capacités de calcul et l'hétérogénéité de la bande passante du réseau.

[4]TPC-DS Decision Support Benchmark [Online]. Disponible: http://www.tpc.org/tpcds.

[5]Berkeley Big Data Benchmark [Online]. Disponible: https://amplab.cs.berkeley.edu/benchmark/.

Désavantages: Tetrium ne tient pas compte de la congestion du réseau entre les centres de données et ne convient pas aux charges de travail orientées flux.

Houtu est un système décentralisé d'analyse de données géo-distribuées conçu pour fonctionner efficacement sur un ensemble de centres de données [27]. Chaque centre de données peut exécuter des travaux dans un mode traditionnel de cluster unique ainsi que dans un mode géo-distribué en collaborant avec d'autres centres de données. Houtu applique un algorithme adaptive feedback (AF) et parametrized delay scheduling with work stealing (Parades) qui étend l'algorithme delay scheduling [70] dans chaque gestionnaire de travaux (JM) pour respectivement gérer les ressources et planifier les tâches pour le travail géo-distribué. Le cycle de vie d'un travail dans Houtu se compose de 5 étapes, comme illustré dans la Fig. 3.4.

- *Soumission des travaux et génération des gestionnaires de travaux:*
  (Étape 0) Un utilisateur soumet un travail sous la forme d'un graphe acyclique dirigé (DAG) au maître. (Étape 1) Le maître génère des gestionnaires de travaux en fonction de la description du travail. (Étape 2) Un JM primaire est créé directement dans son propre cluster. (Étape 2a) La description du travail est envoyée aux maîtres distants si des ressources distantes sont nécessaires. (Étape 2b) Des JMs semi-actifs sont créés dans les clusters distants.

- *Demande de ressources et exécution des tâches:*
  (Étape 3) Les gestionnaires de travaux envoient des demandes de ressources de traitement (exécuteurs de tâches/conteneurs) à leurs maîtres locaux. (Étape 4) Les maîtres (planificateurs de travaux) planifient les ressources pour les JMs en renvoyant les conteneurs pour les ressources demandées. (Étape 5) Les JMs soumettent les tâches qui s'exécutent dans les conteneurs. Les étapes 3 à 5 sont répétées au fur et à mesure que les travaux DAG se déroulent en plusieurs vagues.

Houtu est construit au sein de Spark [20] sur un système YARN [17] ; tandis que Zookeeper [72] est utilisé pour synchroniser les JMs pour un même travail. Lorsqu'il a été déployé sur 20 machines réparties dans 4 régions du cloud Alibaba (AliCloud), Houtu a amélioré de 29 % le temps de réponse moyen des travaux et de 31 % le temps d'exécution par rapport à l'architecture décentralisée avec un algorithme d'ordonnancement statique.

Avantages: L'architecture décentralisée proposée dans Houtu offre la flexibilité des travaux mono et multi-clusters tout en respectant les contraintes réglementaires.

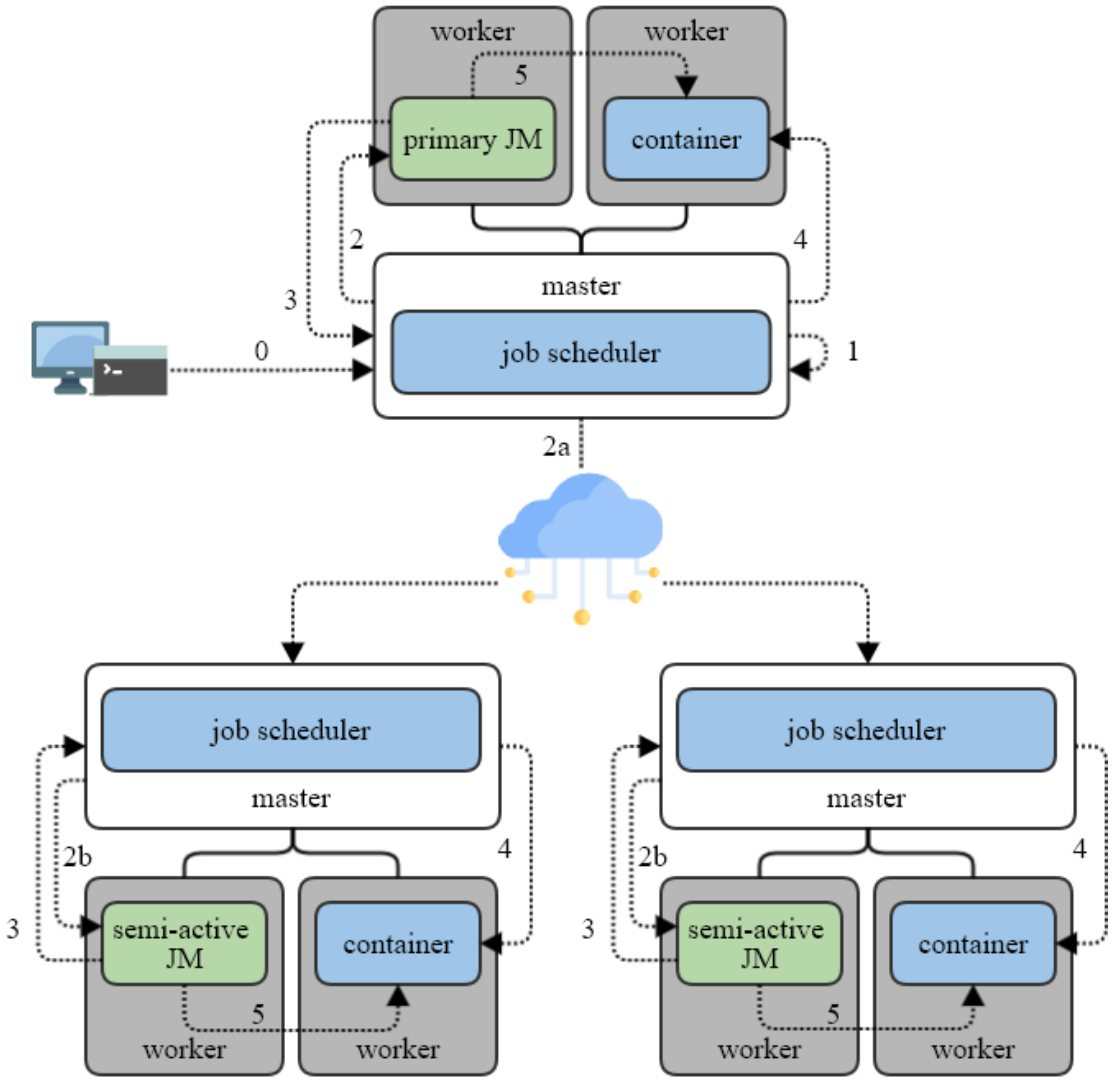


Figure 3.4: Le cycle de vie d'un travail dans Houtu.

Kimchi est un système d'analyse de données géo-distribuées tenant compte des coûts qui détermine *le placement des tâches reduce* en considérant le coût de transfert des données, la bande passante du réseau étendu, la taille et l'emplacement des données intermédiaires ainsi que les préférences pour le compromis entre le coût et la performance des applications [59].

Kimchi est le premier système GDA qui optimise le placement des tâches reduce tout en tenant compte du coût de transfert des données multi-clouds, c'est-à-dire que les fournisseurs de clouds ont des taux de transfert de données différents.

Le placement des tâches est décrit comme un MIP qui prend les entrées suivantes :

- *Préférence de compromis:* les applications doivent fournir un nombre compris entre 0 et 1, 0 étant le coût minimum et 1 la latence minimum.

- *Coût de transfert des données:* ces informations sont recueillies sur les sites Web des fournisseurs de cloud computing.
- *Informations sur la bande passante du réseau:* les exécuteurs situés sur chaque centre de données évaluent la bande passante lors du transfert des données entre les centres de données.
- *Taille des données pour les tâches shuffle:* cette information est disponible auprès du MapOutputTracker de Spark.

La sortie du MIP est un ensemble de paires (tâche, centre) ainsi que la latence et le coût réseau prévus pour chaque tâche. Le MIP est appelé au moment de l'exécution de chaque étape shuffle, puis la sortie est fournie à l'ordonnanceur qui affecte les tâches aux centres de données. Si des changements (conflits du réseau et changements de bande passante) sont détectés, le planificateur appelle un heuristique qui assigne la tâche à un autre centre inactif tout en essayant de respecter la préférence du trade-off.

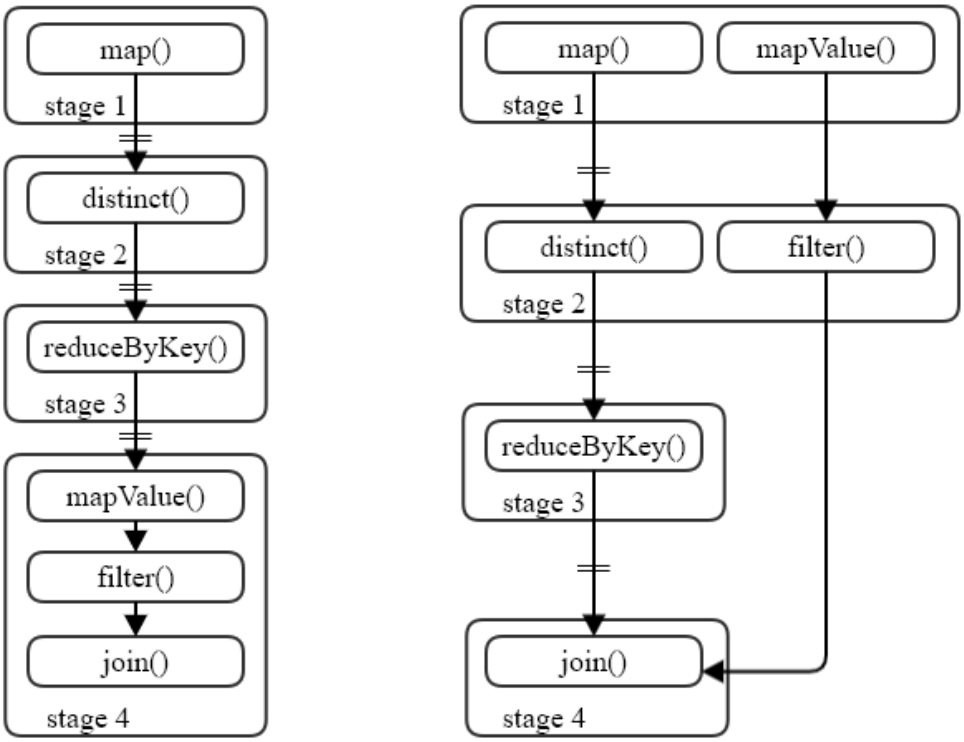
Kimchi [59] est construit sur le framework Spark [20]. Il supporte les nouvelles propriétés de Spark qui contrôlent les paramètres de Kimchi. Kimchi réduirait le coût de 14 à 24 % sans impact sur les performances et réduirait le temps d'exécution des requêtes de 45 à 70 % sans impact sur le coût, par rapport à d'autres approches de base centralisées (coût minimum), vanilla Spark et systèmes tenant compte de la bande passante (par exemple, Iridium [9]).

Avantages: Kimchi offre une grande flexibilité ; les applications peuvent choisir entre la meilleure latence indépendamment du coût et le coût minimum indépendamment de la latence.

Harmony est un framework de traitement géo-distribué qui prend en compte conjointement la bande passante WAN et la capacité de traitement pour la répartition et l'ordonnancement dans le but de minimiser le temps d'exécution des applications [58].

Dans Apache Spark, un programme est défini comme un DAG dont les nœuds et les arêtes représentent respectivement les opérateurs et les dépendances de données. La stratégie de répartition de Spark utilise la dépendance shuffle pour placer les opérateurs dans des étapes qui empêchent certains opérateurs d'être exécutés dans les étapes précédentes, comme le montre la figure 3.5a.

La stratégie de répartition d'Harmony commence par déterminer le chemin critique dans le DAG qui est un ensemble de nœuds d'opérateurs qui fournissent le temps d'exécution le plus court. Un algorithme avide détermine les instants de



(a) La stratégie de répartition de Spark. **(b)** La stratégie de répartition d'Harmony.

Figure 3.5: Programme DAG avec différentes stratégies de répartition.

début et de fin de chaque opérateur et les regroupe en fonction de leur dépendance au shuffle, comme illustré sur la figure 3.5b. Les opérateurs qui sont indépendants du shuffle sont affectés à des étapes en fonction de leurs instants de début et de fin.

Pour minimiser le temps de traitement global, l'ordonnanceur Harmony détermine le temps de traitement minimal de chaque étape à l'aide de la taille des données d'entrée et de la capacité de calcul de chaque centre, puis le plan de transfert des données est calculé sur la base du lien montant et du lien descendant du centre. Avant d'affecter les tâches à chaque emplacement, les données d'entrée sont réaffectées à différents centres de données afin de respecter le temps de traitement minimal de chaque étape et le coût de transfert minimal des données en fonction du plan de transfert des données.

Harmony [58] est implémenté sur Apache Spark et se révèle 1,6 et 2,1 fois plus rapide qu'Iridium lorsqu'il est déployé sur cinq sites AWS EC2 avec des bandes passantes de liaison réseau uniformes et non uniformes, respectivement.

3.5.1.3 Frameworks de traitement de type SQL

Turbo est un framework d'analyse géo-distribué qui utilise l'apprentissage automatique pour prédire la taille de la sortie et le temps d'exécution des requêtes

JOIN afin d'optimiser les performances des requêtes géo-distribuées en changeant dynamiquement le plan d'exécution des requêtes en réponse aux variations des ressources telles que la mémoire, le CPU et la bande passante du WAN [61].

Le framework est conçu pour fonctionner au-dessus des frameworks actuels tels que Spark, Hive et Pig sans modifier les fonctionnalités de la couche inférieure telles que l'ordonnancement/le placement des tâches et la réplication des données.

Une base de données de 15 000 échantillons est construite en exécutant des requêtes du benchmark TPC-H[6] et en enregistrant la taille de la sortie et le temps d'exécution de chaque requête ainsi que des attributs comme le nombre de cœurs du CPU, la taille de la mémoire, la bande passante disponible, etc. Des attributs créés à la main et des croisements d'attributs sont utilisés pour inclure des attributs non linéaires qui pourraient aider à prédire le temps d'exécution et la taille de sortie d'une requête.

L'architecture de Turbo se compose de trois éléments, comme l'illustre la Fig. 3.6 :

- *Entraînement du modèle:* Le Least Absolute Shrinkage and Selection Operator (LASSO) est une méthode d'analyse de régression utilisée pour sélectionner les caractéristiques les plus importantes et écarter celles qui ne sont pas pertinentes pour prédire la taille de la sortie. LASSO seul est incapable de prédire le temps d'exécution car la relation entre les caractéristiques sélectionnées et le temps d'exécution est non linéaire. L'arbre de régression par boosting de gradient est une technique de régression qui produit un modèle de prédiction fort à partir d'un ensemble d'arbres de régression faibles. Contrairement aux modèles linéaires, les modèles d'arbres boostés sont capables de capturer une interaction non linéaire entre les caractéristiques et la cible. Le GBRT prend les caractéristiques sélectionnées par LASSO comme entrée pour prédire le temps d'exécution.

- *Estimateur de coût:* Un plan d'exécution de requête peut contenir une série de jointures, la sortie d'une jointure étant l'entrée de la jointure suivante. Dans le scénario où la sortie est située uniquement dans un centre de données, le coût est prédit par le modèle ML, mais lorsque les tâches reduce sont placées sur plusieurs centres de données, la sortie de la jointure actuelle est répartie sur ces centres et le coût ne peut pas être prédit par le modèle ML. Dans ce cas, le coût est prédit par une heuristique de division et de conquête qui partitionne la jointure en une série de sous-jointures qui peuvent être prédites par le modèle ML.

[6]TPC-H Decision Support Benchmark [Online]. Disponible: http://www.tpc.org/tpch/.

- *Runtime QEP Adjustment:* Adapte régulièrement les plans d'exécution des requêtes aux changements pendant l'exécution en explorant trois politiques gourmandes pour choisir la prochaine jointure par paire avec le moindre coût d'anticipation [61] :
  - Shortest Completion Time First (SCTF) sélectionne la prochaine jonction par paire comme étant celle dont le temps d'exécution est le plus court.
  - Maximum Data Reduction First (MDRF) (Réduction maximale des données d'abord) : sélectionne la paire suivante comme étant celle qui devrait entraîner la plus grande différence de volume entre les données d'entrée et de sortie.
  - Maximum Data Reduction Rate First (MDRRF) sélectionne la paire suivante comme étant celle qui devrait maximiser la réduction des données, c'est-à-dire la taille totale de l'entrée moins la taille de la sortie divisée par le temps d'exécution de la jointure estimée.

Turbo atteint une précision d'estimation des coûts de plus de 95% et réduit les temps d'exécution des requêtes jusqu'à 40% lorsqu'il est déployé sur un cluster Google Cloud avec 33 instances réparties sur 8 régions [61].

Bohr est un système d'analyse de données géo-distribuées tenant compte de la similarité qui réduit le temps du shuffle et par conséquent minimise le temps d'exécution des requêtes [62]. Bohr prétraite les données générées en les stockant dans des cubes de données OLAP [73], puis lorsqu'une requête arrive pour la première fois, Bohr utilise des instructions OLAP (dice, slice ou roll-up) pour récupérer les attributs nécessaires à la requête et lancer une recherche de similarité [74] basée sur ces attributs pour organiser les données. Les bases de données sont ainsi préparé pour le placement des données en fonction de la similarité lorsqu'une requête se répète.

Bohr étend la méthode de placement des tâches et des données d'Iridium [9] en utilisant une trace qui contient les enregistrements les plus représentatifs de sa base de données pour identifier les données à déplacer du centre de données congestionné vers d'autres centres de données. Un programme linéaire (LP) est utilisé pour le placement des tâches reduce et la détection des centres de données congestionnés.

Bohr est basé sur Spark et peut réduire le temps d'exécution des requêtes de 30% par rapport à Iridium lorsqu'il est déployé sur dix régions Amazon EC2 [62].

Avantages: Bohr étend d'avantage Iridium [9] en utilisant la similarité des données pour optimiser les données à déplacer plutôt que de déplacer les bases de données auxquels de nombreuses requêtes accèdent.

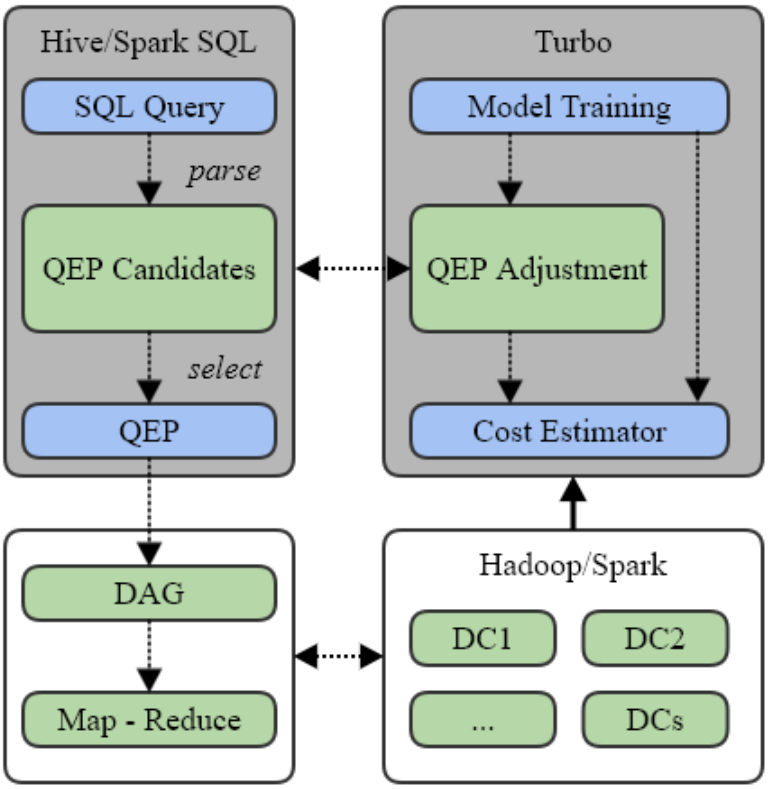


Figure 3.6: Vue d'ensemble de Turbo.

Désavantages: Comme pour Iridium [9], Bohr émet des hypothèses sur l'arrivée des requêtes et ajoute une surcharge en utilisant des cubes OLAP.

3.5.1.4 Autres frameworks de traitement par lots

Hierarchically Distributed Data Matrix - Multi Cluster (HDM-MC) est un framework de traitement des données volumineuses qui permet d'exécuter des analyses de données à grande échelle sur un ou plusieurs clusters [60]. HDM-MC est une extension de la matrice de données distribuées hiérarchiquement (HDM) [75] qui est une représentation de données (qui contient le format, les emplacements, les dépendances entre l'entrée et la sortie) conçue pour prendre en charge l'exécution parallèle d'applications à forte intensité de données. Le framework se compose de trois éléments principaux, qui sont responsables respectivement de la coordination multi-clusters, de la planification et de l'ordonnancement. Nous examinons chacun d'entre eux en détail ci-dessous.

- *Coordination multi-clusters.* HDM-MC prend en charge 2 types d'architecture de coordination: l'architecture hiérarchique et l'architecture décentralisée.

Dans l'architecture hiérarchique, il existe un ou plusieurs clusters super-maîtres qui coordonnent plusieurs clusters enfants-maîtres, chacun d'entre eux contenant des travailleurs/ressources. Dans l'architecture décentralisée, il n'y a pas de super-maître ; chaque maître a deux ou trois maîtres-frères avec lesquels il peut collaborer, tout en gérant ses propres travailleurs et en mettant à jour les informations sur les ressources qui peuvent être utilisées par ses proches.

- *Planification multi-clusters.* La phase de planification s'effectue en deux étapes : la planification des étapes et la planification des tâches. Lors de l'étape de planification des étapes, un travail de traitement qui doit être exécuté est divisé en plusieurs phases de travaux, chacune d'entre elles appartenant à l'une des catégories de travaux suivantes :

  - Local : toutes les bases de données d'entrée se trouvent dans le cluster qui effectue la planification du travail. Cette étape de travail est planifiée dans le cluster actuel.
  - Remote : toutes les bases de données d'entrée se trouvent dans un autre cluster. Cette étape du travail est soumise à l'exécution dans le cluster qui contient les bases de données d'entrée.
  - Collaborative : les bases de données d'entrée sont réparties entre plusieurs clusters. Cette étape du travail est planifiée pour être parallélisée et programmée à la fois sur le cluster actuel et sur les clusters frères.

  Ensuite, à l'étape de planification des tâches, chaque étape de travail identifiée à l'étape de planification des étapes est planifiée pour être exécutée dans l'un des clusters maîtres qui décomposent le travail en tâches pour la planification.

- *Ordonnancement multi-clusters.* Après avoir été décrit, le travail est soumis à un processus d'ordonnancement à deux niveaux. La première couche surveille et ordonnance les étapes de chaque travail, tandis que la deuxième reçoit, surveille et ordonnance les taches de chaque étape active du travail en appliquant l'une des trois stratégies d'ordonnancement suivantes :

  - Delay Scheduling : les tâches qui arrivent attendent pendant un court laps de temps afin d'obtenir une meilleure localisation des données.
  - Min/Max Scheduling : les tâches sont ordonnancées en fonction du temps de réalisation minimum estimé. Min/Max est conscient de la distance entre les travailleurs dans le réseau.

– Algorithme de Hungarian: il s'agit d'un algorithme de graphe qui détermine de manière quasi optimale les distances les plus courtes entre les nœuds (travailleurs) d'un graphe.

Avantages: HDM-MC [60], comme Houtu [27], permet une transition dynamique de l'architecture entre mono- et multi-clusters mais ne repose sur aucun des frameworks big data existants.

3.5.2 Frameworks de traitement de flux géo-distribués

3.5.2.1 Frameworks de traitement par micro-lots

Iridium est un système d'analyse géo-distribué à faible latence qui minimise le temps de réponse des requêtes en optimisant le placement des données et des tâches des requêtes [9]. Le système redistribue les bases de données entre les centres de données avant l'arrivée des requêtes et place les tâches dans des sites ayant une meilleure bande passante afin de réduire les congestions du réseau pendant l'exécution. Le problème de placement des tâches est formulé sous la forme d'un programme linéaire LP qui modélise les bandes passantes des sites et les caractéristiques des requêtes, et il est résolu à l'aide du solveur Gurobi[3]. Un heuristique avide déplace itérativement de petits morceaux des bases de données les plus accédés et/ou les bases de données qui produisent une grande quantité de données intermédiaires. L'architecture d'Iridium se compose de deux éléments principaux, un gestionnaire local et un gestionnaire global centralisé.

- *Gestionnaire local:* Exécute les tâches qui lui sont assignées et assure le suivi des ressources disponibles sur chaque site.
- *Gestionnaire global centralisé:* Convertit les requêtes en graphes acycliques dirigés (DAG) d'étapes, coordonne l'exécution des requêtes et assure le suivi des emplacements des données sur les sites.

Iridium [9] est implémenté au-dessus de Spark et accélère les requêtes de 64% à 92% par rapport à Conviva[7], Bing Edge, TPC-DS[4] et Berkeley Big Data Benchmark[5]. Il permet également de réduire l'utilisation de la bande passante du réseau étendu de 15% à 64%, lorsqu'il est déployé dans huit régions Amazon EC2 [9].

Avantages: Iridium intègre un "bouton" destiné à limiter l'utilisation du réseau étendu. Il minimise la latence d'exécution des requêtes en trouvant et en déplaçant

[7]Conviva [Online]. Disponible: https://www.conviva.com/datasheets/experience-benchmarks.

les données intermédiaires pertinentes avant l'arrivée des requêtes en fonction de l'historique.

Désavantages: Iridium ne tient pas compte de la congestion du réseau entre les centres de données et émet des hypothèses sur l'arrivée des requêtes.

Li et al.(2) [63] ont proposé un framework de streaming basé sur Spark géo-distribué qui vise à réduire le temps de traitement de chaque micro-lot en considérant conjointement la taille des micro-lots, la bande passante, l'ordonnancement des tâches et le routage des flux de données.

Le problème est décrit comme un problème d'optimisation non convexe et résolu par une combinaison de la méthode des multiplicateurs à direction alternée (ADMM) [76] et de LASSO.

L'algorithme ADMM proposé vise à réduire le temps de réponse des requêtes en sélectionnant le chemin le plus rapide (c'est-à-dire le chemin avec la plus grande largeur de bande) pour acheminer le DStream Spark de chaque source vers son site de collecte. L'algorithme peut converger en trois ou quatre itérations. À chaque itération, la taille du lot devient plus petite, ce qui conduit à un arbre de sélection qui a plus de bande passante disponible sur ses liens (une taille de lot plus petite signifie que la requête est exécutée fréquemment, ce qui conduit à plus de flux réseau), tandis qu'un arbre avec moins de bande passante obligera le système à choisir une taille de lot plus grande.

Le framework de streaming proposé, basé sur Spark, réduit la latence du traitement des requêtes et améliore les temps de transfert du réseau par rapport au framework de streaming Spark original lorsqu'il est déployé sur un cluster de sept instances Amazon EC2 avec une bande passante émulée exécutant des requêtes coexistantes de différents types (dont WordCount, Grep et Top-k) [63].

3.5.2.2 Frameworks de traitement de flux natifs

JetStream est un système de traitement de flux géo-distribué qui optimise le traitement grâce à un échantillonnage adaptatif et à l'abstraction de cubes de données [66]. L'architecture du système comporte trois composants principaux : des travailleurs sur chaque nœud pour le traitement des données, un coordinateur centralisé qui gère et distribue le traitement entre les travailleurs disponibles, et un client.

Le cycle de vie d'une requête dans JetStream commence lorsqu'un programme client génère et envoie un graphe de flux de données pour exécution. Le graphe de flux de données est ensuite examiné pour détecter les erreurs de forme et de structure puis transmis au coordinateur. Le coordinateur centralisé commence

à assigner les opérateurs de flux de données liés aux travailleurs puis envoie le sous-ensemble pertinent du graphe à chaque nœud. Les nœuds établissent des connexions réseau entre eux et lancent les opérateurs. La requête s'arrête lorsque le coordinateur envoie un signal d'arrêt ou que toutes les sources envoient un message d'arrêt indiquant qu'il n'y aura plus de données.

Avantages: JetStream traite la limitation de la bande passante du réseau étendu en faisant un compromis entre la qualité des résultats finaux et les performances, ce qui est bon pour les petits réseaux de capteurs.

Désavantages: Le compromis entre la qualité des résultats et les performances entraîne une imprécision des résultats finaux.

Sana est un système de traitement de flux géo-distribué tenant compte du réseau étendu basé sur Apache Flink qui permet de manière incrémentale aux requêtes géo-distribuées de partager leur exécution commune [64].

Le système utilise trois types de stratégies de partage : (1) partage de l'entrée : les requêtes partagent un sous-ensemble commun de données d'entrée, (2) partage de l'opérateur : les requêtes partagent la même exécution/le même traitement de données sur les mêmes données d'entrée, (3) partage de la sortie : les requêtes partagent des parties de la sortie ou des résultats intermédiaires [64]. Le système utilise les stratégies de partage pour générer les plans d'exécution des requêtes et pour planifier l'exécution d'une manière adaptée au réseau étendu.

La conception du système se compose de cinq éléments principaux, comme l'illustre la figure 3.7 :

- *WAN Monitor:* surveille la disponibilité de la bande passante du réseau étendu entre les sites et détecte les liaisons congestionnées grâce au rapport entre l'utilisation actuelle de la bande passante et la bande passante maximale disponible.

- *Shared Job Manager:* assure le suivi du déploiement des requêtes existantes et fournit ces informations à l'optimiseur de requêtes.

- *Query Optimiser:* optimise et génère le plan d'exécution de la requête en identifiant les points communs entre les nouvelles requêtes et les requêtes existantes et en tenant compte des informations sur la bande passante inter-sites.

- *Job Scheduler:* planifie et déploie chaque instance d'opérateur sur un nœud de calcul tout en minimisant la latence et/ou la consommation de bande passante WAN.

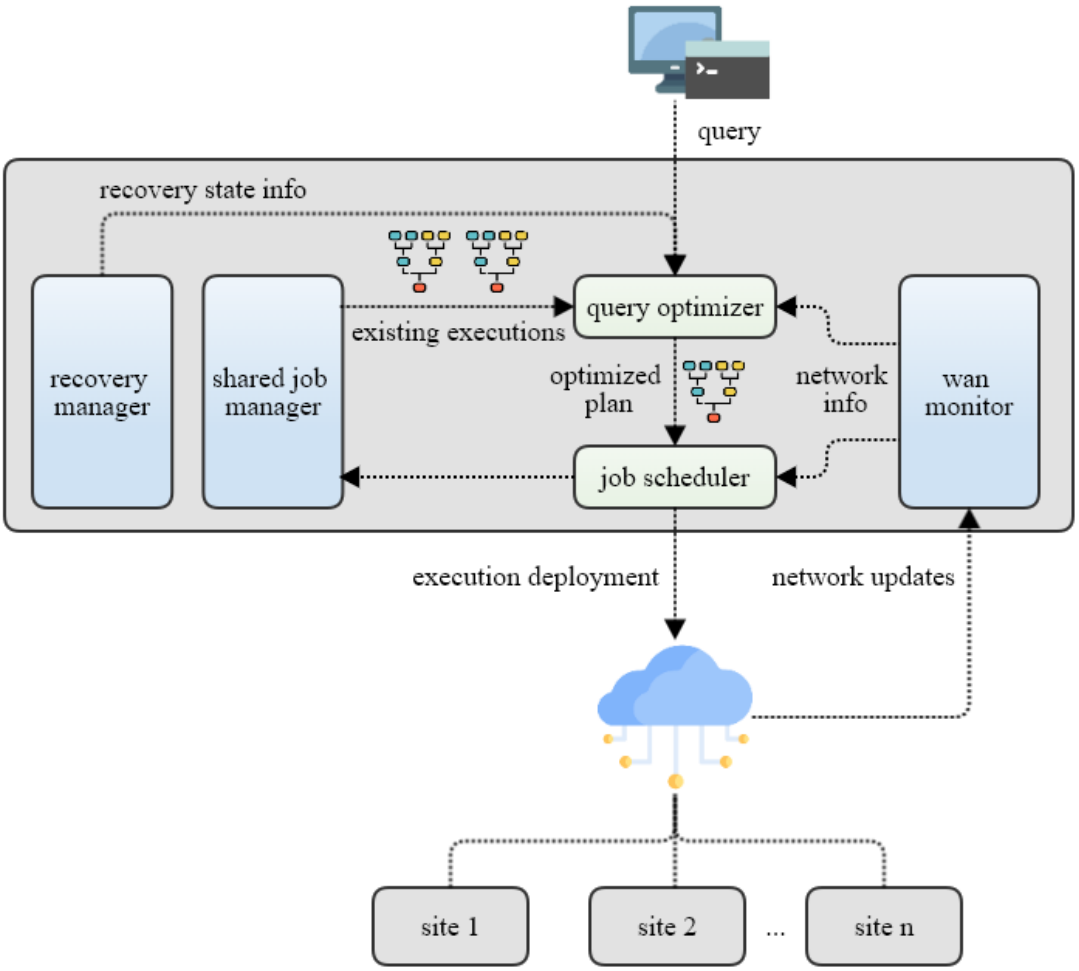


Figure 3.7: L'architecture du système Sana.

- *Recovery Manager:* suit l'état d'exécution de la requête et permet au système, en cas de défaillance, de relancer une requête à partir de son dernier état d'exécution.

Sana augmente le taux de rendement de 21% tout en réduisant l'utilisation de la bande passante du réseau étendu de 33% [64] lorsqu'il est déployé dans 14 centres de données Amazon Elastic Compute Cloud (EC2) géo-distribués avec des données Twitter réelles collectées à partir des API de streaming Twitter[8].

WASP est un système de traitement de flux adaptatif étendu tenant compte des ressources qui vise à maintenir une exécution à faible latence lorsque des changements dynamiques se produisent (variation des ressources ou de la charge de travail, traînards, défaillances) en ajustant le plan physique ou logique des requêtes

[8]Twitter Streaming API's [Online]. Disponible: https://developer.twitter.com/en/docs.

au moment de l'exécution à l'aide de multiples techniques telles que la réaffectation des tâches, la mise à l'échelle des opérateurs et la re-planification des requêtes [65]. L'architecture du système WASP se compose d'un gestionnaire de tâches (JM) et de multiples gestionnaires de tâches (TM) géo-distribués dans chaque centre.

Le JM comprend :

- *Planificateur de requêtes:* Génère les plans d'exécution des requêtes.
- *Ordonnanceur:* Planifie et affecte les tâches aux TM.
- *WAN monitor:* Surveille et signale la largeur de bande du réseau étendu au planificateur de requêtes et à l'ordonnanceur.
- *Moniteur de métriques global:* Rassemble les métriques de chaque TM par le biais d'un moniteur de métriques local et identifie les changements dynamiques.
- *Reconfiguration Manager:* Réajuste le plan d'exécution des requêtes si le moniteur de métriques global détecte un changement dynamique.

La TM comprend :

- *Moniteur de métriques local:* Surveille et recueille les performances des tâches telles que la latence du traitement et les taux de flux d'E/S.
- *Gestionnaire de points de contrôle:* Conserve la trace de l'état des tâches pour permettre aux tâches de commencer/reprendre les exécutions à partir du dernier point de contrôle en cas d'échec de la tâche ou de migration.

Le gestionnaire de reconfiguration, le planificateur de requêtes et l'ordonnanceur travaillent ensemble pour réajuster le plan d'exécution des requêtes en utilisant l'une des techniques suivantes :

- *Réassignation de tâches:* le problème de la réaffectation des tâches est décrit comme un problème ILP qui vise à minimiser le délai de transmission sur le réseau des flux de données en prenant comme contraintes la bande passante entrante et sortante ainsi que les ressources de calcul.
- *Mise à l'échelle de l'opérateur :*
  - Scale up est utilisé pour instancier de nouveaux opérateurs sur le site lorsque des congestions de traitement sont détectées.
  - Scale out est utilisé pour instancier de nouveaux opérateurs sur les différents sites lorsqu'une congestion du réseau est détectée.

  - Scale down est utilisé pour réduire le nombre d'opérateurs lorsque les ressources sont sur-allouées en raison d'une mise à l'échelle ou d'une mauvaise configuration.

- *Planification de requêtes:* le planificateur de requêtes utilise un algorithme heuristique basé sur les coûts pour générer plusieurs plans d'exécution, tandis que l'ordonnanceur calcule le meilleur placement des tâches pour chaque plan et sélectionne le délai le plus faible pour la paire plan/placement.

WASP [65] est implémenté sur Apache Flink [22] et gère les changements dynamiques du réseau étendu avec une faible surcharge tout en maintenant la qualité des résultats lorsqu'il est déployé sur huit nœuds de bordure et huit nœuds CD en utilisant une bande passante et une latence de réseau simulées.

3.6 Conclusion

MapReduce, Spark et Flink sont largement utilisés pour les applications commerciales et la recherche scientifique mais ne sont pas conçus pour l'analyse de données géo-distribuées. Bien que certaines solutions permettent d'exécuter des analyses sur des sites géo-distribués, aucune d'entre elles ne tient compte de la bande passante du réseau étendu (WAN), qui est, selon nous, l'un des facteurs les plus importants du traitement des données géo-distribuées. Dans ce chapitre, nous avons fourni une vue d'ensemble des frameworks les plus utilisés pour l'analyse des big data et discuté des défis que pose la conception de systèmes efficaces de traitement des données géo-distribuées. Nous avons également étudié de nouveaux systèmes capables d'exécuter des analyses géo-distribuées, tout en gérant l'hétérogénéité et en tenant compte de la bande passante du réseau étendu.

Sur la base de cette étude, nous pouvons mettre en évidence les questions et les directions suivantes pour le développement futur: :

- *Sécurité et confidentialité:* en examinant les frameworks de big data géo-distribués, nous avons remarqué qu'aucun d'entre eux ne traite de la sécurité et de la confidentialité. En raison des réglementations sur le stockage et la circulation des données imposées par les gouvernements, nous recommandons aux chercheurs de se concentrer sur la conception de mécanismes d'authentification et de modèles de confiance pour rendre l'analyse des données géo-distribuées applicable et réaliste.

- *Architecture décentralisée:* un autre problème très important que nous avons remarqué tout au long de cette étude est le manque de frameworks supportant l'architecture décentralisée et les clusters multiples. Houtu et HDM-MC sont les seuls systèmes qui supportent de telles fonctionnalités. Nous pensons qu'un système de big data décentralisé et géo-distribué peut offrir une grande flexibilité dans le déploiement car il fournit des clusters autonomes géo-distribués qui peuvent se coordonner pour des tâches géo-distribuées. Un tel système peut éviter une panne générale de l'ensemble du système en cas de défaillance du CD. De plus, un système décentralisé associé à un mécanisme de sécurité/authentification permet de faire face aux contraintes et restrictions réglementaires.

- *Machine learning:* Lube et Turbo sont les premiers à utiliser l'apprentissage automatique pour aider à la planification efficace des tâches en détectant les congestions et en estimant le coût de la durée des requêtes. Cependant, certaines questions restent sans réponse. Quelles sont les indicateurs de performance à choisir pour entraîner un modèle de détection des congestions ? Quelles sont les meilleures caractéristiques pour prédire avec précision et efficacité le temps d'exécution d'une tâche ou d'un travail ? Nous avons également remarqué que le benchmark utilisé pour générer les bases de données d'entraînement et pour évaluer Turbo n'utilise que des données structurées. Nous recommandons donc aux recherches futures de se concentrer sur les données semi-structurées et non structurées pour entraîner et évaluer leurs systèmes.

- *Resource Manager:* les systèmes étudiés n'utilisent pas de gestionnaire de ressources tel que YARN. La bande passante du réseau local a été ajoutée à YARN en tant que ressource récemment [77] mais, à notre connaissance, aucun des systèmes géodistribués existants ne gère la bande passante du réseau étendu en tant que ressource. Un gestionnaire de ressources qui supporte la bande passante WAN peut être moins difficile à concevoir lorsqu'il est combiné à une architecture décentralisée, car le système est capable de faire la distinction entre les travaux locaux et les travaux géo-distribués.

4

Base de données Hadoop pour l'estimation des tâches dans le cloud avec une bande passante limitée

Sommaire

4.1 Introduction 47
4.2 Méthodologie 48
4.3 Dispositif expérimental 49
4.4 La base de données des traces des travaux Hadoop MapReduce 50
4.4.1 Génération de données synthétiques 50
4.4.1.1 Génération de données dans HDFS. 51
4.4.1.2 Description des travaux Hadoop. 52
4.4.2 Collecte des données 53
4.5 Conclusion 57

4.1 Introduction

L'une des caractéristiques de Hadoop MapReduce est le support des services de cloud computing public, qui permet aux organisations d'utiliser des services de cloud computing sur la base d'un paiement à l'utilisation. Cela est avantageux pour les petites et moyennes organisations qui ne peuvent pas mettre en œuvre un cloud privé sophistiqué et à grande échelle en raison de contraintes financières. Par conséquent, l'exécution d'applications Hadoop MapReduce dans un environnement

en cloud pour l'analyse de données volumineuses est devenue une alternative viable pour les praticiens industriels et les chercheurs universitaires.

Puisque l'une des fonctions les plus critiques de Hadoop est la gestion des travaux et des ressources, une gestion plus efficace sera obtenue si l'estimation et la prédiction du temps d'exécution d'un travail sont effectuées avec précision. De plus, les ressources critiques telles que le processeur, la mémoire et la bande passante du réseau sont partagées dans un environnement en cloud et sujettes à des conflits. Il s'agit d'une problématique importante en matière d'ordonnancement efficace, de meilleure consommation d'énergie, de réduction des coûts, de détection de la congestion et de gestion des ressources [78–80].

Plusieurs modèles de performance Hadoop ont été proposés, que ce soit pour un déploiement sur site ou dans le cloud computing [81–86]. Cependant, les données générées et collectées ne sont pas bien décrites. En outre, la plupart des solutions proposées reposent sur des benchmarks qui ne traitent qu'une certaine structure de données.

Dans ce chapitre, nous proposons une nouvelle base de données de traces de travaux Hadoop MapReduce version 2 dans un environnement cloud avec une bande passante réseau limitée.

Le reste de ce chapitre est organisé comme suit. Dans la section Méthodologie nous présentons la méthodologie que nous avons adoptée pour créer la base de données et les objectifs de cette base. La section Dispositif expérimental présente les différents types de déploiement de clusters utilisés dans notre expérimentation. La section La base de données des traces des travaux Hadoop MapReduce décrit les étapes de génération des données synthétiques, de traitement puis collecte des traces de tavaux MapReduce qui construisent notre base de données. Enfin, la section Conclusion conclut le chapitre.

4.2 Méthodologie

Les résultats de notre étude nous amènent à nous concentrer sur l'application de techniques d'apprentissage automatique pour prédire le temps d'exécution des tâches géo-distribuées [10]. À notre connaissance, nous n'avons trouvé aucune base de données publique de traces de travaux Hadoop géo-distribués. Ainsi, la première étape de notre approche est de générer une base de données afin de développer et d'entraîner un modèle pour prédire le temps d'exécution des travaux Hadoop géo-distribués.

Sur la base des articles étudiés dans le chapitre 3, nous avons déterminé les objectifs que notre solution doit inclure et qui ont un impact sur la façon dont nous générons la base de données :

- Nous avons choisi de nous concentrer sur les travaux exécutés sur un cluster Hadoop YARN et plus particulièrement sur les travaux MapReduce v2 car ils sont exécutés dans des conteneurs YARN et le processus pourrait être étendu dans le futur pour inclure des travaux Spark.

- La solution doit être la moins intrusive possible. La modification du code source d'Hadoop dans le but de collecter plus de données entraînera des difficultés lors de l'application de la solution à d'autres versions du framework Hadoop, ainsi dans l'étape de collecte des données, nous ne collectons que les données fournies par le framework Hadoop.

- Le benchmark utilisé doit inclure tous les types de structures de données, les données structurées, semi-structurées et non structurées.

- La base de données doit inclure tous les types d'instructions de requête, y compris JOIN. Les instructions JOIN génèrent des transferts de données intermédiaires sur des liaisons WAN à bande passante limitée, ce qui a un impact sur le temps d'exécution global du travail.

4.3 Dispositif expérimental

Afin d'estimer le temps d'exécution des travaux, nous avons commencé par déployer huit clusters différents avec quatre nœuds, un nœud maître et trois travailleurs sur Google Cloud Platform ; Dataproc[1] est un service de gestion de Spark et Hadoop sur Google Cloud Platform qui permet de créer et de gérer facilement des clusters. La version de Dataproc utilisée est la 1.5-centos8[2] qui comprend CentOS 8 comme système d'exploitation, Apache Spark 2.4.8, Apache Hadoop 2.10.1, Apache Hive 2.3.7 et Python 3.7. Chaque cluster a une configuration de travailleurs/salves différente allant de 2 vCPU à 16, 8 Go de mémoire à 128, et le type et la taille du stockage sont restés les mêmes ; la configuration du nœud maître a été la même tout au long de cette expérimentation avec 4 vCPU et 16 Go de mémoire, les types de VM utilisés dans nos expérimentations sont indiqués dans le tableau 4.1.

[1]Dataproc | Google Cloud [Online]. Disponible: https://cloud.google.com/dataproc.

[2]Dataproc image version list | Google Cloud [Online]. Disponible: https://cloud.google.com/dataproc/docs/concepts/versioning/dataproc-versions.

Table 4.1: Les différents types de VM pour les déploiements de clusters Hadoop

Types de machines	vCPUs	Mémoire (GB)	Bande passante maximale de sortie (Gbps)	Stockage persistant standard
e2-standard-2	2	8	4	
e2-standard-4	4	16	8	
e2-standard-8	8	32	16	
e2-standard-16	16	64	16	500 GB
e2-highmem-2	2	16	4	
e2-highmem-4	4	32	8	
e2-highmem-8	8	64	16	
e2-highmem-16	16	128	16	

Après chaque déploiement, nous avons modifié le facteur de réplication dans HDFS de la valeur par défaut, qui est de 3, à 1. Nous avons également dû changer le moteur d'exécution de Hive de Tez à Mapreduce pour cette expérimentation. Nous avons ensuite limité la bande passante réseau maximale des travailleurs à quatre reprises sur chaque déploiement de cluster à 4,6 Gbps, 2,3 Gbps, 1,1 Gbps et 0,7 Gbps.

4.4 La base de données des traces des travaux Hadoop MapReduce

Dans cette section, nous proposons une base de données de traces de travaux Hadoop dans un environnement en cloud avec une bande passante réseau limitée. Cette base de données comprend de nombreux travaux Hadoop basés sur de multiples méthodes de traitement (MapReduce, Pure QL, NLP...) et différentes structures de données. Cette base de données aidera les chercheurs à développer de nouvelles approches d'ordonnancement et à améliorer les performances d'Hadoop. Cette base de données a été construite dans le but de prédire le nombre de données intermédiaires qui doivent être transférées sur un réseau limité et de prédire le temps d'exécution du travail indépendamment du type d'instruction de la requête.

4.4.1 Génération de données synthétiques

Pour évaluer l'efficacité des systèmes Big Data basés sur Hadoop, nous avons utilisé le Benchmark TPCx-BB Express BB[3]. En exécutant 30 requêtes analytiques

[3]TPCx-BB Express Big Data Benchmark [Online]. Disponible: https://www.tpc.org/tpcx-bb/.

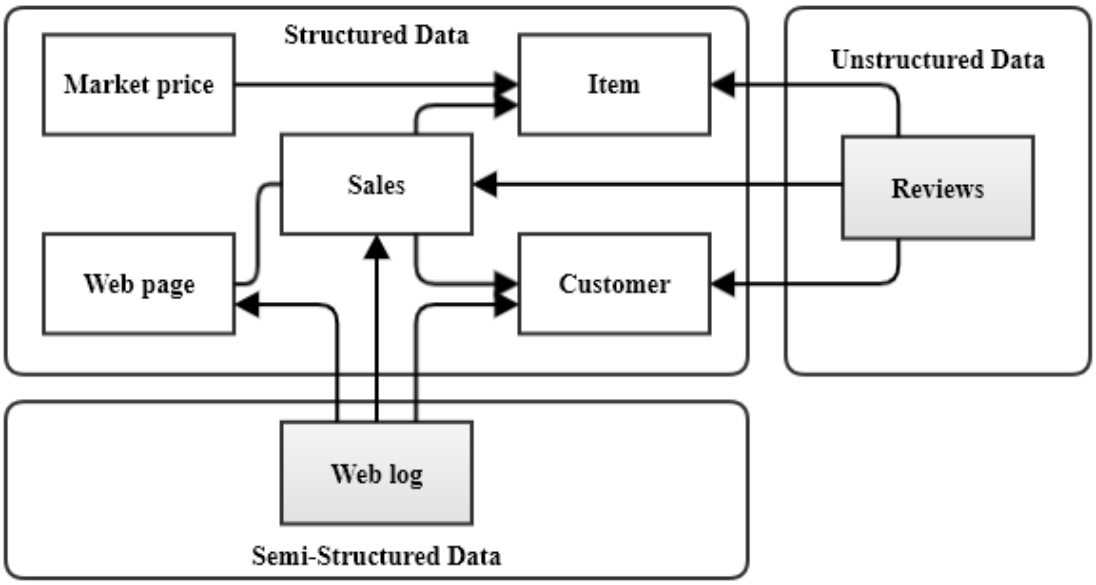


Figure 4.1: Modèle de données TPCx-BB.

fréquemment utilisées dans le contexte du commerce de détail, il évalue les performances des composants logiciels et matériels. Pour les données structurées, les requêtes SQL peuvent faire appel à Hive ou Spark, tandis que pour les données semi-structurées et non structurées, les méthodes d'apprentissage automatique font appel aux bibliothèques ML, aux fonctions définies par l'utilisateur et aux programmes procéduraux.

4.4.1.1 Génération de données dans HDFS.

Afin de alimenter HDFS en données structurées, semi-structurées et non structurées, TPCx-BB utilise une extension du Parallel Data Generation Framework (PDGF). PDGF est un générateur de données parallèle qui génère une grande quantité de données pour un schéma arbitraire. Le PDGF existant peut générer la partie structurée du modèle de référence. Cependant, il ne peut pas générer le texte non structuré des évaluations des produits. Tout d'abord, PDGF est étendu pour produire une base de données clé-valeur pour un ensemble fixe de clés obligatoires et optionnelles. Ceci est suffisant pour générer la partie weblog du benchmark.

Pour générer des données non structurées, un algorithme basé sur la technique de chaîne de Markov est utilisé pour produire un texte synthétique à partir d'un échantillon de texte d'entrée ; l'échantillon d'entrée initial est une évaluation de produits réels provenant d'un magasin de vente au détail en ligne. Un aperçu de haut niveau du modèle de données est présenté dans la figure 4.1.

Table 4.2: Noms et tailles des tables de la base de données du benchmark

Table	Taille	Table	Taille
data/customer	128.5 MB	data/warehouse	2.2 KB
data/customer_address	51.9 MB	data/web_clickstreams	29.7 GB
data/customer_demographics	74.2 MB	data/web_page	98.4 KB
data/date_dim	14.7 MB	data/web_returns	870.4 MB
data/household_demographics	151.5 KB	data/web_sales	21.1 GB
data/income_band	327 B	data/web_site	8.6 KB
data/inventory	16.3 GB	data_refresh/customer	1.3 MB
data/item	65.0 MB	data_refresh/customer_address	537.2 KB
data/item_marketprices	66.6 MB	data_refresh/inventory	168.7 MB
data/product_reviews	633.2 MB	data_refresh/item	673.2 KB
data/promotion	568.5 KB	data_refresh/item_marketprices	26.2 MB
data/reason	38.2 KB	data_refresh/product_reviews	6.3 MB
data/ship_mode	1.2 KB	data_refresh/store_returns	7.3 MB
data/store	33.0 KB	data_refresh/store_sales	152.5 MB
data/store_returns	708.6 MB	data_refresh/web_clickstreams	308.5 MB
data/store_sales	14.7 GB	data_refresh/web_returns	8.8 MB
data/time_dim	4.9 MB	data_refresh/web_sales	219.1 MB

Le benchmark définit un ensemble de facteurs d'échelle basés sur la taille approximative des données brutes générées par PDGF en Gigabytes. Dans notre expérimentation, nous avons utilisé un facteur d'échelle de 100, ce qui donne environ 90 Go de données réparties uniformément sur trois nœuds de données. Le tableau 4.2 indique les noms et les tailles des tables. Il convient de noter que les tailles des tables diffèrent à chaque exécution de la génération de données, c'est-à-dire que la taille des tables est différente sur chaque déploiement de cluster. La base de données générée est principalement non structurée, et les données structurées ne représentent que 20%.

4.4.1.2 Description des travaux Hadoop.

Nous avons utilisé des requêtes basées sur Hive pour réaliser nos expérimentations à partir des 30 requêtes fournies par le benchmark[3] ; nous avons pu exécuter 23 requêtes, et les sept autres requêtes reposent sur des frameworks qui n'entrent pas dans le cadre de notre expérimentation. Les requêtes sont complexes et contiennent diverses clauses telles que SELECT, ORDER BY, GROUP BY, CLUSTER BY et toutes sortes de clauses JOIN. La distribution des différents types de requêtes et les types de données auxquels ils accèdent sont illustrés dans le tableau 4.3.

Table 4.3: La répartition des différents types de requêtes et des types de données

Query	Data type	Method	Query	Data type	Method
1	Structured	UDF/UDTF	14	Structured	Pure QL
2	Semi-Structured	Map Reduce	15	Structured	Pure QL
3	Semi-Structured	Map Reduce	16	Structured	Pure QL
4	Semi-Structured	Map Reduce	17	Structured	Pure QL
6	Structured	Pure QL	19	Un-Structured	UDF/UDTF/NLP
7	Structured	Pure QL	21	Structured	Pure QL
8	Semi-Structured	Map Reduce	22	Structured	Pure QL
9	Structured	Pure QL	23	Structured	Pure QL
10	Un-Structured	UDF/UDTF/NLP	27	Un-Structured	UDF/UDTF/NLP
11	Structured	Pure QL	29	Structured	UDF/UDTF
12	Semi-Structured	Pure QL	30	Semi-Structured	UDF/UDTF/Map Reduce
13	Structured	Pure QL			

4.4.2 Collecte des données

Les requêtes ont généré 101 travaux et 1595 à 1604 tâches par configuration de cluster, soit un total de 3232 travaux et 50732 tâches.

Afin de recueillir des informations sur ces travaux, nous avons développé un toolkit avec python pour recueillir des informations sur les applications, les travaux, les compteurs de travaux et les tâches. La toolkit utilise les API REST fournies par le framework Hadoop[4][5] et SSH pour collecter des données sur les applications, les travaux, les compteurs de travaux, les tâches, les métriques du cluster et la configuration du framework en faisant une requête HTTP, en se connectant au nœud maître par SSH et en analysant les fichiers JSON et XML.

Les caractéristiques collectées et leurs descriptions pour l'application, les métriques du cluster et la configuration de YARN sont présentées dans le tableau 4.4. les caractéristiques collectées pour les tavaux sont présentées dans le tableau 4.5, puis les caractéristiques collectées pour les compteurs de tâches sont présentées dans le tableau 4.6 ; enfin, les caractéristiques collectées pour les tâches sont présentées dans le tableau 4.7.

[4]Apache Hadoop MapReduce 2.10.1 HistoryServer – MapReduce History Server REST APIs. [Online]. Disponible: https://hadoop.apache.org/docs/r2.10.1/hadoop-mapreduce-client/hadoop-mapreduce-client-hs/HistoryServerRest.html.

[5]Apache Hadoop 2.10.1 – ResourceManager REST APIs. [Online]. Disponible: https://hadoop.apache.org/docs/r2.10.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerRest.html.

Table 4.4: Caractéristiques des applications, cluster et YARN collectées

Object Type	Application, Scheduler, Cluster metrics and framework configuration		
How data were acquired	- ResourceManager REST API's allow to get information about the cluster - status on the cluster, metrics on the cluster, scheduler information, information about nodes in the cluster, and information about applications on the cluster. - Connecting through SSH and parsing yarn-site.xml		
File	CSV file, applications.csv		
Application	**Feature name**	**Type**	**Description**
	id	string	The application id
	elapsedTime	long	The elapsed time since the application started (in ms)
	memorySeconds	long	The amount of memory the application has allocated
	vcoreSeconds	long	The number of CPU resources the application has allocated
Cluster metrics	totalMB	long	The amount of total memory in MB
	totalVirtualCores	long	The total number of virtual cores
	totalNodes	int	The total number of nodes
	networkBandwidth	long	The maximum available network bandwidth for each node
YARN configuration	yarn-nodemanager-resource-memory	long	Amount of physical memory, in MB, that can be allocated for containers
	yarn-nodemanager-resource-cpu-vcores	int	Number of vcores that can be allocated for containers
	yarn-scheduler-maximum-allocation	long	The maximum allocation for every container request at the RM in MBs
	yarn-scheduler-minimum-allocation	long	The minimum allocation for every container request at the RM in MBs

Table 4.5: Caractéristiques des travaux collectés

Object Type	Job and MapReduce configuration		
How data were acquired	- MapReduce History Server REST APIs allow to get status on finished jobs. - Connecting through SSH and parsing mapred-site.xml		
File	CSV file, jobs.csv		
Job features	**Feature name**	**Type**	**Description**
	id	string	The job id
	startTime	long	The time the job started
	finishTime	long	The time the job finished
	mapsTotal	int	The total number of maps
	reducesTotal	int	The total number of reduces
	avgShuffleTime	long	The average time of the shuffle (in ms)
MapReduce configuration	mapreduce-job-maps	int	The number of map tasks per job
	mapreduce-map-cpu-vcores	int	The number of virtual cores to request from the scheduler for each map task
	mapreduce-reduce-memory	long	The amount of memory to request from the scheduler for each reduce task
	mapreduce-job-reduce-slowstart-completedmaps	float	Fraction of the number of maps in the job which should be complete before reduces are scheduled for the job
	mapreduce-task-io-sort	long	The total amount of buffer memory to use while sorting files, in megabytes

Table 4.6: Caractéristiques des compteurs de travaux collectés

Object Type	Job Counters		
How data were acquired	MapReduce History Server REST API's allows to get status on finished jobs		
File	CSV file, counters.csv		
Counters features	**Feature name**	**Type**	**Description**
	id	string	The job id
	map-input-records	long	The number of records processed by all the maps
	map-output-records	long	The number of output records emitted by all the maps
	map-output-bytes	long	Output of all the maps in bytes
	reduce-shuffle-bytes	long	Map output copied to Reducers
	file-bytes-read-map	long	The number of bytes read by Map tasks from the local file system
	hdfs-bytes-read-map	long	The number of bytes read by Map and Reduce tasks from HDFS
	file-bytes-written-reduce	long	The number of bytes written by Reduce tasks to local file system
	hdfs-bytes-written-reduce	long	The number of bytes written by Reduce tasks to HDFS

Table 4.7: Caractéristiques des tâches collectées

Object Type	Task		
How data were acquired	MapReduce History Server REST API's allows to get status on finished tasks		
File	CSV file, tasks.csv		
Task features	**Feature name**	**Type**	**Description**
	id	string	The task id
	Job-id	string	The job id
	type	string	The task type - MAP or REDUCE
	elapsedTime	long	The elapsed time since the application started

4.5 Conclusion

Apache Hadoop est une plateforme open-source bien connue pour le traitement de grandes quantités de données. Le temps d'exécution d'un travail doit être estimé avec précision pour une meilleure gestion.

Dans ce chapitre, nous proposons une base de données de traces de travaux Hadoop MapReduce dans un environnement en cloud avec une bande passante réseau limitée. La base de données contient des informations sur la configuration du cluster et du framework ainsi que sur les applications, les travaux, les compteurs et les tâches. L'objectif de cette base de données est de permettre aux chercheurs de développer de nouvelles approches d'ordonnancement et d'améliorer les performances des tâches MapReduce de Hadoop.

Notre objectif est de prédire le temps d'exécution d'un travail Mapreduce. Dans le chapitre suivant, nous allons utiliser la base de données pour mettre au point un modèle d'apprentissage automatique permettant de prédire le temps d'exécution d'un travail Mapreduce.

5

Prédiction du temps d'exécution a l'aide du Deep Learning

Sommaire

5.1 Introduction 58
5.2 Travaux connexes 59
5.2.1 MapReduce 1 60
5.2.1.1 Grey Box 60
5.2.1.2 Black Box 61
5.2.2 MapReduce 2 62
5.2.2.1 Grey Box 62
5.2.2.2 Black Box 62
5.3 Contexte et motivation 63
5.4 Construction du modèle 65
5.4.1 Pré-traitement des données 65
5.4.1.1 Évaluation de la qualité des données 65
5.4.1.2 Nettoyage des données 66
5.4.1.3 Sélection des caractéristiques 66
5.4.2 Modèle d'apprentissage profond 66
5.5 Résultats et discussion 69
5.6 Conclusion 70

5.1 Introduction

Le framework MapReduce est une méthode efficace pour le traitement parallèle des données volumineuses. L'amélioration des performances des clusters MapReduce,

ainsi que la réduction du temps d'exécution des travaux, constituent un défi fondamental de cette approche. En fait, on est confronté à deux défis : comment maximiser le chevauchement d'exécution entre les travaux et comment créer un ordonnancement optimal des travaux. En conséquence, l'un des défis les plus critiques pour atteindre ces objectifs est le développement d'un modèle précis pour estimer le temps d'exécution des travaux en raison du grand nombre et du volume élevé des travaux soumis, des ressources consommables limitées et de la nécessité d'une configuration Hadoop appropriée.

Les réseaux de neurones profonds sont un groupe d'algorithmes appartenant à l'apprentissage profond, qui est un sous-domaine de l'apprentissage automatique. L'apprentissage profond est devenu le système d'apprentissage automatique le plus utilisé dans la littérature, en raison de sa capacité à s'entraîner avec d'énormes quantités de données en utilisant un grand nombre de paramètres de manière très efficace.

Dans ce chapitre, nous présentons un réseau de neurones profonds pour prédire le temps d'exécution des travaux MapReduce grâce la base de données présentée dans le chapitre précédent. Tout d'abord dans la section Travaux connexes, nous étudions la littérature des méthodes ayant une architecture Hadoop similaire et l'utilisation de l'apprentissage automatique, cette étape s'avère déterminante dans la sélection des caractéristiques. Ensuite, dans la section Contexte et motivation, nous étudions l'anatomie de MapReduce et ses performances à chaque étape, puis dans la section Construction du modèle nous décrivons le prétraitement des données, la sélection des caractéristiques et le modèle de deep learning proposé. Par la suite dans la section Résultats et discussion, nous présentons et discutons les résultats obtenus. Enfin, nous concluons le chapitre dans la section Conclusion.

5.2 Travaux connexes

Les méthodes de modélisation des systèmes sont divisés en trois catégories :black box, grey box et white box, elles sont utilisées pour diverses applications telles que l'évaluation des performances, les tests, la simulation et la conception de systèmes.

Black-Box: dans le contexte du benchmarking et de la modélisation des performances, on parle d'approche black-box lorsque le processus de benchmarking et de modélisation est réalisé sans tenir compte du mode de fonctionnement du Framework et des applications sous-jacentes. Pour être plus précis, les modèles de prédiction des performances des applications sont générés à l'aide d'outils d'apprentissage automatique standard en utilisant des données historiques et sans tenir compte des

caractéristiques de performance intrinsèques de la pile de traitement des frameworks comme Apache Hadoop.

White-Box: Uue approche white-box dans le contexte du benchmarking et de la modélisation de la performance consiste à prendre à l'aide de méthodes analytiques. Dans cette approche utopique, toutes les caractéristiques de performance sont incluses dans le processus de modélisation. De telles approches ne modélisent pas seulement le temps d'exécution d'une application, mais elles ajoutent également des modèles basés sur les coûts tels que le coût des E/S, le coût de la mémoire, le coût du CPU et le coût du réseau [87–89]. Les modèles white-box ont une meilleure précision de prédiction que les modèles black-box en raison de leur approche holistique. Cependant, étant donné qu'une telle approche nécessite une connaissance approfondie de tous les éléments internes d'un système qui peuvent inclure le système d'exploitation, la JVM, les frameworks de traitement, les charges de travail et la pile matérielle, il devient difficile de capturer cette complexité avec le modèle basé sur les coûts et donc beaucoup plus difficile à réaliser [90] par rapport aux approches black-box et grey-box.

Grey-Box: en raison de la complexité des frameworks modernes de traitement des big data comme Apache Spark ou Apache Hadoop, il est difficile de construire un modèle de prédiction précis en utilisant une approche black-box ou white-box [90]. Par conséquent, il serait préférable d'utiliser une approche hybride (grey-box) comprenant les avantages des deux méthodes. Il s'agirait notamment d'utiliser des outils d'apprentissage automatique standard pour construire des modèles de performance à partir de données historiques et en tenant compte d'une compréhension légère des paramètres au niveau du framework et de l'application.

Nous avons classé les travaux précédents en deux catégories en fonction de la version de MapReduce en raison de la différence dans la pile Hadoop avec Mapreduce 2 comprenant YARN et non la version 1, nous nous sommes uniquement concentrés sur les solutions qui utilisent l'apprentissage automatique, ce qui exclut par nature toute approche de type white box.

5.2.1 MapReduce 1

5.2.1.1 Grey Box

Zhang et al. [86] a proposé une combinaison de modèles analytiques et ML pour l'estimation du temps d'exécution des tâches MapReduce. Un profileur de travaux est conçu pour recueillir des données sur chaque phase de l'exécution du travail MapReduce, les phases de read, collect, spill, et merge pour la tâche map, et les

phases shuffle et write dans la phase reduce. Un ensemble de micro-benchmarks est ensuite exécuté sur un cluster plus petit que le cluster de production, le job profiler crée les données d'entraînement, puis des modèles de régression linéaire sont utilisés pour estimer le temps d'exécution de chaque phase des tâches MapReduce, le temps d'exécution de la tâche map est estimé comme une somme des durées des phases map, tandis que le temps d'exécution de la tâche reduce est estimé comme une somme des durées des phases shuffle et write. Afin d'estimer le temps d'exécution du travail, un modèle analytique de performance MapReduce ARIA [91] est utilisé, le modèle utilise les informations sur de la durée moyenne et maximale de la tâche map (reduce) pour calculer les limites inférieure et supérieure du temps d'exécution du travail en fonction des ressources allouées (slots map et reduce).

Kadirvel et al. [85] ont développé une approche de type grey box qui utilise des techniques de régression par apprentissage automatique pour estimer la performance des travaux Hadoop. L'approche proposée comprend la sélection systématique de caractéristiques et la prédiction des performances dans des conditions de fonctionnement dégradées qui résultent de pannes de composants. Vingt méthodes sont étudiées, et quatre méthodes (Gaussian Process regression, Multilayer Perceptron, Regression by Discretisation and M5 Pruned Model Tree) présentent une erreur de prédiction inférieure à 12%.

5.2.1.2 Black Box

Khan et al. [84] ont proposé un modèle permettant d'estimer le temps d'exécution d'un travail MapReduce en se basant sur l'historique d'exécution des travaux. Le modèle utilise la technique de régression linéaire localement pondérée (LWLR) afin d'estimer le temps d'exécution du travail. LWLR est une fonction non paramétrique basée sur les instances qui attribue un poids à chaque instance en fonction de sa distance euclidienne par rapport à l'instance de la requête. LWLR attribue un poids élevé à une instance proche de l'instance de requête et un poids faible aux instances éloignées de l'instance de requête. Une fonction gaussienne peut être utilisée pour traiter le poids d'une instance.

Song et al. [83] ont proposé un framework simple pour prédire la performance des jobs Hadoop en utilisant un analyseur de job Hadoop, et un module de prédiction utilisant des méthodes de régression pondérées localement. L'analyseur de tâches échantillonne les données d'entrée et exécute les fonctions Map et Reduce, il collecte ensuite les caractéristiques liées au job (taille des données en entrée, nombre

d'enregistrements) et les paramètres du cluster, l'analyseur estime également la complexité des fonctions Map/Reduce et le rapport entre les données d'entrée et de sortie. La méthode de régression pondérée localement est utilisée en donnant un poids à chaque caractéristique, puis en générant un jeu d'entraînement basé sur la distance pondérée entre la nouvelle instance et celle de la trace historique afin de choisir des types de tâches similaires, puis le nouveau jeu d'entraînement est utilisé pour entraîner le modèle et faire des prédictions.

5.2.2 MapReduce 2

5.2.2.1 Grey Box

Ceesay et al. [81] ont proposé des modèles paramétriques pour estimer le temps d'exécution d'un travail MapReduce donné pour un cluster de nœuds donné sous YARN. Le code MapReduce a été modifié pour permettre la collecte du temps d'exécution de 6 phases d'un job MapReduce (read, collect, spill, merge, shuffle et write). La régression linéaire a donné les meilleurs résultats pour la plupart des phases tandis que les SVM ont donné les meilleurs résultats pour les phases de shuffle et write, ces modèles ont ensuite été utilisés pour prédire le temps d'exécution d'un nouveau travail MR.

5.2.2.2 Black Box

Tariq et al. [82] a proposé un modèle ML pour estimer l'utilisation des ressources telles que le CPU et la mémoire, ainsi que le temps d'exécution des tâches en fonction de la configuration du cluster et de la structure de la base de données. Des arbres de décision sont utilisés pour construire des modèles de prédiction de l'utilisation du CPU, du disque, de la mémoire, du réseau et du temps d'exécution des tâches. Quatre bases de données différentes sont utilisées lors des expérimentations et la validation pour mettre l'accent sur la relation entre la structure des données, la taille des données, les colonnes impliquées dans la requête et les résultats estimés.

Comme mentionné précédemment dans la section 4.2, nous n'avons pu trouver aucun travail antérieur proposant un modèle d'apprentissage automatique pour prédire le temps d'exécution des tâches MapReduce dans un cluster Hadoop géo-distribué, mais plusieurs travaux ont proposé des modèles d'apprentissage automatique pour un cluster Hadoop local, cette étape était cruciale pour choisir les méthodes les plus performantes et pour la sélection des caractéristiques.

La plupart des méthodes proposées reposent sur un profileur de travaux. Il synthétise les propriétés du travail et les performances des fonctions map et reduce. Ce profileur de travaux saisit les propriétés inhérentes à l'application, telles que la sélectivité de la fonction map (reduce), c'est-à-dire le ratio entre la sortie de la fonction map (reduce) et l'entrée de la fonction map (reduce). Ce paramètre décrit la quantité de données produites en tant que sortie de la fonction map (reduce) définie par l'utilisateur, et permet donc de prédire la quantité de données traitées par les autres phases génériques. Le problème avec le paramètre de sélectivité est que la sortie de la fonction map est inconnue au moment de la soumission du travail, et que cette information n'est disponible qu'à la fin de la phase reduce. Nous argumentons que la taille de la sortie map (reduce) devrait être prédite avant de prédire le temps d'exécution du travail ou être omise dans la conception du modèle. En outre, la conception d'un modèle pour chacune des huit phases du pipeline MapReduce fournira une prédiction plus précise, mais au prix d'un surcoût important lors de la collecte de métriques sur chaque phase.

5.3 Contexte et motivation

Pour chaque tâche MapReduce, il existe une tâche map et une tâche reduce. Pour exécuter un travail MapReduce, la phase map est exécutée en premier, puis la phase reduce. Par défaut, lorsque 80% de la phase map est terminée, la phase reduce commence à s'exécuter. Cependant, dans les fichiers de configuration, le paramètre slowstart peut être égal à un, ce qui signifie que la phase reduce est exécutée juste après la fin de la phase map [92]. Comme le montre la figure 5.1, l'exécution d'une tâche map ou reduce implique une séquence de phases de traitement. L'exécution de chaque phase dépend de l'efficacité de cette phase (la quantité de données traitées) et des performances du matériel.

Lorsqu'un fichier est stocké dans Hadoop, il est fragmenté en blocs de taille fixe de 128 Mo par défaut. HDFS est conçu pour avoir une configuration maître-esclave, les blocs de données étant stockés dans les esclaves/Data Nodes (DN) et les métadonnées étant stockées dans le maître/Name Node (NN). Pour chaque

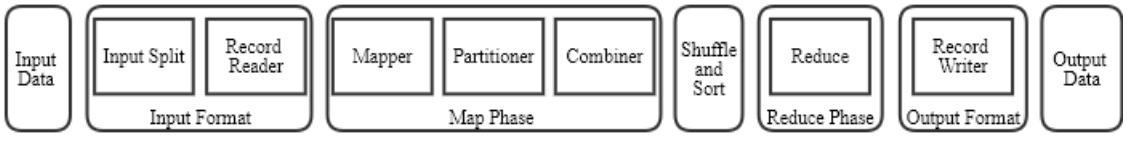


Figure 5.1: Pipeline de traitement MapReduce.

fragment d'entrée, une tâche map est générée. Le fragment est ensuite subdivisé en enregistrements et chaque enregistrement est traité par le mappeur.

MapReduce traite les fragements d'entrée en huit phases, comme suit :

- Record Reader : Il communique avec l'InputSplit et transforme les données en paires clé-valeur pour la lecture par le mapper. Record Reader communique avec l'InputSplit jusqu'à ce que la lecture du fichier soit terminée.

- Mapper : Chaque enregistrement d'entrée est traité par le Mapper et produit une paire clé-valeur intermédiaire. La sortie intermédiaire est stockée temporairement sur le disque local, puis transmise au Combiner.

- Combiner : Le combiner est également connu sous le nom de "mini-reducer". Le combine r est une fonctionnalité optionnelle qui effectue une agrégation locale sur la sortie des mappers, minimisant le transfert de données entre les mappers et les reducers, ce qui améliore les performances globales des reducers. La sortie du combiner est ensuite transmise au partitioner.

- Partitioner : Chaque sortie du mapper est partitionnée en fonction de la valeur de la clé, les enregistrements avec la même valeur de clé vont dans la même partition, puis chaque partition est assignée à un reducer.

- Shuffling et Sorting : Le shuffling est le mouvement physique des données sur le réseau. Une fois que tous les mappers ont terminés et que leur sortie est mélangée, cette sortie intermédiaire est fusionnée et triée, puis fournie comme entrée à la phase reduce. Les valeurs transmises à chaque reducer ne sont pas triées et sont dans un ordre aléatoire. Le tri permet au reducer d'identifier exactement quand une nouvelle tâche de reduce doit commencer, ce qui minimise le temps d'exécution.

- Reducer : Il utilise l'ensemble des paires clé-valeur intermédiaires produites par les mappeurs comme entrée, puis exécute une fonction de réduction sur chacune d'elles pour générer la sortie finale qui est stockée dans HDFS.

- Record Writer : les paires clé-valeur de sortie de la phase reduce sont écrites par le Record Writer dans des fichiers de sortie dans un format défini par Output Format.

- Output Format : Définit comment ces paires clé-valeur de sortie sont écrites dans les fichiers de sortie.

- Output Files : la sortie est stockée dans des fichiers de sortie qui sont généralement stockés dans HDFS.

Le temps d'exécution d'un travail dépend des phases ci-dessus et de certains paramètres de configuration d'Hadoop qui affectent la vitesse de chaque phase.

5.4 Construction du modèle

Dans cette section, nous décrivons le prétraitement des données, la sélection des caractéristiques et le modèle de deep learning proposé pour prédire le temps d'exécution des tâches MapReduce dans un cluster Hadoop avec une bande passante réseau limitée.

Il existe quelques étapes de base à suivre pour une meilleure mise en œuvre d'un réseau neuronal profond. Tout d'abord, le pré-traitement des données est une étape obligatoire pour qu'un réseau obtienne une précision optimale. Vient ensuite la construction du modèle que l'on souhaite expérimenter. Après cela, le choix des hyper-paramètres pour optimiser le processus de formation du modèle choisi, avant de le tester avec de nouvelles données.

5.4.1 Pré-traitement des données

La base de données est divisée en 4 fichiers csv, applications, jobs, counters, et tasks pour chaque configuration de cluster et valeur de bande passante réseau. Comme nous nous concentrons sur le temps d'exécution global des travaux, nous avons omis les informations sur les tâches. Pour regrouper les données dans un seul fichier csv, nous avons joint les applications, les jobs et les counters par l'identifiant du job, puis nous les avons concaténés pour chaque configuration de cluster et de bande passante réseau. La base des données est composée de 64 caractéristiques. Nous avons ensuite suivi les étapes de prétraitement ci-dessous :

5.4.1.1 Évaluation de la qualité des données

L'évaluation de la qualité des données est un processus qui est utilisé pour évaluer la qualité d'une base de données. Elle permet d'identifier les anomalies et les problèmes inhérents aux données, tels que les types de données inadaptés, les valeurs de données mixtes, les données atypiques et les données manquantes.

Grâce à notre processus méticuleux de génération et de collecte de données, la base de données ne contient pas de valeurs manquantes, de données atypiques ou de valeurs mixtes. Dans cette étape, nous n'avons eu qu'à formater les données,

toutes les caractéristiques impliquant des tailles de stockage de données ou des taux de transfert réseau (GB, MB, KB ou Kbps, Mbps) sont converties en octets ou en octets par seconde (bps), nous avons également converti toutes les caractéristiques en millisecondes (ms) en secondes (s).

5.4.1.2 Nettoyage des données

Le processus de nettoyage des données consiste à ajouter les données manquantes et à corriger, réparer ou supprimer les données incorrectes ou non pertinentes d'une base de données.

Dans cette étape, nous avons supprimé les données non pertinentes telles que le type d'application et le statut final de l'application car leur valeur ne change pas ou n'est pas pertinente dans notre étude.

5.4.1.3 Sélection des caractéristiques

La sélection des caractéristiques est le processus qui consiste à décider quelles variables (caractéristiques, traits, catégories, etc.) sont les plus importantes pour l'analyse. Nous avons basé notre sélection de caractéristiques sur les travaux présentés dans la section 5.2 et les travaux sur le réglage des paramètres de MapReduce [93, 94], le tableau 5.1 montre les caractéristiques sélectionnées.

Parmi les 64 caractéristiques, nous en avons sélectionné 19. Nous avons écarté des caractéristiques telles que le nombre d'enregistrements traités par toutes les tâches map (map-input-records) et le nombre de bytes lus par les tâches map depuis hdfs (hdfs-bytes-read-map) car ils sont fortement corrélés avec le nombre de bytes lus par les tâches map depuis le système de fichiers local (file-bytes-read-map), nous avons également écarté toute caractéristique inconnue au moment de la soumission des travaux, comme le nombre de bytes écrits par les tâches map et reduce dans hdfs et dans le système de fichiers local (file-bytes-written-map, file-bytes-written-reduce, hdfs-bytes-written-map, hdfs-bytes-written-reduce).

5.4.2 Modèle d'apprentissage profond

L'apprentissage profond est un sous-domaine de l'apprentissage automatique, connu pour traiter de grandes quantités de données de manière efficace et optimale par rapport aux algorithmes d'apprentissage automatique classiques. L'apprentissage profond est devenu l'état de l'art dans divers domaines d'application, tels que la vision par ordinateur, la reconnaissance vocale, etc. L'apprentissage profond se compose de différents types d'algorithmes, par exemple les réseaux de neurones

Table 5.1: Liste des caractéristiques sélectionnées.

Features	Description
total virtual cores	The cluster total number of virtual cores.
total cluster memory	The cluster amount of total memory in MBs.
total nodes	The cluster total number of nodes.
node manager vcores	Number of vcores that can be allocated for containers.
node manager memory	Amount of physical memory, in MB, that can be allocated for containers.
scheduler min memory	The minimum allocation for every container request at the RM in MBs.
scheduler max memory	The maximum allocation for every container request at the RM in MBs.
network bandwidth(Kbps)	The maximum possible egress and ingress bandwidth per node.
dfs block size	The default block size for new files, in bytes.
IO sort	The total amount of buffer memory to use while sorting files in MBs.
slow start	Fraction of the number of maps in the job which should be complete before reduces are scheduled for the job.
reduce tasks/job	The default number of reduce tasks per job.
map tasks/job	The default number of map tasks per job.
map memory	The amount of memory to request from the scheduler for each map task.
reduce memory	The amount of memory to request from the scheduler for each reduce task.
map vcores	The amount of memory to request from the scheduler for each map task.
reduce vcores	The amount of memory to request from the scheduler for each reduce task.
file-bytes-read-map	The number of bytes read by map tasks from the local file system.
elapsed time	The elapsed time since the application started (in ms).

profonds (DNN) peuvent traiter différents types de données, les réseaux de neurones convolutionnels (CNN) [95] sont spécifiquement adaptés aux données d'image et les réseaux de neurones récurrents (RNN) [96] fonctionnent mieux avec les données séquentielles. Dans notre cas, nous avons adopté un DNN pour la prédiction du

temps d'exécution des tâches MapReduce sur la base de certaines caractéristiques prédéfinies, comme indiqué dans la section 5.4.1.3.

L'objectif principal n'est pas de mettre en œuvre plusieurs modèles pour choisir celui qui est le plus performant, mais plutôt de fixer un modèle afin de déterminer l'impact de la génération de données supplémentaires sur la précision. En outre, il s'agit du premier travail qui utilise l'apprentissage profond comme solution pour la prédiction du temps d'exécution des tâches mapreduce, ce qui répond à la question de l'application de l'apprentissage profond pour résoudre ce problème.

Tout d'abord, nous avons codé les caractéristiques catégorielles en utilisant One Hot Encoding [97]. Enfin, nous avons utilisé la validation croisée (Cross Validation) pour diviser les données en bases de données d'entrainement et de test.

Chaque unité d'entrée de notre base de données dans notre réseau est représentée sous 19 caractéristiques, ce qui signifie que la couche d'entrée a 19 neurones. Ensuite, nous avons ajouté cinq couches cachées, chacune ayant 256 unités. Le choix de cinq couches cachées avec 256 unités a pour effet bénéfique d'extraire tout type de corrélation entre nos caractéristiques d'entrée. Enfin, la couche de sortie contient une seule unité, étant donné que la sortie souhaitée est la prédiction du temps d'exécution d'un certain nombre de travaux.

Dans chaque couche, nous avons adopté la fonction d'activation ReLU (Rectified Linear Units) [98] définie dans l'équation (5.1), étant donné sa popularité et sa robustesse dans la convergence optimale. La figure 5.2 représente l'architecture DNN que nous avons adoptée.

$$f(x) = \max(0, x) \tag{5.1}$$

Pour entraîner notre réseau, nous avons adopté une initialisation normale pour les poids du modèle et nous avons utilisé l'erreur absolue moyenne (MAE) comme fonction de perte telle que formulée dans l'équation (5.2), en plus d'utiliser l'optimiseur Adam [99] comme optimiseur pour notre DNN.

$$MAE = \frac{\sum_{i=1}^{n} |y'_i - y_i|}{n} \tag{5.2}$$

Où y'_i est la valeur prédite, y_i est la valeur de sortie souhaitée et n est le nombre total d'entrées.

Pour alimenter notre réseau en données, nous choisissons une taille de lot de 32, puis nous définissons un hyperparamètre appelé *validation_split* à 0,2, ce qui signifie que 20% des données d'entraînement sont utilisées pour la validation. Nous avons entraîné notre réseau pour 500 époques.

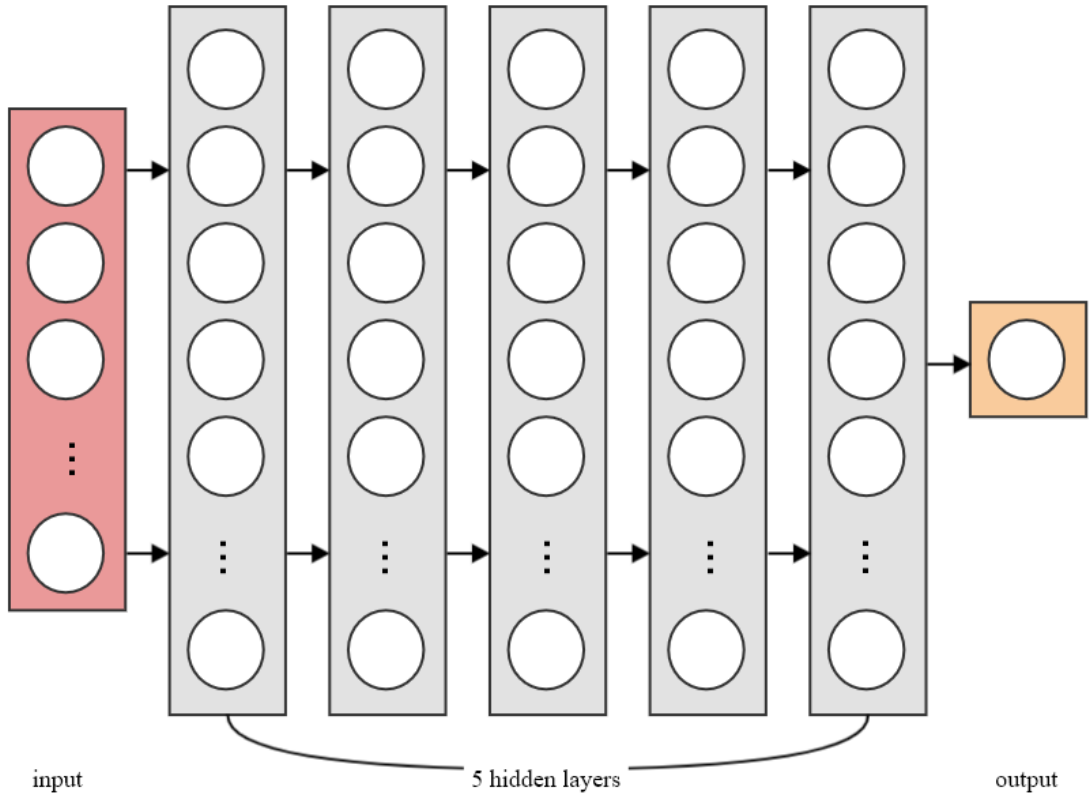


Figure 5.2: L'architecture du réseau de neurone profond (DNN) utilisé dans la méthode proposée.

5.5 Résultats et discussion

Comme nous l'avons mentionné précédemment, nous avons entraîné notre réseau avec différentes quantités de travaux afin d'étudier l'impact de l'ajout de travaux sur les performances du réseau compte tenu des caractéristiques choisies qui sont directement corrélées au temps d'exécution des travaux. À cet égard, nous présentons dans le Tableau 5.2 les performances de notre DNN adopté en termes de précision pour différentes quantités de travaux.

Pour 808 travaux, notre DNN n'a atteint que 29% de précision, ce qui semble normal étant donné la quantité limitée des données. Nous avons ensuite doublé le nombre de travaux et nous avons atteint une précision de 41%. Ensuite, nous avons utilisé 3232 tâches pour entraîner notre DNN et nous avons obtenu 51% de précision.

On peut observer que la précision augmente chaque fois que nous augmentons le nombre de travaux utilisés pour entraîner notre réseau, ce qui est normal étant donné que les réseaux de neurones profonds ont besoin d'énormes quantités de données pour atteindre des précisions plus élevées, ce qui peut être confirmé par notre expérimentation. Notre objectif était de prouver qu'avec les 19 caractéristiques

Table 5.2: Précision de notre DNN pour différentes quantités de travaux.

Nombre de travaux	808	1616	3232
Précision	29%	41%	51%

sélectionnées que nous avons utilisées dans notre expérimentation, nous pouvons prédire le temps qu'un travail peut prendre. En raison de ressources et de financements limités, nous n'avons utilisé que ces quantités de données générées pour mener cette expérience et prouver notre théorie en pratique.

5.6 Conclusion

Dans ce chapitre, nous avons analysé en détail le déroulement d'un travail MapReduce et les approches proposées précédemment pour prédire le temps d'exécution des travaux MR dans les clusters locaux dans le but d'identifier les caractéristiques qui affectent les performances d'un travail MR. Nous avons par la suite proposé un réseau de neurones profonds DNN pour prédire l'exécution des travaux MapReduce avec 19 neurones au niveau de la couche d'entrée et 5 couches cachées comportant 256 unités pour chaque couche. En raison de la quantité limitée de données disponibles, les résultats sont représentés en fonction de la précision de prédiction par rapport au nombre de travaux utilisés pour entraîner le modèle et montrent que lorsque le nombre de travaux est doublé, la précision du modèle augmente de 10%, passant de 29%, 41% et enfin 51%. Ce résultat est encourageant car il prouve que l'apprentissage profond peut être appliqué comme une solution à cette problématique et, avec une quantité suffisante de données, peut atteindre une très bonne précision.

6
Conclusions et Perspectives

Sommaire

6.1 Conclusions 71
6.2 Perspectives 73

6.1 Conclusions

Dans cette thèse, nous nous sommes intéressés à l'optimisation des performances des tâches dans les frameworks d'analyse de données géo-distribuées. À cette fin, nous avons commencé par passer en revue les frameworks d'analyse de big data les plus populaires en donnant une vue d'ensemble de MapReduce et un aperçu complet de l'architecture d'Apache Hadoop et de ses principaux composants (HDFS, YARN), nous avons également donné une brève description des autres frameworks qui font partie de l'écosystème Hadoop (Hive, Spark).

Nous avons passé en revue les frameworks de traitement de données géo-distribuées qui prennent en compte la bande passante du WAN, car c'est la principale caractéristique qui impacte la performance d'un travail géo-distribué, et nous avons donné les avantages et les inconvénients de la plupart d'entre eux. Nous les avons également classés en fonction de la technique de traitement (batching, micro-batching, streaming natif) et du framework de big data sur lequel ils sont basés (MapReduce, Spark, Flink) et les avons comparés en fonction de plusieurs caractéristiques. En conséquence, nous avons pu identifier les problèmes restants

pour des travaux futurs potentiels, tels que le besoin de mécanismes de confidentialité, de sécurité et d'authentification qui sont absents dans tous les frameworks, une architecture décentralisée qui fournit la flexibilité et la tolérance aux pannes dans les systèmes GDA, et l'utilisation de l'apprentissage automatique pour améliorer l'ordonnancement et le placement des tâches.

Suite à notre étude sur les frameworks d'analyse de données géo-distribuées, nous avons identifié Lube et Turbo comme étant les seules solutions qui utilisent l'apprentissage automatique pour aider à la planification efficace des tâches par la détection des congestions et la prédiction du temps de traitement des requêtes. Turbo est basé sur Spark et utilise l'apprentissage machine pour prédire le temps d'exécution des requêtes JOIN et le benchmark utilisé ne fait que peupler le cluster de données structurées, de plus la base de données utilisée n'est pas disponible. Par conséquent, nous avons proposé une nouvelle base de données de traces de travaux Hadoop MapReduce version 2 dans un environnement en cloud avec une bande passante réseau limitée. Tout d'abord, nous avons déployé plusieurs configurations de clusters Hadoop sur GCP, puis nous avons utilisé un benchmark big data pour générer des données synthétiques avec différentes structures ; ces données sont ensuite traitées à l'aide de requêtes de type SQL avec tous types d'instructions.

Nous avons ensuite proposé un toolkit python basé sur les API REST fournies par le framework Hadoop pour construire la base de données en récupérant les traces des travaux MapReduce et les paramètres de configuration du cluster. La base de données est composée de 3232 jobs MR et 50732 tâches.

Enfin, la base de données proposée est utilisée pour prédire le temps d'exécution des travaux MapReduce, ce qui permet une gestion plus efficace des travaux et des ressources. Par conséquent, nous avons étudié en détail l'exécution d'un travail MapReduce et les méthodes précédentes pour prédire le temps d'exécution des travaux MR dans les clusters locaux afin d'identifier les caractéristiques qui ont un impact sur la performance d'un travail MR. Nous avons ensuite proposé un réseau neuronal profond avec 20 neurones dans la couche d'entrée et 5 couches cachées avec 256 unités pour chaque couche. Avec la quantité limitée de données disponibles, les résultats sont représentés en termes de précision de prédiction par rapport au nombre de travaux utilisés pour entraîner le modèle et montrent que chaque fois que le nombre de travaux est doublé, le modèle gagne environ 10% en précision, passant de 29%, 41% et finalement 51%. Le résultat est prometteur car il montre que l'apprentissage profond peut être appliqué comme une solution à ce problème et, avec suffisamment de données, peut atteindre une très bonne précision.

Nous avons démarré cette thèse en 2016 en nous intéressant à l'évaluation des performances dans l'Internet des objets qui s'est transformé en Big Data et à l'optimisation des travaux MapReduce dans un cluster Hadoop local en combinant des nœuds locaux avec des nœuds cloud plus puissants à la demande, c'est la première fois que nous avons rencontré le terme d'analyse de big data géo-distribuée. Nous avons exploré de nombreux domaines de recherche et d'idées tels que l'ajout de la bande passante WAN comme ressource à YARN et l'ordonnancement et le placement des tâches avec LP, ILP et MIP. Le manque de financement et de matériels ont été les principaux facteurs limitants de notre recherche, le déploiement d'une telle infrastructure est très coûteux, nous avons dû être très attentifs aux dépenses lors de nos prises de décision. Néanmoins, nous avons réussi à atteindre l'objectif que nous nous étions fixé, en générant notre propre base de données et en proposant le premier modèle d'apprentissage profond pour prédire le temps d'exécution des travaux MapReduce.

6.2 Perspectives

Notre travail de recherche a laissé quelques pistes non explorées, en raison du manque de financement et de ressources matérielles. Cependant, nous sommes parvenus à des résultats satisfaisants qui peuvent conduire à d'autres découvertes, parmi lesquelles nous tenons à mentionner ces quelques unes :

1. Configuration Matérielle des noeuds: Notre configuration expérimentale dans la section 4.3 est très limitée en termes de configurations matérielles, les configurations des machines virtuelles dans GCP de type standard N2[1] peuvent aller de 2 à 128 vCPUs et de 8 à 512 GB de mémoire. Ce problème est plus lié au financement qu'à une question technique, mais il peut permettre d'obtenir plus de données et donc des prédictions plus précises.

2. Hétérogénéité:

    - *Hétérogénéité des nœuds*: Nous sommes conscients que notre expérimentation manque d'hétérogénéité en termes de configuration des VM et du nombre de nœuds dans le cluster, dans chaque déploiement de cluster, les nœuds ont la même configuration matérielle et le même nombre, ce qui n'est pas représentatif d'un scénario réel, surtout dans une infrastructure géo-distribuée.

[1] General-purpose machine family | Compute Engine Documentation | Google Cloud [Online]. Disponible: https://cloud.google.com/compute/docs/general-purpose-machines#n2-standard.

- *Hétérogénéité des données*: les données synthétiques générées à l'aide du benchmark peuvent également augmenter avec le nombre et la configuration des nœuds, les données générées sont aussi équitablement réparties entre les nœuds, ce qui n'est pas non plus représentatif d'une infrastructure géo-distribuée. Si les nœuds sont répartis sur plusieurs régions et sont responsables de la collecte et du stockage des données des capteurs de cette région, alors la taille des données stockées dépend du nombre de capteurs et du taux de génération de données des capteurs de cette région.
- *Hétérogénéité du WAN*: Nous sommes conscients que le fait de limiter la bande passante de tous les nœuds à la même valeur n'est pas représentatif de la bande passante du WAN, néanmoins c'est un scénario réel car la bande passante maximale de sortie[2] dépend de la configuration de la VM dans GCP.

  Deux solutions peuvent être appliquées afin d'avoir une bonne représentation de la bande passante du WAN: Déployer réellement le cluster dans plusieurs régions puis effectuer les expérimentations, ou simuler le WAN avec la bande passante, la latence et les paquets abandonnés en utilisant des outils tels que WANem[3] et dummynet [100].

[2]Network bandwidth | Compute Engine Documentation | Google Cloud [Online]. Disponible: https://cloud.google.com/compute/docs/network-bandwidth.

[3]WANEM : The Wide Area Network Emulator [Online]. Disponible: https://wanem.sourceforge.net/.

Bibliographie

[1] P. Zikopoulos et al. *Harness the Power of Big Data The IBM Big Data Platform*. 1st. Mcgraw-Hill, 2012.

[2] Jules J. Berman. *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[3] Martin Hilbert. "Big Data for Development: A Review of Promises and Challenges". In: *Development Policy Review* 34 (Jan. 2016), pp. 135–174.

[4] Daniel E. O'Leary. "Artificial Intelligence and Big Data". In: *IEEE Intelligent Systems* 28.2 (2013), pp. 96–99.

[5] Avita Katal, Mohammad Wazid, and R. H. Goudar. "Big data: Issues, challenges, tools and Good practices". In: *2013 Sixth International Conference on Contemporary Computing (IC3)*. 2013, pp. 404–409.

[6] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (2008), pp. 107–113.

[7] Michael Isard et al. "Dryad: Distributed Data-parallel Programs from Sequential Building Blocks". In: *SIGOPS Oper. Syst. Rev.* 41.3 (2007), pp. 59–72.

[8] Ashish Vulimiri et al. "WANalytics: Geo-Distributed Analytics for a Data Intensive World". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD '15. Melbourne, Victoria, Australia: ACM, 2015, pp. 1087–1092.

[9] Qifan Pu et al. "Low Latency Geo-distributed Data Analytics". In: *SIGCOMM Comput. Commun. Rev.* 45.4 (2015), pp. 421–434.

[10] Mohammed Bergui, Said Najah, and Nikola S. Nikolov. "A survey on bandwidth-aware geo-distributed frameworks for big-data analytics". In: *Journal of Big Data* 8 (2021), pp. 1–26.

[11] Mohammed Bergui, Nikola S. Nikolov, and Said Najah. "Hadoop dataset for job estimation in the cloud with limited bandwidth". In: *Proceedings of the 2023 Future of Information and Communication Conference (FICC)*. San Francisco, United States, 2023.

[12] *Apache Hadoop*. URL: http://hadoop.apache.org/.

[13] Kenjiro Taura et al. "Phoenix: A Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources". In: *SIGPLAN Not.* 38.10 (June 2003), pp. 216–229. URL: https://doi.org/10.1145/966049.781533.

[14] Bingsheng He et al. “Mars: A MapReduce Framework on Graphics Processors”. In: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. PACT ’08. Toronto, Ontario, Canada: Association for Computing Machinery, 2008, pp. 260–269. URL: https://doi.org/10.1145/1454115.1454152.

[15] Yunhong Gu and Robert Grossman. “Sector and sphere: The design and of a high-performance data cloud”. In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 367 (July 2009), pp. 2429–45.

[16] Tom White. *Hadoop: The Definitive Guide*. 4th. O’Reilly Media, Inc., 2015.

[17] Vinod Kumar Vavilapalli et al. “Apache Hadoop YARN: Yet Another Resource Negotiator”. In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC ’13. Santa Clara, California: ACM, 2013, 5:1–5:16.

[18] Ashish Thusoo et al. “Hive: A Warehousing Solution over a Map-Reduce Framework”. In: *Proc. VLDB Endow.* 2.2 (Aug. 2009), pp. 1626–1629. URL: https://doi.org/10.14778/1687553.1687609.

[19] H. Geng. *Internet of Things and Data Analytics Handbook*. Wiley, 2017. URL: https://books.google.com.ec/books?id=V3mYDQAAQBAJ.

[20] Matei Zaharia et al. “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI’12. San Jose, CA: USENIX Association, 2012, pp. 2–2.

[21] Matei Zaharia et al. “Discretized Streams: Fault-tolerant Streaming Computation at Scale”. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP ’13. ACM, 2013, pp. 423–438.

[22] P. Carbone et al. “Apache Flink: Stream and Batch Processing in a Single Engine”. In: *IEEE Data Eng. Bull.* 38 (2015), pp. 28–38.

[23] Michael Armbrust et al. “Spark SQL: Relational Data Processing in Spark”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 1383–1394. URL: https://doi.org/10.1145/2723372.2742797.

[24] Ankit Toshniwal et al. “Storm@twitter”. In: SIGMOD ’14. Snowbird, Utah, USA: Association for Computing Machinery, 2014, pp. 147–156. URL: https://doi.org/10.1145/2588555.2595641.

[25] Sanjeev Kulkarni et al. “Twitter Heron: Stream Processing at Scale”. In: SIGMOD ’15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 239–250. URL: https://doi.org/10.1145/2723372.2742788.

[26] Kevin Hsieh et al. “Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds”. In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 629–647.

[27] Xiaoda Zhang et al. “Towards Reliable (and Efficient) Job Executions in a Practical Geo-distributed Data Analytics System”. In: *arXiv e-prints*, arXiv:1802.00245 (2018), arXiv:1802.00245. arXiv: 1802.00245.

[28] Ashish Vulimiri et al. “Global Analytics in the Face of Bandwidth and Regulatory Constraints”. In: *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*. NSDI’15. Oakland, CA: USENIX Association, 2015, pp. 323–336.

[29] Raajay Viswanathan, Ganesh Ananthanarayanan, and Aditya Akella. “CLARINET: WAN-aware Optimization for Analytics Queries”. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI’16. Savannah, GA, USA: USENIX Association, 2016, pp. 435–450.

[30] R. Tudoran, G. Antoniu, and L. Bougé. “SAGE: Geo-Distributed Streaming Data Analysis in Clouds”. In: *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*. 2013, pp. 2278–2281.

[31] Lizhe Wang et al. “G-Hadoop: MapReduce Across Distributed Data Centers for Data-intensive Computing”. In: *Future Gener. Comput. Syst.* 29.3 (2013), pp. 739–750.

[32] C. Jayalath, J. Stephen, and P. Eugster. “From the Cloud to the Atmosphere: Running MapReduce across Data Centers”. In: *IEEE Transactions on Computers* 63.1 (2014), pp. 74–87.

[33] A. Jonathan et al. “Nebula: Distributed Edge Cloud for Data Intensive Computing”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.11 (2017), pp. 3229–3242.

[34] R. Kettimuthu et al. “Differentiated Scheduling of Response-Critical and Best-Effort Wide-Area Data Transfers”. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 1113–1122.

[35] Z. Hu, B. Li, and J. Luo. “Flutter: Scheduling tasks closer to data across geo-distributed datacenters”. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 2016, pp. 1–9.

[36] Srikanth Sundaresan et al. “Broadband Internet Performance: A View from the Gateway”. In: *SIGCOMM Comput. Commun. Rev.* 41.4 (2011), pp. 134–145.

[37] Ramesh K. Sitaraman et al. “Overlay Networks: An Akamai Perspective”. In: *Advanced Content Delivery, Streaming, and Cloud Services*. John Wiley & Sons, Ltd, 2014. Chap. 16, pp. 305–328.

[38] Chien-Chun Hung et al. “Wide-area Analytics with Multiple Resources”. In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys ’18. Porto, Portugal: ACM, 2018, 12:1–12:16.

[39] A. C. Zhou, S. Ibrahim, and B. He. “On Achieving Efficient Data Transfer for Graph Processing in Geo-Distributed Datacenters”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 1397–1407.

[40] Konstantinos Kloudas et al. “Pixida: Optimizing Data Parallel Jobs in Wide-Area Data Analytics”. In: *Proc. VLDB Endow.* 9.2 (2015), pp. 72–83.

[41] A. Jonathan, A. Chandra, and J. Weissman. “Awan: Locality-Aware Resource Manager for Geo-Distributed Data-Intensive Applications”. In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. 2016, pp. 32–41.

[42] L. Zhao et al. “Optimizing Geo-Distributed Data Analytics with Coordinated Task Scheduling and Routing”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.2 (2020), pp. 279–293.

[43] Yuzhen Huang et al. “Yugong: Geo-Distributed Data and Job Placement at Scale”. In: *Proc. VLDB Endow.* 12.12 (Aug. 2019), pp. 2155–2169.

[44] A. Iordache et al. “Resilin: Elastic MapReduce over Multiple Clouds”. In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 2013, pp. 261–268.

[45] Rajagopal Ananthanarayanan et al. “Photon: Fault-Tolerant and Scalable Joining of Continuous Data Streams”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 577–588.

[46] S. Dolev et al. “A Survey on Geographically Distributed Big-Data Processing Using MapReduce”. In: *IEEE Transactions on Big Data* 5.1 (2019), pp. 60–80.

[47] S. Ji and B. Li. “Wide area analytics for geographically distributed datacenters”. In: *Tsinghua Science and Technology* 21.2 (2016), pp. 125–135.

[48] Sushant Jain et al. “B4: Experience with a Globally-deployed Software Defined Wan”. In: *SIGCOMM Comput. Commun. Rev.* 43.4 (2013), pp. 3–14.

[49] Chi-Yao Hong et al. “Achieving High Utilization with Software-driven WAN”. In: *SIGCOMM Comput. Commun. Rev.* 43.4 (2013), pp. 15–26.

[50] Matt Calder et al. “Mapping the Expansion of Google’s Serving Infrastructure”. In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC ’13. Barcelona, Spain: ACM, 2013, pp. 313–326.

[51] Hao Wang and Baochun Li. “Lube: Mitigating Bottlenecks in Wide Area Data Analytics”. In: *Proceedings of the 9th USENIX Conference on Hot Topics in Cloud Computing*. HotCloud’17. Santa Clara, CA: USENIX Association, 2017, pp. 1–1.

[52] P. A. R. S. Costa et al. “Medusa: An Efficient Cloud Fault-Tolerant MapReduce”. In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 2016, pp. 443–452.

[53] P. A. R. S. Costa, F. M. V. Ramos, and M. Correia. “Chrysaor: Fine-Grained, Fault-Tolerant Cloud-of-Clouds MapReduce”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 421–430.

[54] P. A. R. S. Costa, F. M. V. Ramos, and M. Correia. “On the Design of Resilient Multicloud MapReduce”. In: *IEEE Cloud Computing* 4.4 (2017), pp. 74–82.

[55] Moïse W. Convolbo et al. “GEODIS: Towards the Optimization of Data Locality-aware Job Scheduling in Geo-distributed Data Centers”. In: *Computing* 100.1 (2018), pp. 21–46.

[56] P. Li et al. “Traffic-Aware Geo-Distributed Big Data Analytics with Predictable Job Completion Time”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.6 (2017), pp. 1785–1796.

[57] G. Zhang et al. “Improving Performance for Geo-Distributed Data Process in Wide-Area”. In: *2017 IEEE International Conference on Computer and Information Technology (CIT)*. 2017, pp. 162–167.

[58] H. Zhang, L. Ramapantulu, and Y. M. Teo. “Harmony: An Approach for Geo-distributed Processing of Big-Data Applications”. In: *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. 2019, pp. 1–11.

[59] K. Oh, A. Chandra, and J. Weissman. “A Network Cost-aware Geo-distributed Data Analytics System”. In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 2020, pp. 649–658.

[60] D. Wu et al. “Towards Big Data Analytics across Multiple Clusters”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 218–227.

[61] Hao Wang, Di Niu, and Baochun Li. “Dynamic and Decentralized Global Analytics via Machine Learning”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC ’18. ACM, 2018, pp. 14–25.

[62] Hangyu Li, Hong Xu, and Sarana Nutanong. “Bohr: Similarity Aware Geo-distributed Data Analytics”. In: *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*. USENIX Association, 2017.

[63] W. Li et al. “Wide-Area Spark Streaming: Automated Routing and Batch Sizing”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.6 (2019), pp. 1434–1448.

[64] Albert Jonathan, Abhishek Chandra, and Jon Weissman. “Multi-Query Optimization in Wide-Area Streaming Analytics”. In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC ’18. Carlsbad, CA, USA: ACM, 2018, pp. 412–425.

[65] Albert Jonathan, Abhishek Chandra, and Jon Weissman. “WASP: Wide-Area Adaptive Stream Processing”. In: *Proceedings of the 21st International Middleware Conference*. Middleware ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 221–235.

[66] Ariel Rabkin et al. “Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area”. In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI’14. Seattle, WA: USENIX Association, 2014, pp. 275–288.

[67] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Inc., 1990.

[68] Tiberiu Chis. “Sliding Hidden Markov Model for Evaluating Discrete Data”. In: *Computer Performance Engineering*. Ed. by Maria Simonetta Balsamo, William J. Knottenbelt, and Andrea Marin. Springer Berlin Heidelberg, 2013, pp. 251–262.

[69] Kay Ousterhout et al. “Sparrow: Distributed, Low Latency Scheduling”. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP ’13. Farminton, Pennsylvania: ACM, 2013, pp. 69–84.

[70] Matei Zaharia et al. “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling”. In: *Proceedings of the 5th European Conference on Computer Systems*. EuroSys ’10. Paris, France: ACM, 2010, pp. 265–278.

[71] Robert Grandl et al. “Multi-resource Packing for Cluster Schedulers”. In: *SIGCOMM Comput. Commun. Rev.* 44.4 (2014), pp. 455–466.

[72] Patrick Hunt et al. “ZooKeeper: Wait-free Coordination for Internet-scale Systems”. In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC’10. Boston, MA: USENIX Association, 2010, pp. 11–11.

[73] Alfredo Cuzzocrea, Ladjel Bellatreche, and Il-Yeol Song. “Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions”. In: *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP*. DOLAP ’13. San Francisco, California, USA: ACM, 2013, pp. 67–70.

[74] Edgar Chávez et al. “Searching in Metric Spaces”. In: *ACM Comput. Surv.* 33.3 (2001), pp. 273–321.

[75] L. Wang et al. “MapReduce across Distributed Clusters for Data-intensive Applications”. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*. 2012, pp. 2004–2011.

[76] Stephen Boyd et al. “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”. In: *Foundations and Trends® in Machine Learning* 3.1 (2011), pp. 1–122. URL: http://dx.doi.org/10.1561/2200000016.

[77] X. Fan et al. “Adding network bandwidth resource management to Hadoop YARN”. In: *2017 Seventh International Conference on Information Science and Technology (ICIST)*. 2017, pp. 444–449.

[78] Sam R. Alapati. *Expert Hadoop Administration: Managing, Tuning, and Securing Spark, YARN, and HDFS*. 1st. Addison-Wesley Professional, 2016.

[79] Safanaz Heidari et al. “Big data clustering with varied density based on MapReduce”. In: *Journal of Big Data* 6 (Aug. 2019).

[80] Rupinder Singh and Puneet Kaur. “Analyzing performance of Apache Tez and MapReduce with hadoop multinode cluster on Amazon cloud”. In: *Journal of Big Data* 3 (Oct. 2016).

[81] Sheriffo Ceesay, Adam Barker, and Yuhui Lin. “Benchmarking and Performance Modelling of MapReduce Communication Pattern”. In: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2019, pp. 127–134.

[82] Hassan Tariq, Harith Al-Sahaf, and Ian Welch. “Modelling and Prediction of Resource Utilization of Hadoop Clusters: A Machine Learning Approach”. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. UCC’19. Auckland, New Zealand: Association for Computing Machinery, 2019, pp. 93–100. URL: https://doi.org/10.1145/3344341.3368821.

[83] Ge Song et al. “A Hadoop MapReduce Performance Prediction Method”. In: *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*. 2013, pp. 820–825.

[84] Mukhtaj Khan et al. “Hadoop Performance Modeling for Job Estimation and Resource Provisioning”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.2 (2016), pp. 441–454.

[85] Selvi Kadirvel and Jose A. B. Fortes. “Grey-Box Approach for Performance Prediction in Map-Reduce Based Platforms”. In: *2012 21st International Conference on Computer Communications and Networks (ICCCN)*. 2012, pp. 1–9.

[86] Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo. “Benchmarking Approach for Designing a Mapreduce Performance Model”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ICPE ’13. Prague, Czech Republic: Association for Computing Machinery, 2013, pp. 253–258. URL: https://doi.org/10.1145/2479871.2479906.

[87] Herodotos Herodotou and Shivnath Babu. “Profiling, What-If Analysis, and Cost-Based Optimization of MapReduce Programs”. In: *Proc. VLDB Endow.* 4.11 (June 2020), pp. 1111–1122. URL: https://doi.org/10.14778/3402707.3402746.

[88] Herodotos Herodotou. *Hadoop Performance Models*. 2011. URL: https://arxiv.org/abs/1106.0940.

[89] Torsten Hoefler et al. “Performance modeling for systematic performance tuning”. In: *SC ’11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 2011, pp. 1–12.

[90] Guolu Wang, Jungang Xu, and Ben He. “A Novel Method for Tuning Configuration Parameters of Spark Based on Machine Learning”. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2016, pp. 586–593.

[91] Fengguang Tian and Keke Chen. “Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds”. In: *2011 IEEE 4th International Conference on Cloud Computing*. 2011, pp. 155–162.

[92] Shivnath Babu. “Towards Automatic Optimization of MapReduce Programs”. In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC ’10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 137–142. URL: https://doi.org/10.1145/1807128.1807150.

[93] Narges Peyravi and Ali Moeini. “Estimating runtime of a job in Hadoop MapReduce”. In: *Journal of Big Data* 7 (2020), pp. 1–18.

[94] Lizhen Shi et al. “A case study of tuning MapReduce for efficient Bioinformatics in the cloud”. In: *Parallel Computing* 61 (2017). Special Issue on 2015 Workshop on Data Intensive Scalable Computing Systems (DISCS-2015), pp. 83–95. URL: https://www.sciencedirect.com/science/article/pii/S0167819116301065.

[95] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann.lecun. com/exdb/mnist/* (1998).

[96] David E. Rumelhart and James L. McClelland. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[97] Patricio Cerda, Gaël Varoquaux, and Balazs Kegl. "Similarity encoding for learning with dirty categorical variables". In: *Machine Learning* 107 (June 2018), pp. 1477–1494.

[98] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU)*. 2018. URL: https://arxiv.org/abs/1803.08375.

[99] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. URL: https://arxiv.org/abs/1412.6980.

[100] Marta Carbone and Luigi Rizzo. "Dummynet Revisited". In: *SIGCOMM Comput. Commun. Rev.* 40.2 (Apr. 2010), pp. 12–20. URL: https://doi.org/10.1145/1764873.1764876.