République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de -Jijel-Mohamed Sadik Benyahia
Faculté des Sciences Exactes et informatique
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme Master de Recherche en Informatique

Option : *Réseaux et Sécurité*

Thème

Les algorithmes de graphes parallèles : algorithme et implémentation sous Graphx

Présenté par :
Boufenar Yousra
Boufenar Aicha

Encadré par :
Melle :BrighenAssia

Promotion : 2021.

⁕ Remerciements ⁕

⁕ *En préambule à ce mémoire, je remercie Dieu le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce Modeste travail.*

⁕ *Ensuite, je tiens à remercier mon encadreur Brighten Assia pour la confiance qu'il m'a accordé en acceptant d'encadrer ce travail, pour ses multiples conseils et son attention permanente sur l'évolution de mon travail ;*

⁕ *Merci aussi pour toutes les relectures, suggestions et commentaires, qui m'ont permis d'améliorer la qualité de cet mémoire*
⁕ *Aussi, je joins ces remerciements èglement aux membres du jury, pour avoir accepté de juger ce mémoire ;*

;

⁕ *A Enfin, je remercie tous les enseignats qui ont été à la base de mon formation.*

⁂ *Dédicaces* ⁂

Apreès cette réussite que fait la joie à tous qui m 'aime. Je dédie ce modeste travail :

⁂ Au meilleur des pères A ma très chère maman Qu'ils trouvent en moi la source de leur fierté ;

⁂ Mon frère Othman A qui je souhaite un avenir radieux plein de réussite ⁂ A qui je dois tout A ma soeur Hala et Yousra et Amira ;

;

⁂ A mes Amis A tous ceux qui me sont chers .

Boufenar Aicha

⁂ *Dédicaces* ⁂

Apreès cette réussite que fait la joie à tous qui m 'aime. Je dédie ce modeste travail :

⁂ A mon Père L' ma mére, vous êtes pour moi une source de vie car sans vos sacrifices, votre tendresse et votre affectation je ne pourrais arriver jusqu'au bout. Je me réjouis de cet amour filal .Que dieu vous garde afin que votre regard puisse Suivre ma destinée.

;

⁂ A soeur Hala et Amira et Aicha et à mon frère Othman qui ont été toujours présent pour moi. ;

⁂ A mes amis et collègues ;

⁂ A tous ceux qui sont chères, proches de mon caeur, et à tous ceux qui m'aiment et qui aurait voulu partager ma joie… ;

.

Boufenar Yousra

Table des matières

Liste des tableaux

Table des figures

List of Algorithms

Liste des abréviations

SaaS **S**oftware **A**s **A** **S**ervice

PaaS **P**latform **A**s **A** **S**ervice

IAAS **I**nfrastructure **A**s **A** **S**ervice

HADOOP **H**igh **A**vailability **D**istributed **O**bjet **O**riented **P**latform

HDFS **H**adoop **D**istributed **F**ile **S**ystem

GPF **G**raph **P**rocessing **F**ramework

BFS **B**readth**F**irst **S**earch

DFS **D**epth **F**irst **S**earch

SCC **S**trongly **C**onnected **C**omponents

API **A**pplication **P**rogramming **I**nterface

SVD++ **S**ingular **V**alue **D**ecomposition

RDD **R**esilient **D**istributed **D**ataset

CPU **C**entral **P**rocessing **U**nit

BSP **B**ulk **S**ynchronous **P**arallel

CPSG **C**oloration **P**arallèle **S**ommets **G**raphe

RLF **R**ecursive **L**argest **F**irst

YARN **Y**et **A**nother **R**esource **N**egotiator

SQL **S**tructure **Q**uery **L**anguage

S3 **S**imple **S**torange **S**ervice

CRM **C**ustomer **R**elationship **M**anagement

SI **S**ystème **I**nformation

RIP **R**outing **I**nformation **P**rotocol

AWS **A**mazon **W**eb **S**ervice

Amazon EC2 **A**mazon **E**lastic **C**ompute **C**loud

SISD **S**ingle **I**nstruction **S**ingel **D**ata

SIMD **S**ingle **I**nstruction **M**ultiple **D**ata

MISD **M**ultiple **I**nstruction **S**ingle **D**ata

MIMD **M**ultiple **I**nstruction **M**ultiple **D**ata

EPFL **É**cole **P**olytechnique **F**édérale de **L**ausanne

JDK **J**ava **D**evelopment **K**it

MLlib **M**achine **L**earning **Lib**rary

ETL **E**xtract **T**ransform and **L**oad

VPN **V**irtual **P**rivate **N**etwork

LAN **L**acal **A**rea **N**etwork

ERP **E**nterprise **R**esource **P**lanning

IA **I**ntelligence **A**rtificielle

BI **B**usiness **I**ntelligence

MPP **M**assively **P**arallel **P**rocessing

Introduction générale

L'avènement d'Internet et une popularité croissante des technologies de mobile et de capteurs ont entraîné une explosion de données dans les systèmes et le monde Web. Cette explosion de données a posé plusieurs défis aux systèmes utilisés traditionnellement pour le stockage et le traitement des données. En fait, les défis sont si graves qu'il ne serait pas faux de dire que les systèmes traditionnels ne peuvent plus remplir les besoins croissants de l'informatique intensive de données.

Les moteurs de base de données basés sur le standard SQL et créés dans les années 1970 ont de bonnes performances lors du traitement de petites quantités de données relationnelles, mais ces outils sont très limités face à l'expansion des données en volume et en complexité. Dans un autre côté, MPP (le traitement massivement parallèle) créé initialement au début des années 1980 a amélioré légèrement les indicateurs de performance pour les volumes de données complexes. Cependant, ce traitement n'a pas pu être utilisé pour le traitement des données non-relationnelles à expansion permanente. Des outils puissants sont requis pour stocker et exploiter ces données en expansion quotidienne dans le but de soumettre un traitement simple et fiable des données récoltées des utilisateurs. Des résultats rapides et de bonne qualité sont attendus. Pour les industriels et les décideurs en général, ces résultats sont aussi importants que les plus lourds investissements métier [26].

Les opérateurs de modélisation traditionnels sont confrontés à leurs limitations dans ce défi, puisque les informations se multiplient en volume et complexité, une chose qui actuellement ne peut être gérée que par des techniques de modélisation non-relationnelles. Dans ce contexte, Hadoop MapReduce est considéré comme la technique de traitement la plus efficace, comparée aux bases de données SQL et au traitement MPP [26]. Hadoop dispose d'une performance proportionnelle à la complexité des données volumineuses. C'est un outil efficace pour résoudre les problèmes de données massives mais c'est aussi un concept qui a changé l'organisation des systèmes de traitement en large échelle. Cependant malgré le succès qu'il a eu, ce modèle n'a pas encore atteint son aspect final en tant que solution informatique mature. Au contraire, il s'agit d'un point de départ vers d'autres perspectives.

Ces dernières années ont connu un regain d'intérêt pour l'utilisation des graphes comme moyen fiable de représentation et de modélisation des données, et ce dans divers domaines de l'informatique. En particulier, pour les grandes masses de donnes, les graphes apparaissent comme une alternative prometteuse aux bases de données relationnelles. Par ailleurs, les graphes sont des modèles puissants capables à la fois de capturer les différentes relations

entre les données, leur donner une meilleure représentation et de faciliter leur exploration. Les réseaux sociaux comme Facebook ou Twitter manipulent les données qui s'accordent naturelle ment avec une représentation sous forme de graphe. Cependant, les graphes obtenus de ces grandes masses de données sont de grands graphes, d'où la naissance du terme «Big Graph ». Leur exploration nécessite donc des outils nouveaux et efficaces. Ceci pose de nouveaux problèmes de recherche en graphes. Et parce que la théorie des graphes évolue depuis déjà quelques siècles, c'est la raison pour laquelle plusieurs frameworks de traitement et d' analyse de graphes déjà existent, encore d'autres qui sont en cours de développement. Parmi ces nouveaux frameworks on peut citer les plus célèbres qui sont Pregel [50], Giraph [49] , et Graphx [49].

Le traitement de larges graphes toujours été et reste un challenge. Les problèmes commencent avec le traitement de graphes, de l'ordre du milliard de sommets fortement connectés. Un tel graphe ne peut être traité en une fois sur une seule machine. Par Conséquence, les algorithmes de traitement des graphes ne sont plus efficaces, et le temps de calcul peut vite exploser exponentiellement sur ces grands graphes. Dans ce mémoire nous allons exploiter les avantages des systèmes parallèle de traitement de données de graphes, en exprimant efficacement le calcul du graphe dans la structure parallèle. Nous tirons parti de nouvelles idées dans la représentation des graphes distribuée pour distribuer efficacement des graphes en tant que structures de données tabulaires. Pour ce faire, nous avons choisi de travailler avec Spark Graphx où nous exploitons des avancées dans les systèmes de flux de données pour exploiter le calcul en mémoire et la tolérance aux pannes.

Problématique et objectifs du travail

Graphx est un outil puissant de traitement parallèle de graphes. Il est libre, open source et délivré avec une bibliothèque qui contient un ensemble d'algorithmes de graphes prédéfinis bien testés et prêtent à s'exécuter sur des larges graphes tels que :

- PageRank.
- Connected components.
- Label propagation.
- SVD++(Singular Value Decomposition).
- Strongly connected components.
- Triangle count.
- Single-Source-Shortest-Paths.
- Community Detection.

La bibliothèque Graphx ne couvre pas tous les algorithmes de graphes qui se trouvent dans la itterature. Dans ce travail, nous allons proposer et implémenter un nouvel algorithme de graphes sous Graphx. L'algorithme à proposer est un algorithme de coloration des sommets d'un graphe.

Organisation du mémoire

Nous avons organise notre mémoire en quatre Chapitre :

- ✍ Le premier chapitre consiste en quelques concepts de base pour le cloud computing, les grandes données et les graphes. Également dans ce chapitre, nous avons essayé de clarifier certains outils de traitement parallèles sur les graphes.
- ✓ Le deuxième chapitre est consacré à l'affichage de certains frameworks (Pregel, Giraph et GraphX). Au cours de ce chapitre, nous allons nous concentrer sur le frameworks Graphx.
- ✓ Dans le troisième chapitre nous allons définir le problème de coloration des sommets d'un graphes, ainsi nous allons proposer un algorithme de graphes parallèles et l'adapter sous Graphx pour colorer un graphe avec le minimum de couleurs possible.
- ✓ Le quatrième chapitre est dédié à la mise-en-ouvre et l'évaluation de l'algorithme. Nous décrivons tout d'abord l'environnement de travail et les étapes d'installation des outils logiciels et matériels nécessaires, les benchmarks de données utilisés, avant de se focaliser sur l'application de l'algorithme implémenté et la validation des résultats obtenus en termes de scalabilité, durée d'exécution, et la qualité de la solution.

Enfin nous terminerons par une conclusion générale, dans laquelle on va synthétiser tous les résultats obtenus et les renseignements acquis durant la réalisation de ce travail.

Chapitre 1

Notions introductifs et concepts de base

1.1 Introduction

Les Big Data se caractérisent par leur volumétrie (données massives), ils sont connus aussi par leur variété en termes de formats et de nouvelles structures, ainsi, qu'une exigence en termes de rapidité dans le traitement. Mais jusqu'à maintenant d'après les recherches, aucun logiciel est encore capable de gérer toutes ces données qui ont plusieurs types et formes et qui augmentent très rapidement. Alors les problématiques du Big Data font partie de notre quotidien, et il faudrait des solutions plus avancées pour gérer cette masse de données dans un petit temps.

Dans ce chapitre nous allons présenter les différents concepts liés à ce nouveau paradigme informatique, ensuite on verra l'évolution cloud, leurs des différents modèles, le déploiement du cloud et ses caractéristiques, ainsi leurs compatibilités avec le Big Data ainsi en suite nous allons parler d'algorithmes parallèles et de certains concepts associés aux graphes. On va aussi présenter des notions liées au big graphes et les technoogies de traitement et de stockage de Big Data.

1.2 Graphes

Les graphes sont des structures largement utilisées qui aident à représenter des réseaux constitués de noeuds et de leurs interconnexions appelées arcs. Ils pourraient être exploités pour décrire des chemins dans une ville, des réseaux de circuits comme ceux du téléphone et des ordinateurs ou même des réseaux sociaux, où chaque noeuds est une structure et contient des informations telles que le nom de la personne, la date de naissance et l'adresse. Il devient donc évident que les données utilisées dans un large éventail d'applications peuvent être intuitivement formulées dans un graphe, offrant une vue holistique des corrélations aux quelles une entité participe et s'étendant à la fois aux domaines de la visualisation et de l'analyse[15].

Big graphes sont omniprésents, allant des réseaux sociaux et des réseaux d'appels mobiles vers des réseaux biologiques et le world wide web. Des exemple de graphes à grande échelle

Name	Nodes	Edges
Web Graph	More than 20 billion nodes (pages)	More than 160 billion edges (hyperlinks)
Facebook	More than a billion nodes (users)	More than 140 billion edges (friendship relationships)
LinkedIn	Almost 8 million nodes	Almost 60 million edges
SemanticWeb	3.7 million nodes (objects)	400 million edges (facts)

TABLE 1.1 – Graphes à grande échelle dans la littérature actuelle [58]

réelle dans le monde comprennent [58] :

- Graphes sociaux (Facebook, twitter, google+, LinkedIn, etc.)
- Graphes d'endossement (graphe de liaison web, graphe de la citation papier, etc.)
- Graphes de localisation (carte, réseau électrique, réseau téléphonique, etc.)

1.2.1 Defintions et concepts liés aux graphes

Définition 1.2.1. Un graphe fini $G = (V, E)$ est défini par l'ensemble fini$V = v_1, v_2, ..., v_n$ dont les éléments sont appelés sommets (Vertices en anglais), et par l'ensemble fini $E = e_1, e_2, ..., e_m$ dont les éléments sont appelés arêtes (Edges en anglais)[53].

1.2.1.1 Graphes orientés

Définition 1.2.2. [53] On dit qu'un graphe est orienté lorsque ses arêtes (appelées arcs) ou seulement une partie de celles ci ne peuvent être parcourues que dans un sens. Si un arc part du sommet A pour arriver à B, on dit que A est l'origine de l'arc et B est son extrémité. L'origine et l'extrémité d'un arc permettent de le nommer (exemple : (A,B)).

Les arcs sont représentés par des flèches. Si dans un graphe orienté une arête peut être parcourue dans les deux sens, il est fréquent de tracer deux flèches, du moins lorsque les arcs sont peu nombreux. Toutefois les problèmes concrets peuvent nécessiter des graphes très complexes et les flèches doubles évitent alors d'amplifier la confusion.

Exemple 1.2.1. le figure 1.1 presente un exmple d' un graphe orienté. Soit le graphe $G = (V, E)$ où $V = a, b, c, d, e$ *et* $E = (a, b), (a, e), (b, b), (b, c), (c, c), (c, d), (c, e), (d, a), (e, a), (e, d)$.

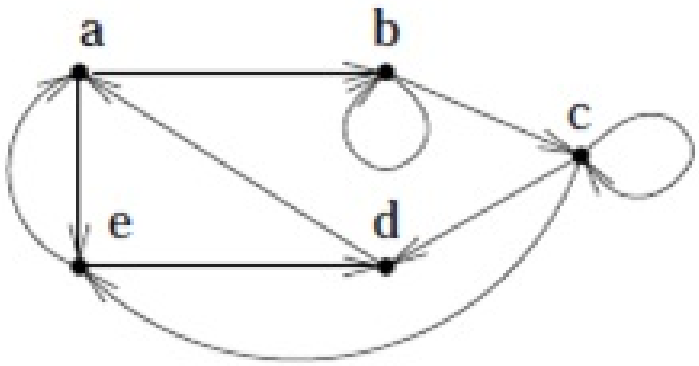


FIGURE 1.1 – Exemple de graphes orientés[53]

1.2.1.2 Graphes non orientés

Définition 1.2.3. On dit qu'un graphe est non orienté lorsque une arête e de l'ensemble E est définie par une paire non ordonnée de sommets, appelés les extrémités de e. Si l'arête e relie les sommets a et b, on dira que ces sommets sont adjacents, ou incidents avec e, ou bien que l'arête e est incidente avec les sommets a et b. On appelle ordre d'un graphe le nombre de sommets n de ce graphe[53].

Exemple 1.2.2. Le Figure 1.2 presente un example d'un graphe non orienté.

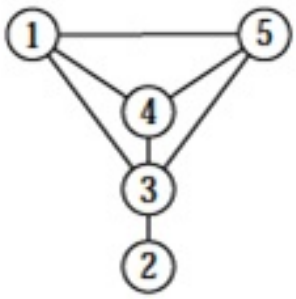
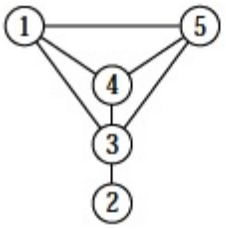


FIGURE 1.2 – Exemple d'un graphe non orientés [53].

1.2.1.3 Représentations de graphes

1. **Matrice d'adjacences :**
   On peut représenter un graphe simple par une matrice d'adjacences. Une matrice (n× m) est un tableau de n lignes et m colonnes. L'element (i, j) de la matrice désigne l'intersection de la ligne i et dela colonne j . Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un ≪1 ≫à la position (i, j) signifie que le sommet i est adjacent au sommet j [53] .

Exemple 1.2.3. Exemple : Un exmple de representer d'un graphe par une matrice d'adjacences est illustré par le Figure 1.3 :



$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

FIGURE 1.3 – Exemple de matrice d'adjacence d'un graphe [53].

2. **Listes d'adjacences :** On peut aussi représenter un graphe simple en donnant pour chacun de ses sommets la liste des sommets auxquels il est adjacent. Ce sont les listes d'adjacences(par exemple Figure1.4)[53].

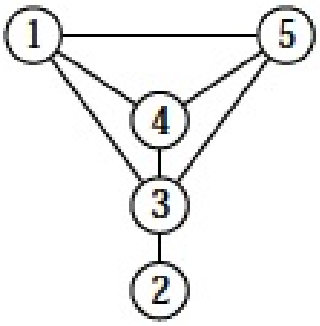


FIGURE 1.4 – Exemple de listes d'adjacences [53].

1.2.2 Exemples de modélisation par des graphes

- **Réseaux de transport :**un réseau de transport (routier, ferroviaire, métro, etc) peut être représenté par un graphe dont les sommets sont des lieux (intersections de rues, gares, stations de métro, etc) et les arcs indiquent la possibilité d'aller d'un lieu à un autre (par un tronçon de route, une ligne de train ou de métro, etc). Ces arcs peuvent être valués, par exemple, par leur longueur, la durée estimée pour les traverser, ou encore un coût.
- **Réseaux sociaux :** les réseaux sociaux (LinkedIn, Facebook, etc) peuvent être représentés par des graphes dont les sommets sont des personnes et les arêtes des relations entre ces personnes.
- **Planification :** certains problèmes peuvent être spécifiés par un état initial, un état final, un certain nombre d'états intermédiaires et des règles de transition précisant comment on peut passer d'un état à l'autre. Résoudre le problème consiste alors à trouver une suite de transitions permettant de passer de l'état initial à l'état final.

1.2.3 Type de graphes

Il y a plusieurs type spécifiques de graphes [15] :

- **Graphe isolé :** le graphe isolé d'ordre n, est un graphe a n sommets sans arête/arc.
- **Graphe cyclique :** le graphe cyclique d'ordre n, est un graphe simple a n sommets $x_1, .., x_n$ tels que le sommet x_i soit voisin uniquement avec $xi - 1$ et $xi + 1/4$: $E = xi, xi + 1$ *pour* $i\epsilon[1, n-1] \cup x_n, x_1$.
- **Graphe complet :** un graphe est dit complet si tous ses sommets sont adjacents. Pour un graphe simple G =(X, E) d'ordre n, le graphe complet et on a : $d(x) = n - 1$ *et* $m = |E| = n(n-1)/2$.
- **Graphe biparti :** un graphe simple G = (X, E) est dit biparti si l'ensemble X peut être partitionner en deux sous-ensembles X_1 et X_2 de sorte que les sommets d'un même sous ensemble soient non adjacents.
- **Graphe régulier :** un graphe est dit régulier si tous ses sommets sont de même degré. On le note k-régulier si chaque sommet est de degré k.

1.2.4 Problèmes fondamentaux de la théorie des graphes

- **problème de plus court chemin :** le problème de plus court chemin est un problème qui consiste à trouver un chemin d'un sommet à un autre de façon que la somme des poids des arcs de ce chemin soit minimale.
- **PageRank :** le PageRank ou PR est l'algorithme d'analyse des liens concourant au système de classement des pages Web utilisé par le moteur de recherche Google. Il mesure quantitativement la popularité d'une page web. Le PageRank n'est qu'un indicateur parmi d'autres dans l'algorithme qui permet de classer les pages du Web dans les résultats de recherche de Google. Ce système a été inventé par Larry Page, cofondateur de Google [57].
- **Composante connexe :** composante connexe est un sous-graphe induit maximal connexe, c'est-à-dire un ensemble de points qui sont reliés deux à deux par un chemin. On peut ainsi regrouper les sommets d'un graphe selon leur appartenance à la même composante connexe [25].
- **Coloration de graphe :** en théorie des graphes, la coloration de graphe consiste à attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente. On cherche souvent à utiliser le nombre minimal de couleurs, appelé nombre chromatique. La coloration fractionnaire consiste à chercher non plus une mais plusieurs couleurs par sommet et en associant des coûts à chacune [72].

1.3 Algorithme parallèle

Un algorithme parallèle est conçu pour s'exécuter sur une machine parallèle pour résoudre un problème en améliorant le temps de calcul tout en respectant les contraintes de complexité en temps de calcul et en mémoire. La conception d'un algorithme parallèle pour un problème donné est beaucoup plus complexe que la conception d'un algorithme séquentiel du même problème, car elle demande la prise en compte de plusieurs facteurs tels que : la partie du programme qui peut être traitée en parallèle, la manière de distribuer les données, les dépendances des données, la répartition de charges entre les processeurs, les synchronisations entre les processeurs. Il y a essentiellement deux méthodes pour concevoir un algorithme parallèle, l'une consiste à détecter et à exploiter le parallélisme à l'intérieur d'un algorithme séquentiel déjà existant, l'autre consistant à inventer un nouvel algorithme dédié au problème donné [46].

1.3.1 Machines Parallèles

Les machines parallèles sont construites autour d'un ensemble de noeuds interconnectés, chacun est constitué d'un ou plusieurs processeurs identiques, ou non, qui collaborent pour l'exécution d'une application. Une classification de ces machines à été introduite par M. J. Flynn (1966) [46].

- **SISD (Single Instruction, Singel Data) :** c'est le modèle classique de Von Neumann où un seul processeur exécute une seule instruction sur une seule donnée (un seul flot de données) à la fois résidant dans une seule mémoire.
- **SIMD (Single Instruction, Multiple Data) :** le contrôle est centralisé sur un seul processeur, les autres processeurs sont synchronisés entre chaque instruction du même algorithme qu'ils exécutent.
- **MISD (Multiple Instruction, Single Data) :** une même donnée est traitée simultanément par plusieurs processeurs.
- **MIMD (Multiple Instruction, Multiple Data) :** le contrôle est réparti sur tous les processeurs et la synchronisation, lorsqu'elle est nécessaire, est réalisée par communication.

1.3.2 Machines parallèles à mémoire partagée

Ces machines sont caractérisées par une horloge indépendante pour chaque processeur, mais une seule mémoire partagée entre ces processeurs, où tous les processeurs lisent et écrivent dans le même espace d'adressage mémoire, ce qui permet de réaliser un parallélisme de données et de contrôles (voir Figure 1.5). Le programmeur n'a pas besoin de spécifier l'emplacement des données, il définit seulement la partie du programme que doit exécuter chaque processeur en plus de la gestion de la synchronisation. Ce type d'architecture possède plusieurs avantages dont la simplicité, la scalabilité et la parallélisation de haut niveau.

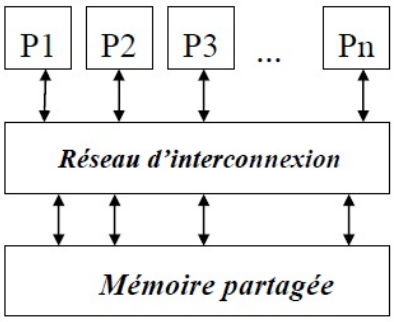


FIGURE 1.5 – Machines parallèles à mémoire partagée [46].

Son inconvénient majeur est lié principalement à la limite de la bande passante du réseau d'interconnexion[46].

1.3.3 Machines parallèles à mémoire distribuée

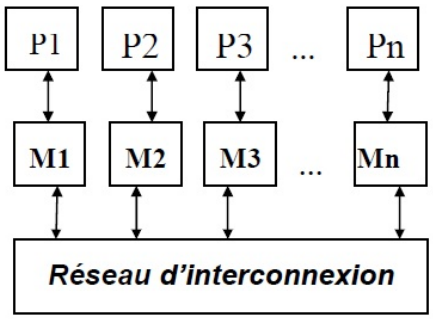


FIGURE 1.6 – Machines parallèles à mémoire distribuée [46].

Dans ce type de machine, chaque processeur possède sa propre mémoire locale, où il exécute des instructions identiques ou non aux autres processeurs. Les différents noeuds définis par l'ensemble de mémoires et de processeurs sont reliés entre eux par un réseau d'interconnexion (voir Figure 1.6). Le parallélisme est implémenté par échange de messages. L'avantage principal des machines parallèles à mémoire distribuée est l'augmentation facile du nombre de processeurs avec des moyens simples, tels que les clusters. Seulement elles présentent plus de difficulté dans la programmation[46].

1.3.4 Algorithme distribué

Définition 1.3.1. Un algorithme distribué A, sur un système distribué S, est la suite de transitions locales à effectuer sur chaque entité de S. Les algorithmes distribués se différencient par le mode de communication selon lequel interagissent les différents éléments distribués (mémoire partagée, échange de messages...), par le modèle temporel suivi (synchrone, asynchrone) et par la nature du réseau implémenté (fiable, non fiable, anonyme...) [7].

1.4 Cloud Computing

Définition 1.4.1. Le Cloud Computing est un néologisme utilisé pour décrire l'association d'Internet (cloud , le nuage) et l'utilisation de l'informatique(computing)[24]. Le terme nuage fait référence à un réseau ou à Internet. En d'autres termes, on peut dire que ce nuage est quelque chose qui est présent à un emplacement éloigné. Le cloud peut fournir des services sur réseau, c'est à dire sur des réseaux publics ou sur des réseaux privés, (LAN ou VPN). Des applications telles que l'Email, la conférence Web, la gestion de la relation client (CRM), tous sont exécutés dans le cloud. Le Cloud Computing fait référence à la manipulation, à la configuration et à l'accès aux applications en ligne. Il propose un stockage en ligne de données, infrastructure et application (Figure1.7) [71] .

FIGURE 1.7 – Utilisation du Cloud Computing [24].

Nous n'avons pas besoin d'installer un logiciel sur notre PC local et c'est ainsi que le Cloud Computing surmonte les problèmes de dépendance de la plateforme [71].

1.4.1 Modèles de service

Les modèles de service sont les modèles de référence sur lesquels repose le Cloud Computing. Ceux ci peuvent être classés en trois modèles de service de base énumérés ci dessous (voir Figure1.8) [71] :

- **IaaS (Infrastructure as a Service) :** concerne les serveurs, moyens de stockage, réseau, le modèle IaaS consiste à pouvoir disposer d'une infrastructure informatique hébergée. Une entreprise pourra par exemple louer des serveurs Linux, Windows ou autres systèmes, qui tourneront en fait dans une machine virtuelle chez le fournisseur de l'IaaS. Ainsi l'accès à la ressource est complet et sans restriction, équivalent de fait à la mise à disposition d'une infrastructure physique réelle. Par exemples : Amazon EC2(Elastic Compute Cloud), GoGrid, Sun Grid .

- **PaaS (Platform as a Service) :** concerne les environnements middleware, de développement, de test, le modèle PaaS consiste à mettre à disposition un environnement prêt à l'emploi, fonctionnel et performant, y compris en production ; l'infrastructure hébergée étant totalement transparente. Par exemple une plateforme PaaS peut être un environnement de développement et de test. Par exemples :Force.com ,Google App Engine ,Azure Services Platform.
- **SaaS (Software as a Service) :** le terme SaaS évoque bien un service dans le sens où le fournisseur vend une fonction opérationnelle, et non des composants techniques requérant une compétence informatique pour l'utilisateur. Concernant les applications d'entreprise : CRM(Customer Relationship Management), outils collaboratifs, messagerie, BI, ERP,... le modèle SaaS permet de déporter une application chez un tiers. Ce modèle convient à certaines catégories d'applications qui se doivent d'être globalement identiques pour tout le monde, la standardisation étant un des principes du cloud. Par exemples :CRM Salesforce.com, Google Docs, Adobe Photoshop Express Online.

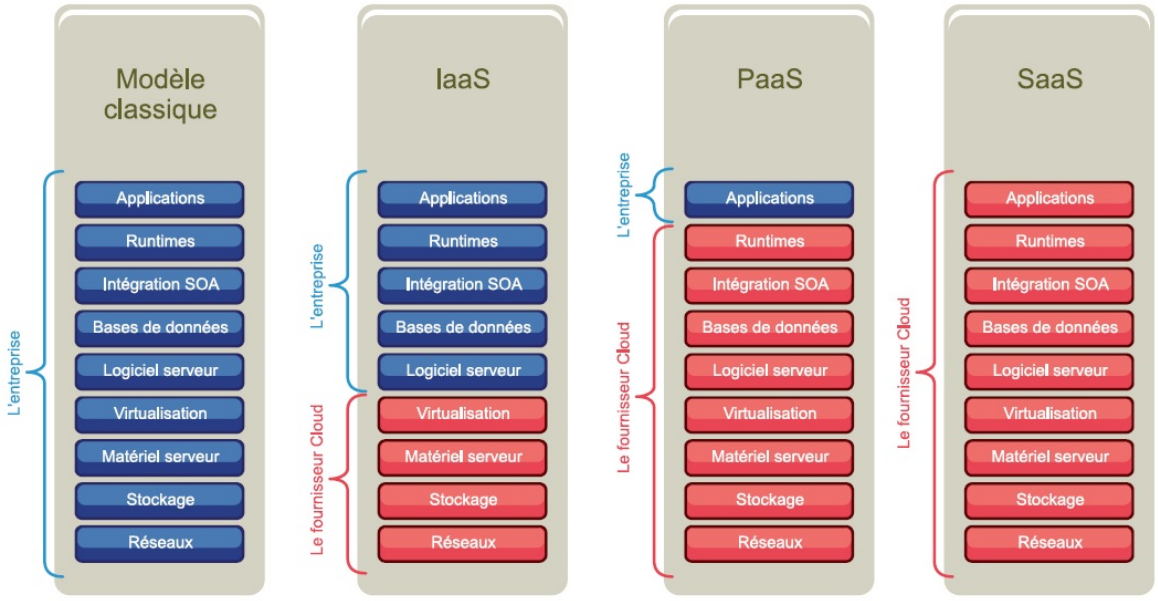


FIGURE 1.8 – Modèles de service du Cloud Computing [71]

1.4.2 Modèles de déploiement du cloud computing

Quatre grandes familles de cloud existent en fonction du mode de déploiement (voir Figure1.9)[71] :

- **Le cloud privé :** est réservé pour un usage interne. Soit la solution cloud est interne, dans ce cas, l'entreprise est propriétaire. Soit la solution cloud est basée sur une location entre l'entreprise et le fournisseur.
- **Le cloud public :** est réservé pour un usage externe. La solution est fournie à toute catégorie d'utilisateur. L'infrastructure quant à elle reste à la charge de l'entreprise

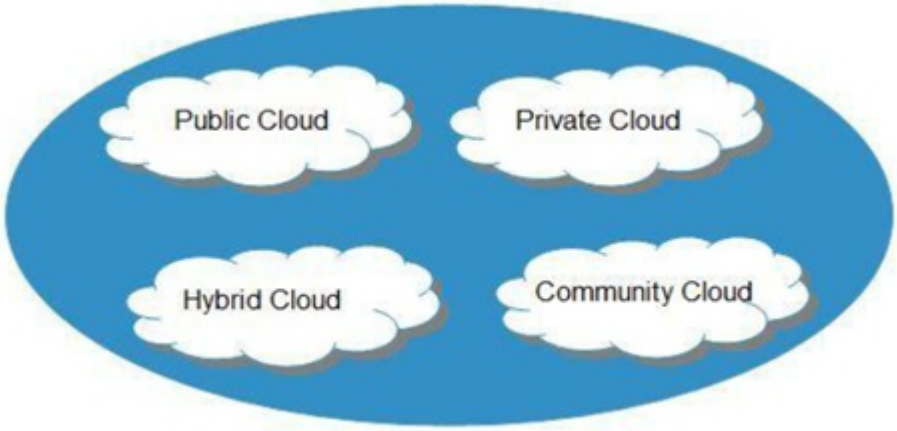


FIGURE 1.9 – Modèles de déploiement du Cloud Computing [71].

qui propose les services.

- **Le cloud communautaire :** est une solution cloud partagée entre plusieurs organisations d'une même communauté (un etat par exemple). Cette solution peut être gérée en interne ou bien en externe, via un tiers.
- **cloud hybride :** Un cloud hybride mélange deux ou plusieurs types d'environnements cloud. Les déploiements de cloud hybride combinent les cloud publics et privés, et ils peuvent également inclure des infrastructures héritées sur site. Pour qu'un cloud soit réellement hybride, ces différents environnements doivent être étroitement interconnectés entre eux, fonctionnant essentiellement comme une infrastructure combinée. Presque tous les cloud hybrides comprennent au moins un cloud public.

1.4.3 Caractéristiques du Cloud Computing

Les Services Cloud Computing ont des caractéristiques qui les distinguent des autres technologies [41] :

- En général, les utilisateurs du Cloud Computing ne sont pas propriétaires des ressources informatiques qu'ils utilisent. Les serveurs qu'ils exploitent sont hébergés dans des data centres externes.
- Les services sont fournis selon le modèle pay-per-use ou le modèle d'abonnement.
- Les ressources et les services fournis au client sont souvent virtuels et partagés par plusieurs utilisateurs.
- Les services sont fournis via l'internet.

Ces spécificités font de la technologie Cloud Computing une nouvelle option qui offre à ses utilisateurs la possibilité d'accès à des logiciels et à des ressources informatiques avec la flexibilité et la modularité souhaitées et à des coûts très compétitifs.

1.4.4 Concepts liés aux cloud computing

1.4.4.1 Datacenter

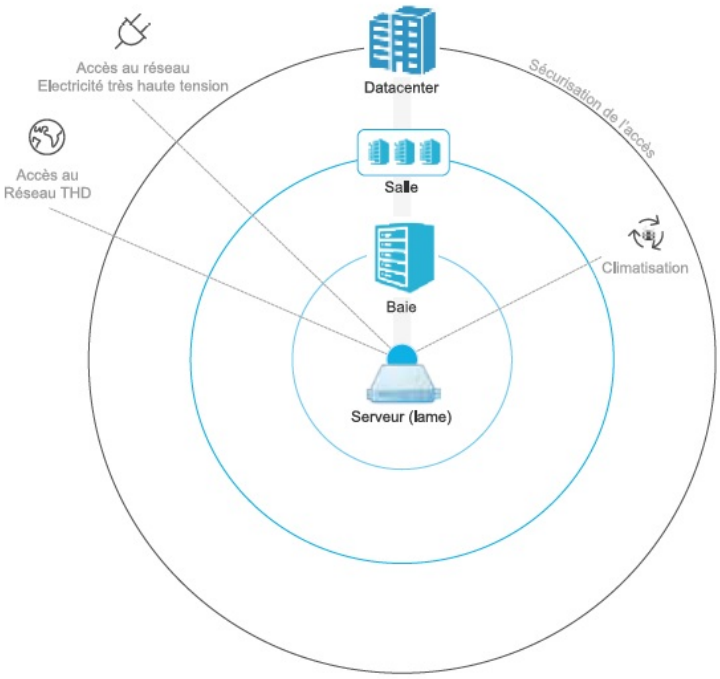


FIGURE 1.10 – Éléments constitutifs d'un Datacenter.

Un datacenter est une infrastructure immobilière et technique destinée à l'hébergement d'une concentration importante d'équipements informatiques. Il est composé de salles sécurisées pour accueillir les équipements informatiques et d'infrastructures techniques assurant la continuité de l'alimentation électrique, du refroidissement des serveurs et de l'accès au réseau à très haut débit pour l'ensemble de ces ressources et de points d'accès aux réseaux électriques à haute tension et aux réseaux de télécommunication très haut débit, et d'un bâtiment spécialisé et sécurisé intégrant l'ensemble de ces composants. Un Datacenter constitue ainsi un site sécurisé pouvant héberger des services numériques nécessitant une haute disponibilité. Un datacenter doit être opéré en tenant compte de quatre enjeux critiques(voir Figure1.10) :

- La continuité de l'alimentation électrique.
- La maitrise des systèmes de refroidissement.
- La maîtrise des accès aux réseaux.
- La sécurisation du site.

1.4.4.2 Cluster

À un niveau élevé, un cluster d'ordinateurs est un groupe d'au moins deux ordinateurs, ou noeuds, qui s'exécutent en parallèle pour atteindre un objectif commun. Cela permet aux charges de travail constituées d'un nombre élevé de tâches individuelles parallélisables

d'être réparties entre les noeud du cluster. Pour créer un cluster d'ordinateurs, les noeuds individuels doivent être connectés dans un réseau pour permettre la communication entre noeuds. Le logiciel de cluster d'ordinateurs peut ensuite être utilisé pour joindre les noeuds et former un cluster. Il peut avoir un périphérique de stockage partagé et un stockage local sur chaque noeud. Un utilisateur accédant au cluster ne devrait pas avoir besoin de savoir si le système est un cluster ou une machine individuelle. En outre, un cluster doit être conçu pour minimiser la latence et éviter les goulots d'étranglement dans la communication noeud à noeud [54]. Les clusters d'ordinateurs peuvent généralement être classés en trois types :

1. Hautement disponible ou basculement.
2. Calcul haute performance.
3. L'équilibrage de la charge.

1.4.4.3 Virtualisation

La virtualisation consiste en la création d'une ou plusieurs machines virtuelles à partir d'une machine physique telle qu'un serveur ou un ordinateur. Un logiciel nommé hyperviseur est alors chargé de faire le lien entre les propriétés de l'entité physique et celles de l'entité virtuelle. L'hyperviseur attribue des propriétés propres à chaque machine virtuelle afin de répondre au mieux aux besoins spécifiques du projet informatique.

1.4.4.4 Noeud

Un noeud du Cloud peut être une machine ou une machine virtuelle. Il est caractérise par un adresse propre. Il est constitué par un système de virtualisation adapté à l' infrastructure matérielle et qui gère le cycle de vie de plusieurs images logicielles.

1.5 Big data

Définition 1.5.1. A l'origine du concept de "BIG DATA" se trouve l'explosion du volume de données informatiques, conséquence de la flambée de l'usage d'Internet, au travers des réseaux sociaux, des appareils mobiles, des objets connectés, etc[27].

Définition 1.5.2. Littéralement , ces termes signifient méga données, grosses données ou encore données massives. Ils désignent un ensemble très volumineux de données qu'aucun outil classique de gestion de base de données ou de gestion de l'information ne peut vraiment travailler. En effet, on procrée environ 2,5 trillions d'octets de données tous les jours. Ce sont les informations provenant de partout : messages que nous nous envoyons, vidéos que nous publions, informations climatiques, signaux GPS, enregistrements transactionnels d'achats en ligne et bien d'autres encore. Ces données sont baptisées BIG DATA ou volumes massifs de données[27].

Définition 1.5.3. Selon le CXP , les BIG DATA désignent des méthodes et des technologies (pas seulement des outils) pour des environnements évolutifs (augmentation du volume de données, augmentation du nombre d'utilisateurs, augmentation de la complexité des analyses, disponibilité rapide des données) pour l'intégration, le stockage et l'analyse des données multistructurées (structurées, semi structurées et non structurées)[27].

Définition 1.5.4. Les BIG DATA est une démarche (ou un ensemble de technologies, d'architectures, d'outils et de procédures) qui consiste à collecter puis à traiter en temps réel des énormes volumes proviennent de sources, diverses structurées et non structurées, difficilement gérables avec des solutions classiques de stockage et de traitement[27].

1.5.1 Caractéristiques de BIG DATA

Le BIG DATA est une démarche un ensemble de technologies, d'architectures, d'outils et de procédures consistant à collecter puis à traiter en temps réel, ou presque, des données à la fois très nombreuses et très hétérogènes. Depuis les débuts du BIG DATA, 5V permettent de définir les attentes des utilisateurs [5] :

- **Volume :**les données dépassent les limites de la scalabilité verticale des outils classiques, nécessitant des solutions de stockage distribués et des outils de traitement parallèles.
- **Vélocité :** les données doivent être traitées et analysées rapidement eu égard à la vitesse de leur capture.
- **Variété :** les données sont hétérogènes ce qui rend leur intégration complexe et coûteuse.
- **Véracité :**la notion de véracité met en avant la dimension qualitative des données nécessaire au fonctionnement des outils de BIG DATA. Car au delà de l'intégrité des données, la véracité des données permet au directions métiers de mieux les utiliser au quotidien. Au final, ces 5 V permettent de définir le BIG DATA. Un outil qui permet de gérer des données qualitatives et volumineuses, très diverses, et traitées en temps réel. Surtout, un outil qui assure à l'utilisateur une visibilité des données, via des outils de reporting, qui lui permette de prendre de bonnes décisions(voir Figure 1.11) :
- **Variabilité :** le format et le sens des données peut varier au fil du temps.

1.5.2 Défis spécifiques du Big Data

Il y a plusieurs défis spécifiques pour les grandes données [67] :

- Manque de bonne compréhension du Big Data.
- Problèmes de croissance des données comment stocker correctement tous ces énormes ense confusion lors de la définition de l'outil Big Data.
- Les entreprises sont souvent confuses lors du choix du meilleur outil pour analyser et stocker des données volumineuses.

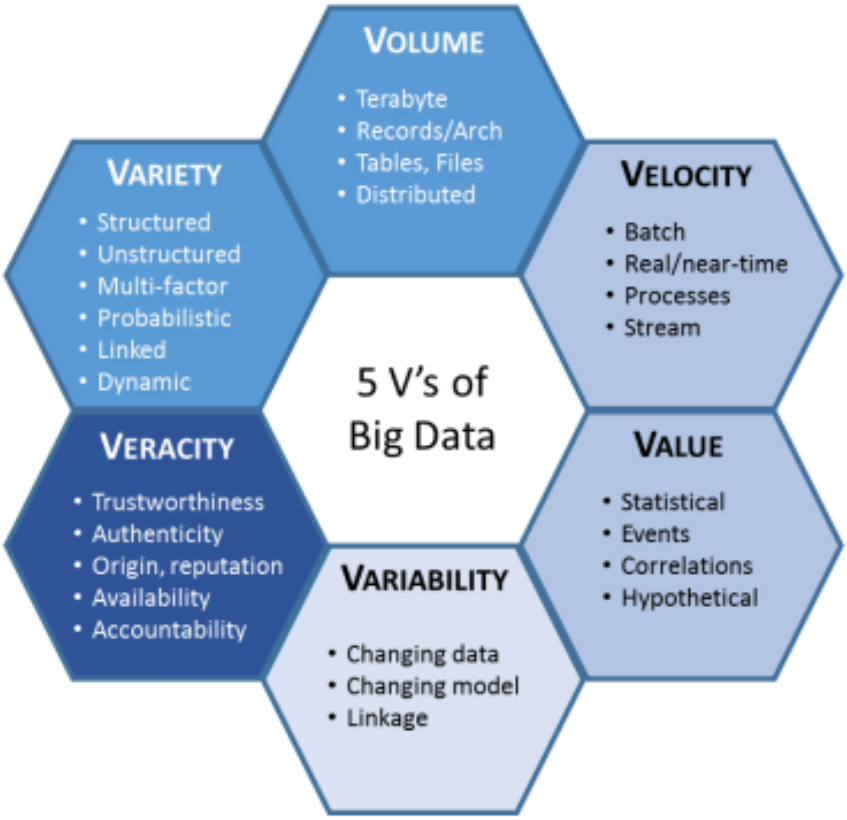


FIGURE 1.11 – Modèle de Big Data des cina v adapté à I'IMT [66] .

- Manque de professionnels des données parmi ces professionnels, scientifiques de données, analystes de données et ingénieurs de données qui ont une expérience de travail avec des outils et une connaissance des grands ensembles de données.
- La sécurité des données est l'un des défis majeurs du Big Data.

1.6 Technologies de traitement et de stockage de Big Data

Les Big Data requièrent de redéfinir les systèmes de stockage et de traitement de données, qui peuvent supporter ce volume de données. En effet, plusieurs technologies ont été proposées afin de représenter ces données, ces technologies prennent au moins un axe parmi les deux soit l'amélioration des capacités de stockage, soit l'amélioration de la puissance de calcul [75] :

- **Amélioration de la puissance de calcul :** le but de ces techniques est de permettre de faire le traitement sur un grand ensemble de données, avec un coût considérable et d'améliorer les performances de l'exécution comme le temps de traitement et la tolérance aux pannes. Avant l'apparition de la plateforme Hadoop on trouve plusieurs technologies comme le Could Computing, les architectures massivement parallèles MPP et les technologies In-Memory.

- **Amélioration des capacités de stockage :** l'amélioration du stockage vers des systèmes distribués, où un même fichier peut être réparti sur plusieurs disques durs, cela permet d'augmenter les volumes de stockage en utilisant un matériel de base. Ces technologies de stockage évoluent toujours pour offrir des accès plus rapides aux données comme le NoSQL, HDFS de la plateforme Hadoop, HBase, le Cloud Computing, etc.

1.6.1 MapReduce

Les systèmes d'entreprise traditionnels disposent normalement d'un serveur centralisé pour stocker et traiter les données. Le modèle traditionnel n'est certainement pas adapté au traitement de gros volumes de données évolutives et ne peut pas être géré par des serveurs de base de données standard. De plus, le système centralisé crée trop de goulot d'étranglementlors du traitement simultané de plusieurs fichiers. Google a résolu ce problème de goulot d'étranglement à l'aide de modèle MapReduce[75].

Définition 1.6.1. MapReduce a été conçu dans les années 2000 par les ingénieurs de Google. C'est un modèle de programmation conçu pour traiter plusieurs téraoctets de données sur des milliers de noeuds de calcul dans un cluster. MapReduce peut traiter des téraoctets et des pétaoctets de données plus rapidement et efficacement. Par conséquent, sa popularité a connu une croissance rapide pour diverses marques d'entreprises . Il fournit une plateforme très efficace pour l'exécution parallèle des applications, l'allocation de données dans des systèmes de bases de données distribuées, et communications réseau à tolérance de pannes. L'objectif principal de MapReduce est de faciliter la parallélisation des données, leur distribution et l'équilibrage de la charge dans une bibliothèque simple [75].

1.6.1.1 Architecture de modèle MapReduce

La Figure 1.12 montre l'architecture du paradigme MapReduce [1] :

L'architecture de MapReduce se compose de divers composants :

Une brève description de ces composants peut améliorer notre compréhension du fonctionnement de MapReduce.

- **Job :** il s'agit du travail réel qui doit être exécuté ou traité.
- **Tâche :** il s'agit d'un élément du Job réel qui doit être exécuté ou traité. Un job Mapreduce comprend de nombreuses petites tâches qui doivent être exécutées.
- **Job Tracker :** ce tracker joue le rôle de planification des travaux et de suivi de tous les travaux affectés au suivi des tâches.
- **Suivi des tâches :** ce suivi joue le rôle de suivi des tâches et de rapporter de l'état des tâches au suivi des travaux.
- **Données d'entrée :** il s'agit des données utilisées pour traiter dans la phase de mappage(mapping).

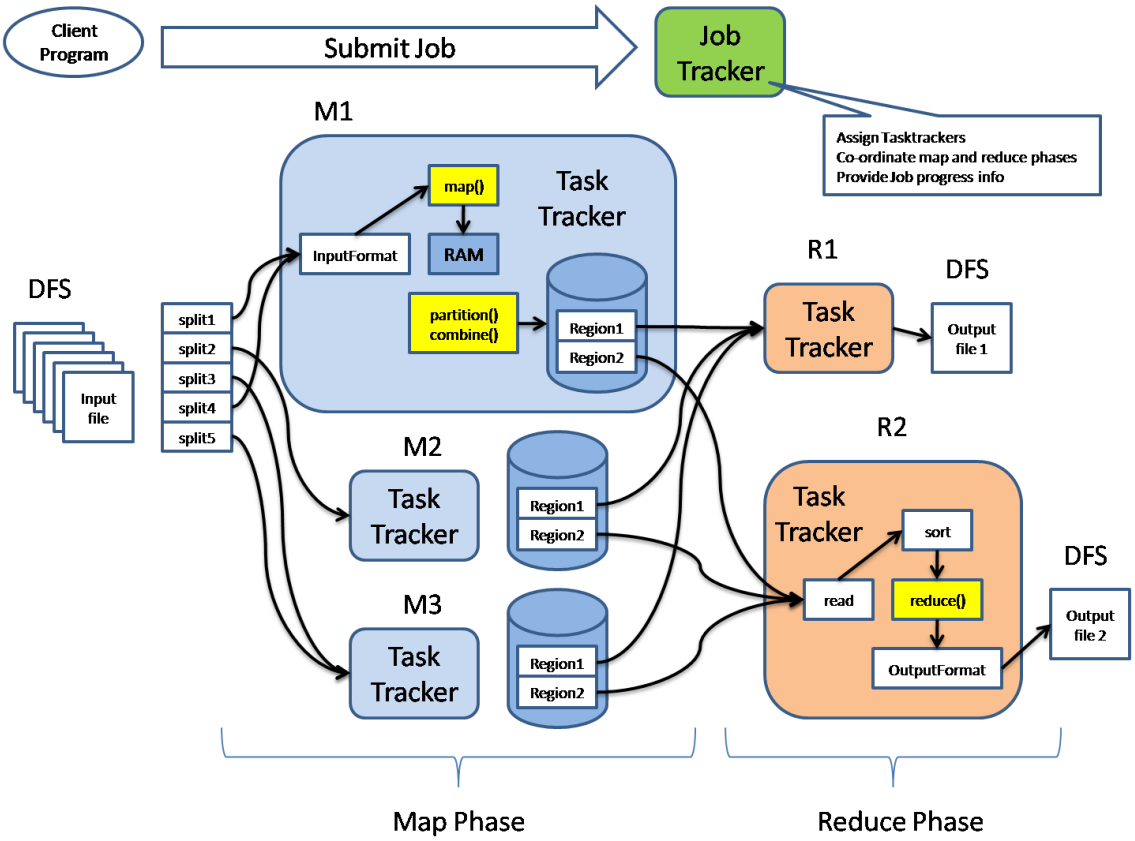


FIGURE 1.12 – Architecture MapReduce [1]

- **Données de sortie :** il s'agit du résultat du mappage (mapping) et de la réduction (reducing).
- **Client :** il s'agit d'un programme ou d'une interface de programmation d'application (API) qui soumet des travaux à MapReduce. MapReduce peut accepter les jobs des nombreux clients.
- **Tracker :** ce sont des sous-jobs qui résultent de la division du job principal.

Dans l'architecture MapReduce, les clients soumettent des jobs au master. Ce maître subdivisera ensuite le travail en sous-parties égales. Les pièces de travail seront utilisées dans les deux phases principales de MapReduce : Mapp et Reduce. Le développeur écrira une logique qui répond aux exigences de l'organisation ou de l'entreprise. Les données d'entrée seront divisées et mappées. Les données intermédiaires seront ensuite triées et fusionnées. Le reducteur qui générera une sortie finale stockée dans le HDFS traitera la sortie résultante. La figure 1.13 montre un organigramme simplifié pour un programme MapReduce :

FIGURE 1.13 – Phase d'execution pour un programme MapReduce

1.6.1.2 Applications de MapReduce

L'application MapReduce facilite l'exécution de nombreuses applications parallèles des données. MapReduce est le facteur principal dans de nombreuses applications importantes et peut améliorer le parallélisme du système. Il reçoit une attention considérable, pour les applications gourmandes en données et en temps de calculs sur des clusters de machines. Il est utilisé comme un outil de calcul distribué, efficace pour résoudre des problèmes variables, telsque la recherche, le regroupement, l'analyse de journaux, différents types d'opérations de jointure, la multiplication de matrices, la correspondance de modèles et l'analyse de réseaux sociaux. Il permet aux chercheurs d'étudier dans différents domaines. MapReduce est utilisé dans de nombreuses applications de Big Data tels que : l'exploration de messages courts, les algorithmes génétiques, k-means, algorithme de clustering, fragment d'ADN, système de transport intelligent, applications scientifiques dans le domaine de la santé, systèmes de classification à base de règles floues, environnements hétérogènes, la machine d'apprentissage extrême, forêt aléatoire, l'énergie proportionnelle, capteur mobile de données, web sémantique, etc[75].

1.6.1.3 Avantages de Mapreduce

- **Vitesse :** MapReduce peut traiter d'énormes données non structurées en peu de temps.
- **Tolérance à la faute :** le framework mapreduce peut gérer des échecs.
- **Costant efficace :** Hadoop a une fonction de sortie permettant aux utilisateurs de traiter ou de stocker des données de manière rentable.
- **Évolutivité :** Hadoop fournit un cadre hautement évolutif. MapReduce permet aux utilisateurs d'exécuter des applications à partir de nombreux noeuds.
- **Disponibilité des données :** des répliques de données sont envoyées à différents noeuds du réseau. Cela garantit que des copies des données sont disponibles en cas de panne.
- **Traitement parallèle :** dans MapReduce, plusieurs parties de travail du même jeu de données peuvent être traitées de manière parallèle. Cela réduit le temps nécessaire pour terminer une tâche.

1.6.2 Hadoop

Définition 1.6.2. Hadoop est un framework logiciel open source permettant de stocker des données, et de lancer des applications sur des clusters de machines standards. Cette solution offre un espace de stockage massif pour tous les types de données, une immense puissance de traitement et la possibilité de prendre en charge une quantité de tâches virtuellement illimitée. Basé sur Java, ce framework fait partie du projet Apache, sponsorisé par Apache software foundation [39].

1.6.2.1 L'écosystème de Hadoop

Hadoop de plusieurs éléments est composé de deux composants classés comme composants de base et composants supplémentaires [63] :

-composants de base : les principaux composants de Hadoop sont :

1. **YARN :** YARN(Yet Another Resource Negotiator) signifie encore un autre négociateur de ressources. Il gère et planifie les ressources et décide de ce qui doit se passer dans chaque noeud de données.
2. **Hadoop mapreduce :** MapReduce est un modèle de programmation massivement parallèle adapté au traitement de très grandes quantités de données. Les programmes adoptant ce modèle sont automatiquement parallélisés et exécutés sur des clusters (grappes) d'ordinateurs. MapReduce est un produit Google Corp.
3. **HDFS (Maintaining the Distributed File System)** Hadoop (HDFS en abrégé) est le système de stockage de données principal sous les applications Hadoop. Il s'agit d'un système de fichiers distribué et fournit un accès haut débit aux données d'application. Il fait partie du paysage du Big Data et permet de gérer de grandes quantités de données structurées et non structurées. HDFS distribue le traitement de grands ensembles de données sur des clusters d'ordinateurs bon marché [28].
   (a) **Architecture HDFS :**
   La structure HDFS apparaît sous la forme suivante (Figure 1.14) [28] :

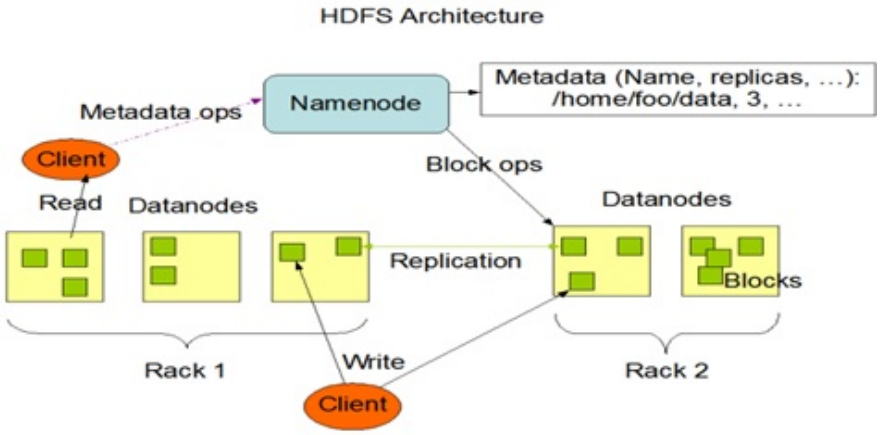


FIGURE 1.14 – Architecture HDFS

-Namenode :est le serveur maître qui gère l'espace de noms HDFS et régule l'accès aux fichiers par les clients. Il exécute des opérations telles que l'ouverture / la fermeture / renommer des fichiers et des répertoires et caraptive les blocs de données sur les données de Datanodes.

-Datanode : gère le stockage connecté aux noeuds qu'ils utilisent et servent des demandes de lecture / d'écriture des clients du système de fichiers. Il y a généralement l'un d'entre eux par grappe. Les fichiers sont divisés en blocs de données et stockés sur des jeux de Datanodes.

(b) **Les caractéristiques clés de HDFS :** les caractéristiques HDFS sont les suivantes [28] :

- **Système de fichiers distribué :** HDFS est un système de fichiers distribué (ou stockage distribué) qui gère de grands ensembles de données qui s'exécutent sur du matériel de base. Vous pouvez utiliser HDFS pour mettre à l'échelle un cluster Hadoop sur des centaines / milliers de noeuds.
- **Blocs :** HDFS est conçu pour prendre en charge des fichiers très volumineux. Il divise ces gros fichiers en petits morceaux appelés blocs. Ces blocs contiennent une certaine quantité de données qui peuvent être lues ou écrites, et HDFS stocke chaque fichier sous forme de bloc. Par défaut, la taille du bloc est de 128 Mo (mais vous pouvez la modifier en fonction de vos besoins). Hadoop distribue des blocs sur plusieurs noeuds.
- **Réplication :** vous pouvez répliquer les données HDFS d'un service HDFS à un autre. Les blocs de données sont répliqués pour fournir une tolérance aux pannes et une application peut spécifier le nombre de répliques d'un fichier. Le facteur de réplication peut être spécifié au moment de la création du fichier et peut être modifié ultérieurement. Les fichiers dans HDFS sont à écriture unique et n'ont qu'un seul graveur à la fois. Le NameNode prend toutes les décisions concernant la réplication des blocs et recevra un Heartbeat et un Blockreport de chacun des DataNodes du cluster.
- **Fiabilité des données :** HDFS crée une réplique de chaque bloc de données qui se trouve sur les noeuds d'un cluster donné, offrant une tolérance aux pannes. Si un noeud échoue, vous pouvez toujours accéder à ces données sur d'autres noeuds qui contiennent une copie des mêmes données dans ce cluster HDFS. Par défaut, HDFS crée trois copies de blocs.

1.6.2.2 Avantages de Hadoop

Il y a plusieurs avantages spécifiques pour Hadoop [34] :

- Apache Hadoop est un projet open source. Cela signifie que l'on peut modifier son code pour les besoins en entreprise.
- Hadoop est hautement évolutif.
- La tolérance aux pannes est la principale caractéristique de Hadoop.
- Hadoop peut travailler avec différents types de données. Il est suffisamment flexible pour stocker différents formats de données.
- Débit élevé et faible latence
- Hadoop est très évolutive, car nous pouvons facilement ajouter le nouveau matériel au noeud. Hadoop fournit également une évolutivité horizontale qui signifie que nous pouvons ajouter des noeuds à la volée sans aucun temps d'arrêt.

- Compatibilité dans Hadoop, HDFS est la couche de stockage et MapReduce est le moteur de traitement.
- Hadoop fonctionne sur un groupe de matériel de produits de base qui n'est pas très coûteux.
- Hadoop est très facile à utiliser, car il n'est pas nécessaire de faire face à l'informatique distribuée ; Le cadre prend soin de toutes les choses.

1.6.2.3 Inconvénients de Hadoop

il y a plusieurs inconvénients spécifiques pour Hadoop [34] :

- Absence de garanties donc lors du traitement des données sensibles collectées par l'entreprise, il est impératif de prévoir des mesures de sécurité obligatoires.
- problèmes de petites données. Où plateformes de Big Data sur le marché qui ne sont pas adaptées aux petites données.
- Hadoop ses opérations risquées il est également lié à diverses controverses car les cybercriminels peuvent facilement exploiter les frameworks basés sur Java. Hadoop est l'un de ces frameworks entièrement basés sur Java. Par conséquent, la plateforme est très vulnérable et peut subir des dommages.

1.6.3 Apache Spark

Définition 1.6.3. Apache Spark est un framework informatique, conçue pour accélerer le calcul. Il est basé sur Hadoop MapReduce et étend le modèle MapReduce pour l'utiliser efficacement pour plus de types de calculs, qui incluent des requêtes interactives et le traitement de flux. Spark est conçu pour couvrir un large éventail de charges de travail telles que les applications par lots, les algorithmes itératifs, les requêtes interactives et le streaming. En plus de prendre en charge toutes ces charges de travail dans un système respectif, cela réduit la charge de gestion liée à la maintenance d'outils séparés[37].

1.6.3.1 Caractéristiques Apache Spark

Il y a plusieurs avantages caractéristiques de Apache Spark[37] :

- **Traitement rapide :** la caractéristique la plus importante d'Apache Spark qui a incité le monde du Big Data à choisir cette technologie plutôt que d'autres est sa vitesse. Les mégadonnées se caractérisent par le volume, la variété, la vitesse et la véracité qui doivent être traitées à une vitesse plus élevée. Spark contient Resilient Distributed Dataset (RDD) qui permet de gagner du temps dans les opérations de lecture et d'écriture, lui permettant de s'exécuter presque dix à cent fois plus vite que Hadoop.
- **Flexibilité :**Apache Spark prend en charge plusieurs langages et permet aux développeurs d'écrire des applications en Java, Scala, R ou Python.

- **Informatique en mémoire :** Spark stocke les données dans la RAM des serveurs, ce qui permet un accès rapide et accélère à son tour la vitesse de l'analyse.
- **Meilleures analyses :** contrairement à MapReduce qui inclut les fonctions Map et Reduce, Spark inclut bien plus que cela. Apache Spark se compose d'un riche ensemble de requêtes SQL, d'algorithmes d'apprentissage automatique, d'analyses complexes, etc. Avec toutes ces fonctionnalités, les analyses peuvent être effectuées de manière plus efficace à l'aide de Spark.
- **Traitement en temps réel :** Spark est capable de traiter des données de streaming en temps réel. Contrairement à MapReduce qui ne traite que les données stockées, Spark est capable de traiter des données en temps réel et est donc capable de produire des résultats instantanés.

1.6.3.2 Composants Apache Spark

Apache Spark est composé de plusieurs composants comme suit[37] :

- **Apache Spark Core :** Spark Core est le moteur d'exécution général sous jacent de la plateforme Spark sur lequel reposent toutes les autres fonctionnalités. Il fournit des ensembles de données de calcul et de référencement en mémoire dans des systèmes de stockage externes.
- **Spark SQL :** Spark SQL(Structure Query Language) est le module d'Apache Spark pour travailler avec des données structurées. Les interfaces offertes par Spark SQL fournissent à Spark plus d'informations sur la structure des données et du calcul effectué.
- **Spark Streaming :** ce composant permet à Spark de traiter les données de streaming en temps réel. Les données peuvent être ingérées à partir de nombreuses sources telles que Kafka, Flume et HDFS (Hadoop Distributed File System). Ensuite, les données peuvent être traitées à l'aide d'algorithmes complexes et transférées vers des systèmes de fichiers, des bases de données et des tableaux de bord en direct.
- **MLlib (Machine Learning Library) :** Apache Spark est équipé d'une riche bibliothèque connue sous le nom de MLlib. Cette bibliothèque contient un large éventail d'algorithmes d'apprentissage automatique : classification, régression, clustering et filtrage collaboratif. Il comprend également d'autres outils pour la construction, l'évaluation et le réglage des pipelines ML. Toutes ces fonctionnalités aident Spark à évoluer sur un cluster.
- **Graphx :** Spark est également livré avec une bibliothèque pour manipuler des bases de données graphes et effectuer des calculs appelés Graphx. Graphx unifie le processus ETL (Extract, Transform, and Load), l'analyse exploratoire et le calcul itératif de graphes au sein d'un seul système.

1.6.4 Spark contre Apache Hadoop et MapReduce

" Spark vs. Hadoop " est un terme fréquemment recherché sur le Web, mais comme indiqué ci dessus, Spark est davantage une amélioration de Hadoop et, plus précisément, du composant de traitement de données natif de Hadoop, MapReduce. En fait, Spark est construit sur le framework MapReduce, et aujourd'hui, la plupart des distributions Hadoop incluent Spark. Comme Spark, MapReduce permet aux programmeurs d'écrire des applications qui traitent plus rapidement d'énormes ensembles de données en traitant des parties de l'ensemble de données en parallèle sur de grands clusters d'ordinateurs. Mais là où MapReduce traite les données sur disque, ajoutant des temps de lecture et d'écriture qui ralentissent le traitement, Spark effectue des calculs en mémoire, ce qui est beaucoup plus rapide. Par conséquent, Spark peut traiter les données jusqu'à 100 fois plus rapidement que MapReduce[37]. Les API intégrées de Spark pour plusieurs langages le rendent plus pratique et accessible pour les développeurs que MapReduce, qui a la réputation d'être difficile à programmer. Contrairement à MapReduce, Spark peut exécuter des applications de traitement de flux sur des clusters Hadoop à l'aide de YARN, le framework de gestion des ressources et de planification des tâches d'Hadoop. Comme indiqué ci dessus, Spark ajoute les capacités de MLlib, Graphx et SparkSQL. Et Spark peut gérer les données d'autres sources de données en dehors de l'application Hadoop, y compris Apache Kafka. Sinon, Spark est compatible avec et complémentaire à Hadoop. Il peut traiter des données Hadoop, y compris les données de HDFS (le système de fichiers distribuée Hadoop), HBase (une base de données non relationnelle qui s'exécute sur HDFS), Apache Cassandra (a NoSQL alternative to HDFS), Et Hive (a Hadoop-based data warehouse)[37].

1.7 Big data sur le cloud

Big Data est le grand jeu de données recueilli à partir de grands systèmes de réseau. Le cloud est l'emplacement que ces données sont traitées et accessibles, en utilisant généralement un logiciel comme modèle de service (SAAS) et utilise l'AI et l'apprentissage automatique pour présenter des données aux utilisateurs. Les grandes données et les données de cloud ont une relation symbiotique, car l'infrastructure de cloud permet de manière efficace le stockage, le traitement en temps réel et l' analyse de données à grande échelle rapidement. Le plus grand avantage d'utiliser le stockage en nuage pour les grandes données est l' évolutivité.
Sans l'informatique en nuage, il y aurait une énorme quantité de potentiel inexploité dans les grandes analytiques de données, car les ordinateurs actuels ne peuvent pas analyser cette échelle de données de manière gênée, voire pas du tout. En même temps, les grandes données jouent un rôle dans le développement du cloud computing car sans de grandes données, il n'y aurait plus près de la demande de solutions basées sur le cloud. Donc, vraiment, des services de Cloud Computing existent en raison de Big Data. La seule raison pour laquelle nous collectons de grandes données est que nous avons maintenant les services capables de

collecter, de stocker et de le traiter. Une combinaison des deux concepts, cloud et big data peut transformer les organisation en un leader efficace et axé sur les données[52].

1.8 Frameworks de traitement parallèle de larges graphes

Le monde devient un lieu de plus en plus conjoint et le nombre de sources de données telles que les réseaux sociaux, les transactions en ligne, les moteurs de recherche web et les appareils mobiles augmentent encore plus que prévu. Un grand pourcentage de cet ensemble de données croissant existe sous forme de données liées, plus généralement de graphes, et de tailles sans précédent. Alors que les données actuelles des réseaux sociaux contiennent des centaines de millions de noeuds reliés par des milliards de d'arcs, des données interconnectées provenant de capteurs répartis dans le monde entier qui forment Internet des objets peut provoquer une croissance exponentielle de cette taille. Bien que l'analyse de ces grands graphes soit essentielle pour les entreprises et les gouvernements qui les possèdent, des outils de Big Data conçus pour l'analyse de texte et de tuple tels que Mapreduce ne peut pas les traiter efficacement. Ainsi, les abstractions et Graph Processing Framework sont développé pour concevoir des algorithmes de graphes itératifs et traiter de grands graphes avec de meilleures performances et une meilleure évolutivité[15].

Graph Processing Framework (GPF) est un ensemble d'outils et de frameworks concn spéufiquent pour le traitement de grands graphes. Les sommets de graphe sont utilisés pour la modélisation des données et la modélisation des arêtes pour les relations entre les sommets. En règle générale, GPF comprend un flux de données d'entrée, un modèle d'implémentation et une interface de programmation d'application (API) qui contient un ensemble de fonctions qui implémentent les algorithmes graphes spécifiés. Il existe un nombre croissant de GPF et dans littératuse parmi ces plateformes on peut citer quelques une qui sont les plus important
.

L'une des plateformes les plus important rappelle six plateformes importantes premier Pregel [23]est un système de traitement de graphes à grande échelle développé Google. Il fournit un Framework évolutif pour exécuter et analyse de graphes sur des clusters de machines. Apache Giraph [22] est un système de traitement de graphe itératif construit pour une évolutivité élevée. Il gère les jobs en tant qu' un job sur Hadoop et utilise HDFS pour les E S. GPS [16]est un système à source ouverte pour une exécution d'algorithmes évolutif, tolérante à la faute et facile à programmer sur des graphes extrêmement volumineux. C'est un système distribué, conçu pour fonctionner sur un groupe de machines, telles que l'EC2 de Amazon. GraphLab [47]est un frameworks pour les calculs des graphes parallèles asynchrones pour machine learning. Il diffère de Pregel en ce sens qu'il ne fonctionne pas dans des étapes synchrones en BSP(Bulk Synchronous Parallel), mais permet plutôt de traiter les sommets de manière asynchrone sur la base d'un planificateur.

Doekemeijer et Varbanescu (2014) examinent 83 frameworks différents. Pour expliquer brièvement les aspects les plus pertinents des GPF, nous avons presenté dans le Tableau 4.3 frameworks. Quatre dimensions principales peuvent être utilisées pour classer les frameworks GPF : (1) la plateforme cible, (2) le modèle de calcul, (3) l'approche de traitement de graphe et (4) le modèle de communication[16] :

Framework	Year	Platform	Computation	Approach	Communication
Surfer [16]	2010	Distributed memory	MapReduce	Graph-centric	Message passing
Pegasus [16]	2009	Distributed memory	MapReduce	Matrix-centric	Dataflow
GBase [16]	2011	Distributed memory	MapReduce	Matrix-centric	Dataflow
Pregel [50]	2010	Distributed memory	BSP-based	Vertex-centric	Message passing
Graphx [49]	2013	Distributed memory	BSP-based	Graph-centric	Dataflow
Giraph [48]	2012	Distributed memory	BSP-based	Vertex-centric	Dataflow
Giraphx [15]	2013	Distributed memory	BSP-based	Vertex-centric	Shared memory
Gelly [16]	2015	Distributed memory	Delta iterative/GSA	Vertex-centric	Message passing
GraphLab [47]	2010	Single machine/Distributed memory	GAS model	Vertex-centric	Shared memory
GraphChi [16]	2012	Single machine	Sequential model	Vertex-centric, PSW	Shared memory
Totem [16]	2012	Hybrid CPU + GPU	BSP-based	Graph-centric	Hybrid CPU + GPU

TABLE 1.2 – Framework de traitement parallele de large graphes[16]

1.9 Conclusion

De nombreux mégadonnées sont présentés sous forme de graphes ou de réseaux à grande échelle. De nombreuses données volumineuses non graphes sont souvent converties en modèles graphes à des fins d'analyse. Une structure de données graphe est une bonne représentation des données le traitement graphe parallèle a toujours été un sujet très brûlant, il y a deux points principaux ici, le premier est de savoir comment mettre en parallèle les algorithmes

graphes, et l'autre est de trouver un cadre de traitement parallèle approprié. Dans le chapitre suivant, nous parlerons du Graphx cadre comme l'un des cadres.

Chapitre 2

Frameworks de traitement de larges graphes et les algorithmes de graphes

2.1 Introduction

La croissance des données structurées de graphes dans des applications modernes telles que les réseaux sociaux et les bases de connaissances crée un besoin crucial de platesformes évolutives et d'architectures parallèles pouvant le traiter en BSP(Bulk Synchronous Parallel). Malgré son rôle de premier plan dans les big analytics de données, Mapreduce n'est pas le modèle de programmation optimal pour le traitement des graphes cet chapitre explique pourquoi, puis explore les systèmes de développement pour lutter contre le défi de traitement des graphes exemple : Pregel, Giraph, Graphx où est le système de Google que les pouvoirs PageRank, ce qui en fait un système très intéressant d'étudier. C'est aussi l'inspiration pour Apache Giraph et Graphx .

2.2 Pregel

Pregel a été décrit pour la première fois dans un article publié par Google en 2010. Il s'agit d'un système de traitement de graphes à grande échelle (pensez à des milliards de noeuds) et a servi d'inspiration à Apache Giraph, que Facebook utilise en interne pour analyser son réseau social, ainsi qu' Apache Spark bibliothèque Graphx, qui fournit une API pour les calculs Pregel. Dans cette section nous allons présenter les entrées sorties de Pregel et les principes de pregel [50] suivi par des exemples d'implémentation [50].

2.2.1 Entrée et sortie

Il existe de nombreux formats de fichiers possibles pour les graphe, tels qu'un fichier texte, un ensemble de sommets dans une base de données relationnelle ou des lignes de bigtable. Pour éviter d'imposer un choix spécifique de format de fichier, Pregel découle la tâche d'interprétation d'un fichier d'entrée sous forme de graphe à partir de la tâche du calcul du graphe. De même, la sortie peut être générée dans un format arbitraire et stockée dans un format adapté à une application donnée. La bibliothèque de Pregel fournit des lecteurs et

de nombreux formats de fichiers courants, mais les utilisateurs ayant des besoins inhabituels peuvent écrire leur propre formats [50].

2.2.2 Principes de Pregel

L'idée est de "penser comme un sommet" le algorithmes dans le cadre de pregel sont des algorithmes dans lesquels le calcul de l'état pour un sommet donné ne dépend que des états de ses voisins. La Figure 2.1 montre le modèle de flux de données de Pregel. Un calcul de Pregel prend un graphe et un ensemble correspondant d'états de sommet comme ses intrants. À chaque itération, appelé superstep, chaque sommet peut envoyer un message à ses voisins, traiter les messages qu'il a reçus dans une superstep précédente et mettre à jour son état. Ainsi, chaque superstep est constitué d'une série de messages passés entre voisins et à la mise à jour de l'état sommet global [59].

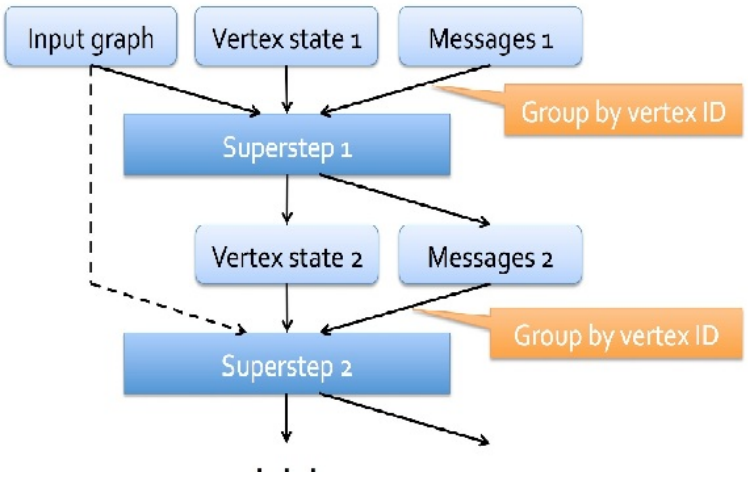


FIGURE 2.1 – Modèle de données de Pregel paradigm [59].

2.2.3 Exemples d'implémentations des algorithmes de graphes sous Pregel

Input: G :Graph[V,E]
while $err \geq \varepsilon$ **do**
  for *vertex* i **do**
    $R[i] = 0.15 + 0.85 \sum_{j \epsilon N_{in}(i)} M[j]$
    $M[i] = R[i]/|N_{out}|$
    Send $M[i]$ to all $N_{out}(i)$
  end
  $err = |R - previousR|$
end

Algorithm 1: Algorithme PageRank [59].

Pagerank est très intuitivement implémentée dans le paradigme de Pregel. À chaque superstep, chaque sommet met à jour son état avec une somme pondérée de PageRank de tous ses voisins, le traitement de l'ensemble des messages entrants précédents. Il envoie ensuite une part égale de son nouveau PageRank à chacun de ses voisins, envoyant un ensemble de messages sortants. Cela continue jusqu'à la convergence [59].

2.3 Giraph

Comme Google's Pregel est propriétaire, Yahoo ! lancé le développement du système Giraph comme une mise en oeuvre open source de Google Pregel. Giraph est mis en oeuvre en Java. Plus tard, Facebook a construit ses services de recherche sous frome de graphe sur Giraph sur le dessus et amélioré les performances [48].

2.3.1 Architecture

Chaque algorithme Giraph s'execute sur des sous ensembles de sommets divisés en partitions et traitées en parallèle. Le partitionnement est une optimisation clé permettant une exécution parallèle des tâches de traitement des graphes. Il est dynamique, ce qui signifie que la cession de partitions aux workers peut changer en fonction des caractéristiques de la demande de fonctionnement. À chaque superstep, toutefois, la structure fonctionnelle globale d'une application Giraph ressemble à la Figure 2.2[65]. Tous les services représentés sur cette figure sont internes à l'implémentation Giraph et ne sont pas visibles pour l'utilisateur final [65].

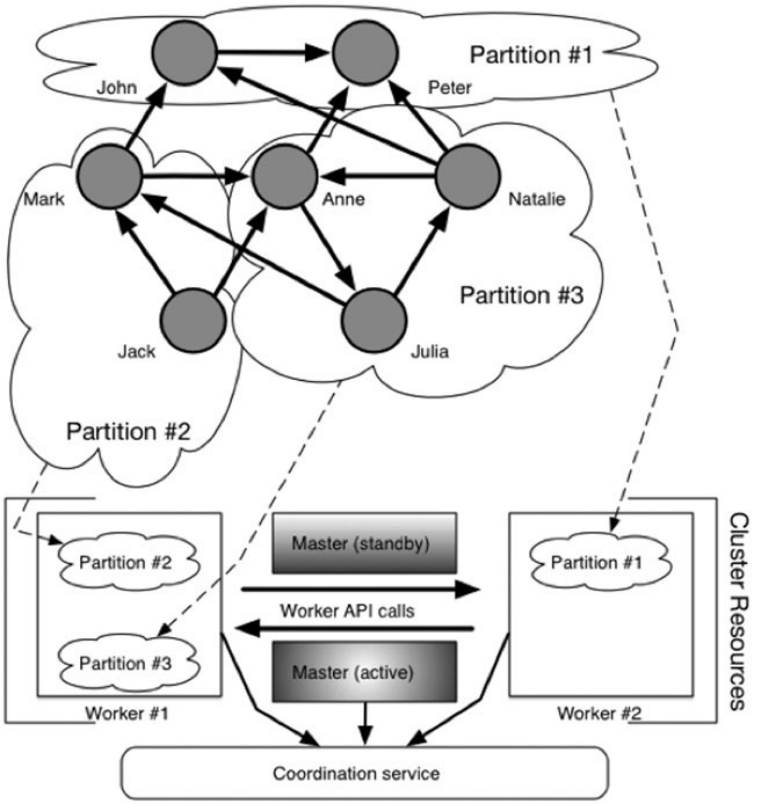


FIGURE 2.2 – Structure fonctionnelle d'une application Giraph en cours d'exécution[65].

Les composants de architechre Giraph sont les suivant :

- **Maitrse :** il y a toujours un service maître actif et quelques maîtres en attente qui souhaitent devenir actifs si le courant le maître échoue. Les maîtres de secours sont inactifs et ne jouent pas un rôle actif dans le cycle de vie d'une application Giraph. Une fois qu'un maître devient actif, son travail est de coordonner le calcul. Cette tâche consiste à effectuer les opérations suivantes[65] :
  - Transition des travailleurs (workers) d'un super-pas à l'autre de manière coordonnée.
  - Avant chaque super-étape, attribuer des partitions aux noeuds de calcul actifs.
  - Exécution du code de calcul principal.
  - Suivi de la santé et des statistiques de tous les travailleurs(workers).
- **Workers :** workers représentent la majorité des services Giraph. Leur fonction principale est de gérer l'état des partitions de graphe qui leur sont attribuées. Chaque workers expose un ensemble d'API réseau pour permettre aux workers distants de manipuler les données dans les partitions qui lui sont attribuées. Les workers passent d'un superpas à un superstep selon les instructions du maître actif [65]. Pendant chaque superstep, les workers itèrent sur les partitions graphe qu'ils possèdent et exécutent la méthode de Compute() pour tous les sommets appartenant à ces partitions. Les workers sont également responsables de vérifier leur état de temps en temps comme moyen de récupération après une défaillance d'un workers[65].

- **Coordonnateurs :** les noeuds exécutant le service de coordination fournissent le tissu nerveux pour le reste des services Giraph. Ils ne participez à aucun travail de traitement de graphe, au lieu de cela, ils fournissent des services de registre de configuration, de synchronisation et de dénomination distribués pour le reste de Giraph[65].

2.3.2 Avantages

Giraph présente de nombreux avantages et est comme suit [65] :

- Aucun verrouillage : communication basée sur des messages.
- Pas de sémaphores : synchronisation globale.
- Isolation d'itération : massivement parallisable.
- Communauté.
- Open Source
- Adaptable.
- Scalable.
- Mature .

2.3.3 Inconvénients

Giraph présente de nombreux inconvénients et est comme suit [65] :

- Les inconvénients de l'Apache Giraph les plus courants sont la necessité de de contrôle dans le Giraph et la construction de graphe sur Hadoop avec l'absence de sécurité dans ce framework .
- Repose sur Hadoop .
- Lourd .
- Moins de contrôle .

2.4 Graphx

Graphx est la nouvelle API Spark pour les graphes et le calcul parallèle aux graphes. À un niveau élevé, Graphx étend le Spark RDD en introduisant le resilient distributed property graphe, un multigraphe dirigé avec des propriétés attachées à chaque sommet et arête. Pour prendre en charge le calcul des graphes, Graphx expose un ensemble d'opérateurs fondamentaux (par exemple, subgraph, JoinVertices et MapReduceTriplets) ainsi qu'une variante optimisée de l'API Pregel. De plus, Graphx inclut une collection croissante d'algorithmes et de générateurs de graphes pour simplifier les tâches d'analyse de graphes [49].

2.4.1 Graphx et Spark Apache

Graphx est implémenté sous Apache de Spark, un moteur parallèle de données largement utilisé. Semblable à Hadoop MapReduce, un cluster Spark consiste en un seul noeud master et

plusieurs noeuds de workes. Le noeud master est responsable de la planification des tâches et de l'expédition, tandis que les noeuds de travailleurs sont responsables du calcul et du stockage de données physiques. Toutefois, Spark présente également plusieurs failles qui le différencient de MapReduce traditionnels et qui sont importants pour la conception de Graphx [49].

- **Caching in-Memory :** Spark fournit l'abstraction de stockage de mémoire de jeu de données de mémoire (RDD) de résiliente. Les RDD sont des collections d'objets qui sont partitionnés sur un cluster. Graphx utilise RDDS comme base de collections et de graphes distribués.
- **DAGS de calcul :** contrairement à la topologie MapReduce à deux étapes, Spark prend en charge les DAG de calcul général en composant plusieurs opérateurs parallèles de données sur RDD, ce qui le rend plus approprié pour exprimer des flux de données complexes. Graphx utilise et étend des opérateurs de Spark pour obtenir l'abstraction de programmation unifiée.
- **Tolérance des pannes basée sur la lignée :** RDD et les calculs parallèles de données sur les RDD sont tolérants à la faute. Spark peut reconstruire automatiquement toutes les données ou exécuter des tâches perdues lors des échecs.
- **Partitionnement programmable :** les RDD peuvent être co-partitionnement et co-partitioned. Lorsque vous rejoignez deux RDD qui sont co-partitionnées et co-partitioned, Graphx peut exploiter cette propriété pour éviter toute communication réseau.
- **Shell interactif :** Spark permet aux utilisateurs d'exécuter de manière interactive des commandes de Spark dans une coque Scala ou Python. Nous avons étendu le shell d'Spark pour prendre en charge les analyses de graphes interactives.

2.4.2 Pourquoi Graphx

Voici quelques raisons pour lesquelles Spark Graphx peut-être utilisé [49] :

- Vous avez déjà des pipelines de traitement de données Spark et souhaitez incorporer le traitement des graphes.
- Vous êtes parmi les nombreux pour lesquels des données graphes sont devenues importantes.
- Vos données graphes sont trop grandes pour s'adapter à une seule machine.
- Soit vous n'avez pas besoin de plusieurs applications accédant au même magasin de données ou vous envisagez d'ajouter un serveur de repos à Spark, par exemple, avec l'addon à l'origine par Ooyala appelée Spark job Server.
- Soit vous n'avez pas besoin de transactions de type de base de données ou vous planifiez d'utiliser une base de données graphe telle que NEO4J ou Titan en liaison avec Graphx.
- Vous avez déjà un cluster de Spark à votre disposition.
- Vous souhaitez utiliser le pouvoir concis et expressif de Scala.

2.4.3 Opérateurs Graphx

En tant que paradigme de traitement de graphe intégré dans Spark, Graphx comprend des opérateurs disponibles dans Spark (e.g. map, filter, and reduceByKey) en plus de certains opérateurs de graphe spécialisés, tel que [70] :

- **Property Operators :** dans lesquels un nouveau graphe est généré avec des propriétés de sommet ou d'arête modifié mais la structure n'est pas affectée (mapVertices, mapEdges, mapTriplets).
- **Structural Operators :** dans lesquels un nouveau graphe généré avec une structure modifiée (reverse, subgraph, mask, groupEdges).
- **Join Operators :** dans lesquels RDD est joint au graphe (joinVertices,externalJoinVertices).
- **Graph Builders :** fournit plusieurs façons de créer un graphe (fromEdges, fromEdgeTuples, edgeListFile) et pour repartitionner les arêtes du graphe use(partitionBy).
- **Neighbourhood Aggregation :** dans laquelle les informations sur les triplets adjacents sont agrégé (agrégatMessages, mapReduceTriplets).
- **Collecting Neighbours :** pour collecter les sommets voisins et leurs attributs à chaque vertex (collectNeighborIds, collectNeighbors).
- **Computing Degree Information :**(inDegrees, outDegrees, degrés).
- **Caching and Uncaching operators :** pour mettre en cache des graphes en mémoire ou les supprimer de la mémoire (persist, cache, unpersistVertices).

Listing 2.1 montre certaines fonctions du Graphx :

```
1
2  class Graph[VD, ED] {
3    // Information about the Graph ==================================================
       ==========================
4    val numEdges: Long
5    val numVertices: Long
6    val inDegrees: VertexRDD[Int]
7    val outDegrees: VertexRDD[Int]
8    val degrees: VertexRDD[Int]
9    // Views of the graph as collections ============================================
       ==========================
10   val vertices: VertexRDD[VD]
11   val edges: EdgeRDD[ED]
12   val triplets: RDD[EdgeTriplet[VD, ED]]
13   // Functions for caching graphs ================================================
       ==========================
14   def persist(newLevel: StorageLevel = StorageLevel.MEMORY_ONLY): Graph[VD, ED
       ]
15   def cache(): Graph[VD, ED]
16   def unpersistVertices(blocking: Boolean = false): Graph[VD, ED]
17   // Change the partitioning heuristic  ============================================
       ==========================
18   def partitionBy(partitionStrategy: PartitionStrategy): Graph[VD, ED]
19   // Transform vertex and edge attributes ==========================================
       ==========================
20   def mapVertices[VD2](map: (VertexId, VD) => VD2): Graph[VD2, ED]
```

```
21   def mapEdges[ED2](map: Edge[ED] => ED2): Graph[VD, ED2]
22   def mapEdges[ED2](map: (PartitionID, Iterator[Edge[ED]]) => Iterator[ED2]):
      Graph[VD, ED2]
23   def mapTriplets[ED2](map: EdgeTriplet[VD, ED] => ED2): Graph[VD, ED2]
24   def mapTriplets[ED2](map: (PartitionID, Iterator[EdgeTriplet[VD, ED]]) =>
      Iterator[ED2])
25     : Graph[VD, ED2]
26   // Modify the graph structure ================================================
      ==========================
27   def reverse: Graph[VD, ED]
28   def subgraph(
29       epred: EdgeTriplet[VD,ED] => Boolean = (x => true),
30       vpred: (VertexId, VD) => Boolean = ((v, d) => true))
31     : Graph[VD, ED]
32   def mask[VD2, ED2](other: Graph[VD2, ED2]): Graph[VD, ED]
33   def groupEdges(merge: (ED, ED) => ED): Graph[VD, ED]
34   // Join RDDs with the graph ==================================================
      ==========================
35   def joinVertices[U](table: RDD[(VertexId, U)])(mapFunc: (VertexId, VD, U) =>
       VD): Graph[VD, ED]
36   def outerJoinVertices[U, VD2](other: RDD[(VertexId, U)])
37       (mapFunc: (VertexId, VD, Option[U]) => VD2)
38     : Graph[VD2, ED]
39   // Aggregate information about adjacent triplets =============================
      ==========================
40   def collectNeighborIds(edgeDirection: EdgeDirection): VertexRDD[Array[
      VertexId]]
41   def collectNeighbors(edgeDirection: EdgeDirection): VertexRDD[Array[(
      VertexId, VD)]]
42   def aggregateMessages[Msg: ClassTag](
43       sendMsg: EdgeContext[VD, ED, Msg] => Unit,
44       mergeMsg: (Msg, Msg) => Msg,
45       tripletFields: TripletFields = TripletFields.All)
46     : VertexRDD[A]
47   // Iterative graph-parallel computation ======================================
      ==========================
48   def pregel[A](initialMsg: A, maxIterations: Int, activeDirection:
      EdgeDirection)(
49       vprog: (VertexId, VD, A) => VD,
50       sendMsg: EdgeTriplet[VD, ED] => Iterator[(VertexId, A)],
51       mergeMsg: (A, A) => A)
52     : Graph[VD, ED]
53   // Basic graph algorithms ====================================================
      ==========================
54   def pageRank(tol: Double, resetProb: Double = 0.15): Graph[Double, Double]
55   def connectedComponents(): Graph[VertexId, ED]
56   def triangleCount(): Graph[Int, ED]
57   def stronglyConnectedComponents(numIter: Int): Graph[VertexId, ED]
58 }
```

Listing 2.1 – Quelques opérateurs de Graphx [68]

2.4.4 API Graphx Pregel

Graphx fournit un opérateur Pregel qui est il recommandé pour les algorithmes itératives. L'opérateur Pregel nécessite la définition de trois fonctions [42] :

- Une méthode « Vertex Program » qui spécifie ce que chaque sommet doit faire avec les données.
- Une méthode « Envoyer un message » qui mappe chaque triplet en messages destinés au sommet source ou destination du Triplet.
- Une méthode « Merge Message » qui réduit tous les messages à un sommet en une seule valeur (une « valeur unique » pourrait en fait être une collection contenant plusieurs valeurs).

  L'opérateur de Pregel applique essentiellement la fonction "Envoyer un message" à tous les triplets du graphe afin de générer des messages, utilise la méthode "Message de fusion" pour réduire tous les messages destinés au même sommet, puis applique le "Programme Vertex" à chaque sommet à l'aide des données de message réduites. Il répète ce processus jusqu'à ce qu'une itération ne donne aucun nouveau message, à quel point il revient.

2.4.5 Ensembles de données distribués résilients

Les ensembles de données distribués résilients(RDD) sont les éléments de construction fondamentaux des programmes de Spark, fournissant une transformation flexible, de haute performance, de traitement parallèle et de la tolérance aux pannes. La classe graphe de base dans Graphx est appelée graphe, qui contient deux RDD : une pour les arêtes et un pour les sommets (voir la Figure 2.3) [49].

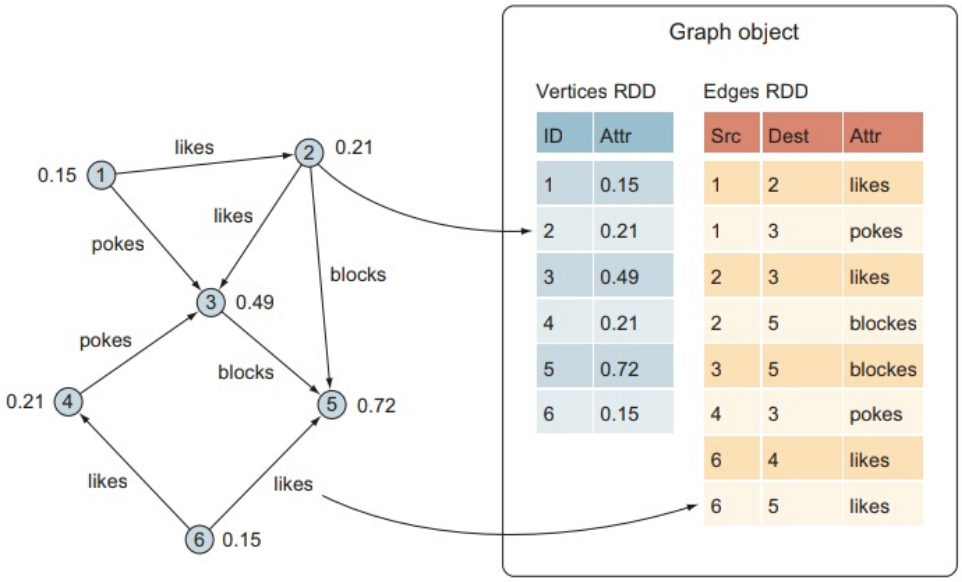


FIGURE 2.3 – Objet de Graphx graphe est composé de deux RDD : une pour les sommets et une pour les arêtes[49].

Dans Graphx, il n'est pas nécessaire de parcourir un graphe (à partir de certains sommets) pour accéder aux arêtes et aux sommets. Par exemple, la transformation des données de proportions de sommet peut être effectuée dans un script de Graphx de chute de Graphx, alors que dans les autres systèmes de traitement des graphes et des données graphes, une telle opération peut être faite en termes de requête nécessaire et de la manière dont un tel système va à l'encontre de l'opération. Les RDDs peuvent contenir tout type de Python, Java ou Scala, y compris des classes définies par l'utilisateur [49].

2.4.6 Modèle de stockage Graphx (Exprimer des graphes)

Le type de données principal dans Graphx et Spark est la base de données distribuée résiliente (RDD). Ce type de données est une collection d'informations en lecture seule. Étant donné qu'un RDD est en lecture seule, il n'ya aucun moyen de transformer le graphe une fois que cela a été créé, uniquement en créant un nouveau RDD peut modifier un graphe. Bien que cela soit vrai, les RDD ont leurs avantages car ils sont généralement plus rapides en raison de leur capacité à exécuter des copies de sauvegarde des noeuds et peuvent être programmées à l'esprit de la localité. Les RDD sont également très tolérants au défaillance et lorsque des défauts se produisent, la reconstruction de données peut être parallélisé afin que l'impact de la performance soit minimal. L'abstraction générale dans Graphx est un graphe de propriété, un graphe dirigé avec des métadonnées de arête stockées dans le graphe [30]. Les arêtes et les noeuds sont enregistrés dans leurs propres tables respectives ainsi que leurs métadonnées. Un exemple de ceci peut être illustré par la Figure 4.2.

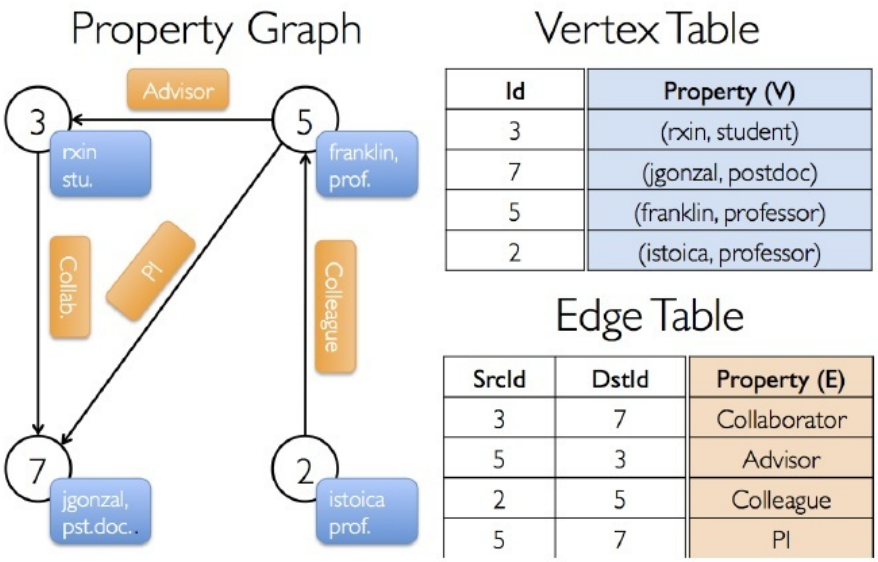


FIGURE 2.4 – Exemple d'un graphe de propriétés[30].

2.4.7 Modèle d'exécution

La base le Graphx est le moteur sous jacent Spark. Le modèle de Spark d'exécution générale peut être vu à la Figure 2.5. Le contexte Spark est un objet dans un programme de master qui contrôle le gestionnaire de cluster et le gestionnaire de cluster gère les noeuds de workes individuels qui réduisent [30].

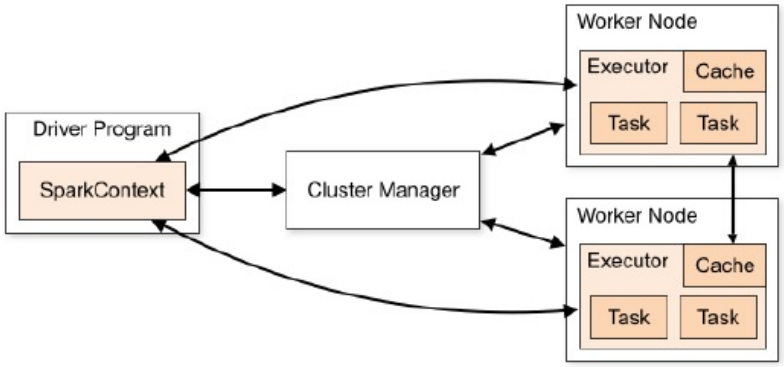


FIGURE 2.5 – Vue du modèle d'exécution de Spark[30].

Les tâches et les noeuds de travail renvoient ensuite les résultats de ces tâches au programme de master [30]. Cela se fait complètement en mémoire garantit des temps d'exécution rapides. Dans Graphx, pour le paralleralization, la bibliothèque utilise une technique appelée coupe d'arête comme illustré à la Figure 2.6.

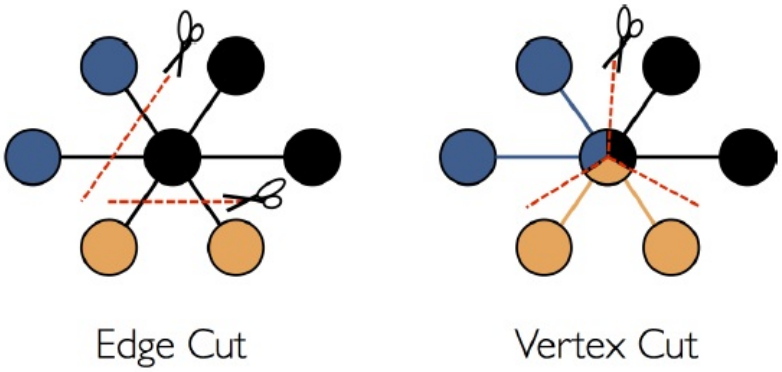


FIGURE 2.6 – Méthodes coupé arête et somment [30].

Ceci est fait soit à aléatoirement (par défaut), soit à l'aide d'une heuristique de coupe définie par l'utilisateur. Cela permet d'effectuer des opérations sur les sous graphes et les messages passés par une table de routage. Un exemple de ceci peut être vu à la Figure 2.7.

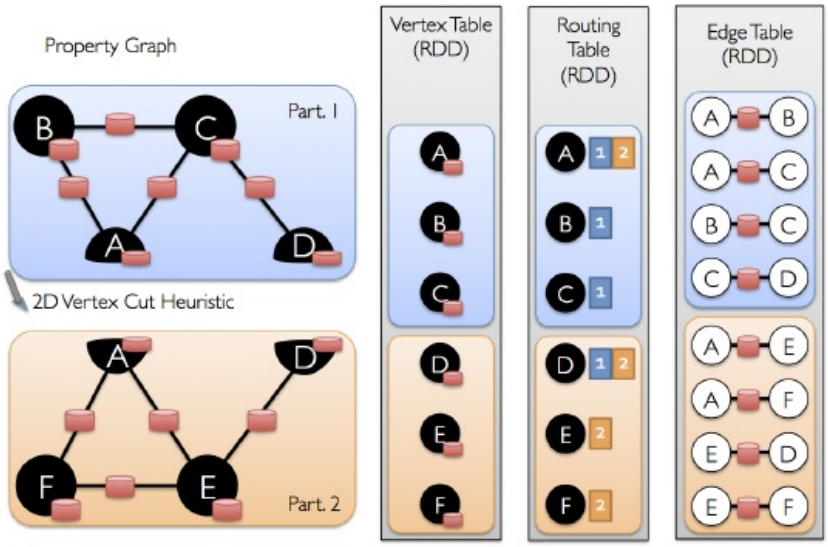


FIGURE 2.7 – Exemple d'un graphe de propriétés[30].

2.4.8 Syntaxe de Graphx

La langue principale utilisé par Graphx et Spark est Scala, bien que Java puisse être utilisée à la place de Scala. Étant donné qu'un graphe est juste un autre type de données composé d'arête et de sommetes exprimés en tant que RDD, Il peut être codé comme un RDD. Un exemple du code de la listing 2.2 est exprimé dans Scala est donné ci dessous [30] :

```
1
2  val sc: SparkContext
3
4  val users: RDD[(VertexId, (String, String))] =
5  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
6  (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
7
8  val relationships: RDD[Edge[String]] =
9  sc.parallelize(Array(Edge(3L, 7L, "collab"), Edge(5L, 3L, "advisor"),
10 Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))
11
12 val defaultUser = ("John Doe", "Missing")
13
14 val graph = Graph(users, relationships, defaultUser)
```

Listing 2.2 – Exemple Graphx

2.4.9 Algorithmes de graphes

Les algorithmes de graphes aident à analyser des ensembles de données de graphes.

Spark Graphx est livré avec un ensemble d'algorithmes de graphes prédéfinis pour faciliter le traitement des données graphes et les tâches d'analyse. Ces algorithmes sont disponibles dans le package org.apache.spark.graphx.lib. C'est aussi simple que d'appeler ces algorithmes

en tant que méthodes dans la classe graphe. Parmi ce algorithme ont trouve [57] :

- PageRank
- composant connexe
- Label propagation
- Singular Value Decomposition(SVD++)
- Strongly connected components
- Triangle count
- Single Source Shortest Paths(SSSP)
- Community Detection

Nous allons expliqer en détails sur les algorithmes suivants : PageRank, composant connexe.

- **PageRank :**

  L'algorithme PageRank est utilisé pour déterminer l'importance relative d'un objet dans un ensemble de données de graphe. Il mesure l'importance de chaque noeud dans un graphe, en supposant qu'un arête d'un autre sommet à ce noeud représente une approbation. Le moteur de recherche de Google est un exemple classique de PageRank. Google utilise le PageRank comme l'une des mesures pour déterminer l'importance d'une page Web en fonction du nombre d'autres pages Web qui la référencent. Un autre exemple est les réseaux social comme Twitter. Si un utilisateur de Twitter est suivi par de nombreux autres utilisateurs, alors cet utilisateur a une plus grande influence sur le réseau. Cette métrique peut être utilisée pour la sélection/le placement d'annonces auprès des utilisateurs qui suivent le premier utilisateur (100 000 utilisateurs suivent un chef $\geq$ probablement des gourmands). Graphx fournit deux implémentations de PageRank :

  — **PageRank statique :** cet algorithme fonctionne pour un nombre fixe d'itérations pour générer des valeurs PageRank pour un ensemble donné de sommet dans un ensemble de données graphe.

  — **Pagerank dynamique :** d'autre part, l'algorithme de PageRank dynamique s'exécute jusqu'à ce que les valeurs de Pagerank convergent sur la base d'une valeur de tolérance prédéfinie.

  Spark Graphx fournit les deux manières suivantes d'exécuter PageRank [25] :

  — **Static PageRank :** staticPageRank fonctionne pour un nombre défini d'itérations pour calculer le classement de la page. Voici la signature des opérations de PageRank statique : **Staticpagerank (int Numiter, double ResetProb)** cette méthode nécessite deux arguments. Le premier argument est le nombre d'itérations nécessaires à exécuter pour calculer le rang de page et le deuxième paramètre est la probabilité de réinitialisation utilisée pour calculer le facteur d'amortissement dans PageRank et sa valeur doit être 0,0 $\leq$ resetprob $\leq$1.0.

  — **Dynamic PageRank :** au lieu de prendre un nombre défini d'itérations comme

entrée, le PageRank dynamique nécessite que les utilisateurs fournissent une valeur de tolérance (double). Le nombre d'itérations dans ce cas est dynamique, car les rangs continueront à calculer jusqu'à ce que la valeur de classement change de plus que la valeur de tolérance spécifiée. Voici la signature de la méthode : **pageRank(double tol, double resetProb)**

l'implémentation de PageRank algorithme est dans le code 2.3 suivant :

```
1
2  package org.apache.spark.examples
3
4  import org.apache.spark.sql.SparkSessio
5  object SparkPageRank {
6
7    def showWarning(): Unit = {
8      System.err.println(
9        """WARN: This is a naive implementation of PageRank and is given
       as an example!
10         |Please use the PageRank implementation found in org.apache.
       spark.graphx.lib.PageRank
11         |for more conventional use.
12       """.stripMargin)
13   }
14
15   def main(args: Array[String]): Unit = {
16     if (args.length < 1) {
17       System.err.println("Usage: SparkPageRank <file> <iter>")
18       System.exit(1)
19     }
20
21     showWarning()
22
23     val spark = SparkSession
24       .builder
25       .appName("SparkPageRank")
26       .getOrCreate()
27
28     val iters = if (args.length > 1) args(1).toInt else 10
29     val lines = spark.read.textFile(args(0)).rdd
30     val links = lines.map{ s =>
31       val parts = s.split("\\s+")
32       (parts(0), parts(1))
33     }.distinct().groupByKey().cache()
34     var ranks = links.mapValues(v => 1.0)
35
36     for (i <- 1 to iters) {
37       val contribs = links.join(ranks).values.flatMap{ case (urls, rank)
       =>
38         val size = urls.size
39         urls.map(url => (url, rank / size))
40       }
41       ranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)
42     }
43
44     val output = ranks.collect()
```

```
45      output.foreach(tup => println(s"${tup._1} has rank:  ${tup._2} ."))
46
47      spark.stop()
48    }
49  }
```

Listing 2.3 – Code PageRank [13]

Exemple d'application

L'emplacement du programme PageRankExample.scala est mentionné ci dessous (Listing 2.4) :

Les ensembles de données pour ce programme sont présents dans le répertoire :

graphx / data / followers.txt

graphx/ data / users.txt

Laissez-nous comprendre la ligne de code par ligne :

— Importer Spark's Graphx et SQL Package.

— Crée une Spark.

— Chargez les artes sous forme de graphe.

— Run Pagerank.

— Rejoignez les rangs avec les noms d'utilisateur.

— Imprimer le résultat.

```
1  import org.apache.spark.graphx.GraphLoader
2    import org.apache.spark.sql.SparkSession
3    object PageRankExample {
4          def main(args: Array[String]): Unit = {
5              val spark = SparkSession.builder
6                  .appName(s"${this.getClass.getSimpleName}")
7                  .getOrCreate()
8              val sc = spark.sparkContext
9              val graph = GraphLoader.edgeListFile(sc, "data/graphx/
     followers.txt")
10             val ranks = graph.pageRank(0.0001).vertices
11             val users = sc.textFile("data/graphx/users.txt").map { line
     =>
12                 val fields = line.split(",")
13                 (fields(0).toLong, fields(1))
14             }
15             val ranksByUsername = users.join(ranks).map {
16                 case (id, (username, rank)) => (username, rank)
17             }
18             println(ranksByUsername.collect().mkString("\n"))
19             spark.stop()
20         }
```

Listing 2.4 – Exempe Code PageRank [13]

- **composant connexe (SCC) :**

  Un composant connexe dans un graphe est un sous-graphe connecté lorsque deux sommets sont connectés les uns aux autres par un arête et il n'y a pas de sommets supplémentaires dans le graphe principal. Cela signifie que les deux noeuds appartiennent au même composant connecté lorsqu'il y a une relation entre eux. Le sommet

numéroté le plus petit ID dans le sous graphe est utilisé pour étiqueter les composants connexes dans un graphe. Les composants connexe peuvent être utilisés pour créer des clusters dans le graphe par exemple dans un réseau social.
Un autre algorithme est appelé des composants fortement connectés (SCC) dans le traitement des données graphes. Si tous les noeuds d'un graphe sont accessibles à partir de chaque noeud, le graphe est considéré comme fortement connecté [25].

— **Implémentation**

L'implémentation de cet algorithme est dans le code 2.5 suivant :

```
package org.apache.spark.graphx.lib

import scala.reflect.ClassTag

import org.apache.spark.graphx._

/** Connected components algorithm. */
object ConnectedComponents {
  /**
   * Compute the connected component membership of each vertex and return a graph with the vertex
   * value containing the lowest vertex id in the connected component containing that vertex.
   *
   * @tparam VD the vertex attribute type (discarded in the computation)
   * @tparam ED the edge attribute type (preserved in the computation)
   * @param graph the graph for which to compute the connected components
   * @param maxIterations the maximum number of iterations to run for
   * @return a graph with vertex attributes containing the smallest vertex in each
   *         connected component
   */
  def run[VD: ClassTag, ED: ClassTag](graph: Graph[VD, ED],
                                      maxIterations: Int): Graph[VertexId, ED] = {
    require(maxIterations > 0, s"Maximum of iterations must be greater than 0," +
      s" but got ${maxIterations}")

    val ccGraph = graph.mapVertices { case (vid, _) => vid }
    def sendMessage(edge: EdgeTriplet[VertexId, ED]): Iterator[(VertexId, VertexId)] = {
      if (edge.srcAttr < edge.dstAttr) {
        Iterator((edge.dstId, edge.srcAttr))
      } else if (edge.srcAttr > edge.dstAttr) {
        Iterator((edge.srcId, edge.dstAttr))
      } else {
        Iterator.empty
      }
    }
    val initialMessage = Long.MaxValue
    val pregelGraph = Pregel(ccGraph, initialMessage,
      maxIterations, EdgeDirection.Either)(
```

```
      vprog = (id, attr, msg) => math.min(attr, msg),
      sendMsg = sendMessage,
      mergeMsg = (a, b) => math.min(a, b))
    ccGraph.unpersist()
    pregelGraph
  } // end of connectedComponents

  /**
   * Compute the connected component membership of each vertex and return a graph with the vertex
   * value containing the lowest vertex id in the connected component containing that vertex.
   *
   * @tparam VD the vertex attribute type (discarded in the computation)
   * @tparam ED the edge attribute type (preserved in the computation)
   * @param graph the graph for which to compute the connected components
   * @return a graph with vertex attributes containing the smallest vertex in each
   *         connected component
   */
  def run[VD: ClassTag, ED: ClassTag](graph: Graph[VD, ED]): Graph[VertexId, ED] = {
    run(graph, Int.MaxValue)
  }
}
```

Listing 2.5 – Code composant connexe [13]

— **Exemple d'application** L'emplacement du programme ConnectedComponentsExample.scala est mentionné ci dessous(Listing 2.6) :

Les ensembles de données pour ce programme sont présents dans le répertoire :

graphx / data / followers.txt

graphx/ data / users.txt

Laissez-nous comprendre la ligne de code par ligne :

— Importez le package Graphx et SQL de Spark.

— Crée une SparkSession.

— Chargez le graphe comme dans l'exemple PageRank.

— Trouvez les composants connectés.

— Joignez les composants connectés avec les noms d'utilisateur.

— Imprimer le résultat

```
//construire le graphe
import org.apache.spark.graphx.GraphLoader
  import org.apache.spark.sql.SparkSession
  object ConnectedComponentsExample {
        def main(args: Array[String]): Unit = {
            val spark = SparkSession.builder
                .appName(s"${this.getClass.getSimpleName}")
                .getOrCreate()
            val sc = spark.sparkContext
```

```
10            val graph = GraphLoader.edgeListFile(sc, "data/graphx/
    followers.txt")
11            val cc = graph.connectedComponents().vertices
12            val users = sc.textFile("data/graphx/users.txt").map {
    line =>
13                val fields = line.split(",")
14                (fields(0).toLong, fields(1))
15            }
16            val ccByUsername = users.join(cc).map {
17                case (id, (username, cc)) => (username, cc)
18            }
19            println(ccByUsername.collect().mkString("\n"))
20            spark.stop()
21        }
22    }
```

Listing 2.6 – Code exemple composant connexe [13]

2.4.10 Graphx VS base de données graphes

Car Graphx est strictement un système de traitement en mémoire, nous avez besoin d'un endroit pour stocker les données graphes. Spark s'attend à ce que le stockage distribué, tel que HDFS, Cassandra ou S3. Mais certains utilisent Graphx, un système de traitement de graphe, conjointement avec une base de données graphe pour obtenir le meilleur des deux environement (voir la figure 2.8). Graphx versus NEO4J est un débat de fréquentation, mais pour certains cas d'utilisation, les deux sont meilleurs que l'un ou l'autre. Hadoop, Spark fournit une API (calcul distribué) map / reduce plus le stockage distribué. La principale différence est que Spark stocke des données dans la RAM dans tout le cluster, tandis que Hadoop stocke des données sur le disque dans tout le cluster. Graphx s'appuie sur les fondements de Spark afin de fournir un traitement flexible et efficace parallèle. Spark fournit également un traitement parallèle des données qui le rend idéal pour les gros problèmes de données réels qui appellent souvent à la fois au processus de graphe et parallèle. Graphx n'est pas une base de données graphe et n'est pas adapté pour interroger des sommets individuels ou de petits groupes de sommets. Il s'agit plutôt d'un système de traitement de graphe adapté aux algorithmes masculinels tels que PageRank [49].

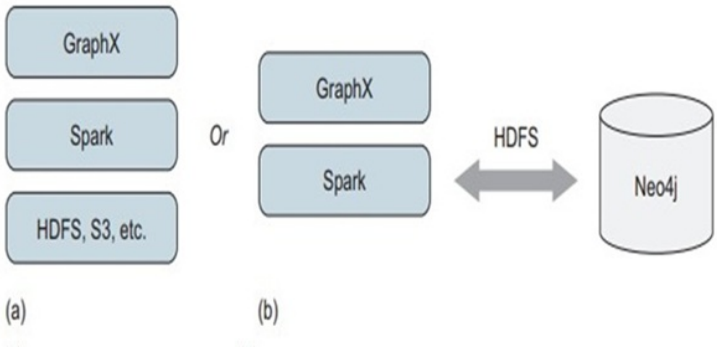


FIGURE 2.8 – Stockage de donées de graphes [49].

2.4.11 Caractéristiques de Spark Graphx

Les caractéristiques de Spark Graphx sont les suivantes[14] :

- **Souplesse :** nous pouvons travailler avec des graphes et des calculs avec Spark Graphx. Il inclut une analyse exploratoire (extrait, transformation et charge), ainsi que graphe itératif. Il est possible d'afficher les mêmes données que les graphes, les collections, la transformation et les graphes de jointure avec RDD. Aussi à l'aide de l'API de Pregel, il est possible d'écrire des algorithmes de graphes itératifs personnalisés.
- **Vitesse :** la vitesse est l'une des meilleures fonctionnalités de Graphx. Il fournit des performances comparables aux systèmes de traitement des graphes spécialisés les plus rapides. Il est le plus rapide de comparer avec les autres systèmes de graphes. Même en conservant la flexibilité d'une Spark, la tolérance aux défaillances et la facilité d'utilisation.
- **Bibliothèque d'algorithme en croissance :** fondamentalement, nous avons une bibliothèque croissante d'algorithmes de graphes que Spark Graphx offre. Nous pouvons en choisir. Certains des algorithmes populaires tels que le PageRank, les composants connees, la propagation des étiquettes. Comprend également des composants SVD ++, fortement connectés et le nombre de triangles.
- **Communauté :** dans le framwork du projet Apache Spark, Graphx est également développé. Il est donc testé et mis à jour avec chaque Spark.

2.4.12 Avantages de Graphx

Les avantages du Graphx sont les suivants[12] :

- Graphx simplifie les tâches d'analyse de graphes en réutilisant le concept de RDD.
- Il fournit une API pour un développement rapide et robuste pour tirer parti des graphes.
- Graphx est largement utilisé dans les analyses de données et l'informatique, car les graphes sont la structure de données idéale pour décrire les réseaux sociaux. Pour

cette raison, des entreprises comme Facebook mettent l'accent sur le développement de logiciels.

- Graphx optimise le moyen de représenter le sommet et les arêtes lorsqu'ils sont des types de données primitifs.
- Graphx prend en charge les opérateurs fondamentaux tels que sous graphe, rejoindre les sommets et les messages agrégés.

2.5 Comparison entre Giraph et Graphx

Une équipe Facebook a récemment publié une comparaison des performances de leur système de traitement de graphes basé sur Giraph avec le nouveau paradigrme Graphx, qui fait partie du Spark leur conclusion estla suivante[56] :

- Giraph a été amélioré, même sur des ensembles de données graphes plus petits.
- Giraph était également plus efficace de la mémoire.
- Graphx permet de lire des graphes à partir de la ruche à l'aide d'une requête de type SQL, permettant des transformations de colonnes arbitraires.
- L'utilisation de Scala à partir de l'environnement Shell est un moyen pratique de tester des applications de Graphx simples.

Dans ce tableau, nous résumons les différences et les similarités les plus importantes entre Giraph et Graphx[3] :

System	Pregel/Giraph	Spark/Graphx
Memory/Disk	Memory	Memory/Disk
Architecture	Parallel	Parallel
Computing paradigm	VertexCentric	BSPextension
Declarative Language	Memory	Parallel
Partitioning	×	×
Synchronization	Synchronous	Synchronous/Asynchronous
FaultTolerance	Global checkpoint	Global checkpoint

TABLE 2.1 – Comparison entre Giraph et Graphx[3].

2.6 Conclusion

Dans ce chapitrer nous avons détaillé trois systèmes de traitement parallèle de de larges graphes : Pregel, Giraph et Graphx. Ces trois systèmes sont considérés parmi, les plus important et les plus populaire nous avons un aussi que Graphx est l'un des meilleurs choix pour le traitement des large graphes. Il fournit un algorithme de traitement de données unifié et des outils de solution pour la fourniture de connaissances précieuses et de modèles de prévision sur les données connectées générées par divers processus métier dans des organisations, dans le prochain chapitre nous allons proposer un algorithme de graphe sous Graphx.

Chapitre 3

Propsition d'un algorithme de graphe distribué sous Graphx

3.1 Introduction

Les graphes jouent un rôle irremplaçable dans un large éventail de domaines d'application. Cependant, le traitement de graphe est confronté à des défis à tous les niveaux, de l'architecture du système aux paradigmes de programmation. Le problème de la coloration des graphes est l'un des problèmes les plus célèbres dans le domaine de la théorie des graphes et a une longue et illustre histoire. Le problème que nous avons essayé de réaliser est de colorer une grille avec un nombre de couleurs imposé k, ou le problème de coloration k, que nous avons intériorisé dans la coloration de graphes. Le problème de la coloration de graphe, est de trouver une couleur pour chaque sommet afin qu'il soit différent de ses voisins. Dans ce chapitre nous allons concentrer sur le problème de coloration des graphes et les domaines d'application et historique de coloration.

De graphe où nous allons présenter quelques algorithmes séquentiels et heuristiques pour résoudre ce problème, parmi les algorithmes existants : Glouton, Dsatur et Welsh et Powell. Ce denier est le plus utilisé dans la théorie des graphes pour la coloration de tous les sommets d'un graphe avec plus petit nombre de couleurs possible (nombre chromatique). Ainsi ; la suite nous allons essayer d'adapter (rend parallèle) cet algorithme pour être s'exécuter sous Graphx.

3.2 Problème de coloration de graphe

Les deux problèmes suivants sont définis dans la zone de coloration de graphe :
1. Le problème d'optimisation de la coloration graphe : étant donné un graphe G, il est nécessaire de déterminer le nombre minimum de couler nécessaires pour colorer le graphe afin que deux sommets adjacents ne soient pas de couleur de la même couleur. Le nombre utilisé de couler s'appelle le nombre chromatique du graphe .

Définition 3.2.1. On appelle nombre chromatique d'un graphe G=(X,E), le plus petit nombre de couleurs nécessaires pour colorier ce graphe. Ce nombre est noté X(G) . Le nombre

chromatique est toujours compris entre 1 et le nombre de sommets [72].

1. Le nombre chromatique d'un graphe est égal à 2 si et seulement si il n'existe pas de cycle de longueur impaire[72].
2. Si un graphe contient un sous graphe complet de p points, alors son nombre chromatique est supérieur ou égal à p[72].
3. Les arbres et les forêts qui sont des graphes qui ne contiennent pas de cycle sont 2-chromatiques et peuvent donc être coloriés avec seulement deux couleurs[72].

3.3 Historique

Le problème de le colaration des graphes a été noté pour la première fois en 1852 par un étudiant de l'university college london, francis guthrie (1831-1899), qui en coloriant une carte des comtés d'angleterre, remarqué que seules quatre couleurs étaient nécessaires pour garantir que tous les comtés voisins se sont vu attribuer des couleurs différentes [45]. Pour montrer comment la coloration des cartes est liée à la coloration des sommets d'un graphe, prenons l'exemple de la carte des comtés historiques du Pays de Galles donnée à la figure 4.1 (a). Cette carte particulière concerne 16 ≪ régions≫, dont 14 comtés, la mer à gauche et l'angleterre limitrophe à droite. La Figure 4.1 (d) montre que cette carte peut en effet être colorée en utilisant seulement quatre couleurs (gris clair, gris foncé, noir, blanc) [45] ?

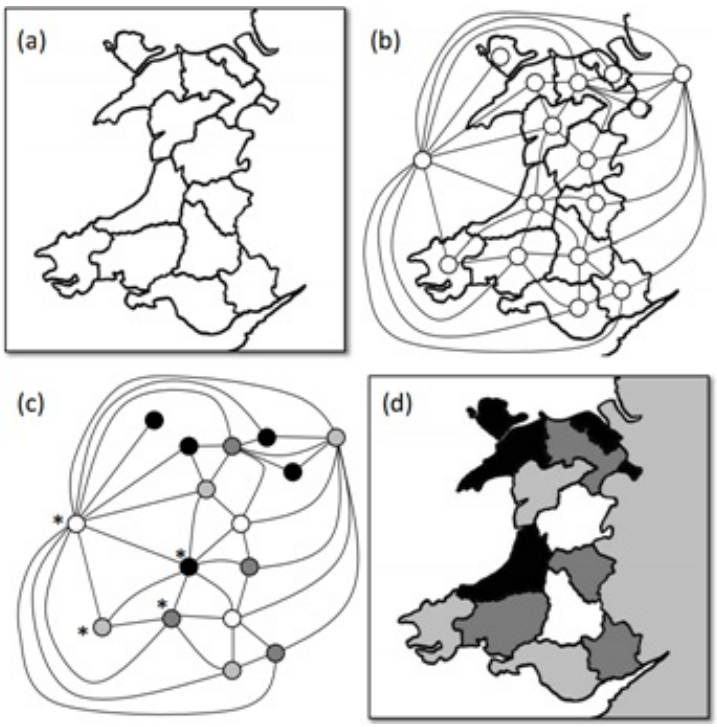


FIGURE 3.1 – Illustration de la façon dont les graphe peuvent être utilisés pour colorer les régions d'une carte[44].

3.4 Quelques applications pratiques simples

Considérons maintenant quatre applications pratiques simples de coloration graphe pour illustrer davantage les concepts sous jacents du problème [45] :

-Un exercice d'équipe : comme d'exemple, imaginez que nous avons un ensemble d'étudiants universitaires que nous souhaitons diviser des intogroupes pour un exercice de construction d'équipes. De plus, imaginez que nous sommes intéressés à diviser les étudiants afin qu'aucun étudiant ne soit mis dans un groupe contenant un ou plusieurs de ses amis, et afin que le nombre de groupes utilisés soit minime.

- Construire des horaires : une deuxième application importante de coloration de graphe se pose dans la production de calendriers dans les collèges et les universités. Dans ces problèmes, nous recevons un ensemble d'événements, tels que des conférences, des examens, des sessions en classe. La tâche consiste ensuite à attribuer les événements aux délais de passage conformément à un ensemble de contraintes. L'une des plus importantes de ces contraintes est ce qui est souvent connu sous le nom de la contrainte "événements clash ". Cela précise que si une personne (ou une autre ressource dont il n'y en a qu'un) est nécessaire pour être présente dans une paire d'événements, ces événements ne doivent pas être attribués à la même pièce de temps car une telle affectation entraînera cette personne / ressource devant être à deux endroits à la fois.

-Planification des taxis : imaginez qu'un cabinet de taxi a reçu n des réservations de voyage, chacune d'entre eux ayant une heure de départ, indiquant que le taxi quittera le dépôt et une heure de fin de nous le dire lorsque le taxi devrait revenir.

- Allocation de registre du compilateur : est notre quatrième et dernier exemple concerne l'attribution de variables de code informatique à des registres sur un processeur informatique. Lors de l'écriture de code dans un langage de programmation particulière, que ce soit C ++, Pascal, Fortran ou une autre option, le programmeur est libre d'utiliser autant de variables qu'il voit en forme.

3.5 Autres domaines d'applications

La coloration de graphe est très prisée pour la résolution de problèmes complexes d'optimisation combinatoire. En effet, chaque fois que l'on désire minimiser le nombre de ressources pour effectuer une série de tâches avec la contrainte que certaines tâches ne doivent pas partager la même ressource, il suffit de résoudre le problème de coloration pour le graphe dont les sommets représentent les tâches et où deux sommets sont adjacents si les tâches correspondantes violent la contrainte. Ainsi, l'allocation des bandes de fréquences dans un réseau cellulaire où des cellules voisines ne doivent pas avoir la même bande de fréquences afin d'éviter les interférences. La planification des examens dans une université de sorte qu'un étudiant n'ait pas deux examens au même moment. L'organisation d'un tournoi sur la plus

courte durée possible en gardant à l'esprit que certaines rencontres ne sauraient avoir lieu simultanément, la gestion des ressources dans un univers de cloud computing. Sont quelques exemples dans la longue liste des applications pratiques du problème de coloration de graphe [2].

3.6 Algorithmes séquentiels de colorations de graphes

Dans cette section, nous allons cite quelques algorithmes séquentiels de colorations de graphes :

3.6.1 Algorithme Glouton

Les algorithmes gloutons sont couramment utilisés dans la résolution de problèmes. Le principe de tels algorithmes consiste à choisir des solutions locales optimales d'un problème dans le but d'obtenir une solution optimale globale au problème. Dans le cas du coloriage de graphe, cela va consister à colorier les sommets un par un avec la plus petite couleur possible ; l'ensemble des couleurs possibles étant donné par les couleurs de ses voisins. L'ordre dans lequel les sommets sont traités définit les différentes variantes de d'algorithme. Une méthode est de colorer les sommets par ordre de difficultés décroissantes [64].

L'algorithme 2 présent le pseudo code de la méthode de glouton.

Function `glouton`$(S \leftarrow \theta,\ \pi)$
- **for** $i \leftarrow 1$ **to** $|\pi|$ **do**
  - **for** $j \leftarrow 1$**to** $|S|$ **do**
    - **if** $(S_j \cup \{\pi_i\})$*is an independent set* **then**
      - $S_j \leftarrow S_j \cup \{\pi_i\}$
      - *break*
    - **else**
      - $j \leftarrow j + 1$
- **if** $j > |S|$ **then**
  - $S_j \leftarrow \{\pi_i\}$
  - $S \leftarrow S \cup S_j$

Algorithm 2: Algorithme glouton [45].

Remarque 3.6.1. L'algorithme glouton le plus utilisé est l'algorithme de Welsh Powel que nous allons présenter par le suite. Néanmoins, il existe d'autres algorithmes reposant sur le principe glouton, moins connu, comme Dsatur et RLF, ayant des limites.

3.6.2 Algorithme DSATUR

Dsatur est un algorithme de coloration de graphes créé par Daniel Brélaz en 1979 à l'EPFL (École Polytechnique Fédérale de Lausanne). Il s'agit d'un algorithme de coloration séquentiel par heuristique. DSAT est l'abréviation de dégrée de saturation [64] :

Propriété 3.6.1. *On considère un graphe $G = (V, E)$ simple connexe et non orienté. Pour chaque sommet v de V, on calcule le degré de saturation $DSAT(v)$ de la manière suivante :*

- *Si aucun voisin de x est colorié, alors $DSAT(x) =$ le degré de x.*
- *Sinon, $DSAT(x) =$ le nombre de couleurs différentes utilisées dans le voisinage de x.*

L'algorithme $DSATUR$ est un algorithme de coloration séquentiel, au sens ou il colorie un seul sommet à la fois et tel que :

1. Au départ le graphe n'est pas colorié.
2. On colorie un sommet non déjà colorié.
3. On stoppe $DSATUR$ quand tous les sommets de G sont coloriés.

Méthode d'application de l'algorithme :

1. Ordonner les sommets par ordre décroissant de degré.
2. Colorer un sommet de degré maximum avec la couleur 1.
3. Choisir un sommet non coloré avec $DSAT$ maximum. Si conflit choisir celui avec degré maximum.
4. Colorer ce sommet par la plus petite couleur possible.
5. Si tous les sommets sont colorés alors stop. Sinon aller en 3.

Et voilà le pseudo code d' Algorithme $DSATUR$ ci-dessus dans la Algorithme 3 :

Function `DSATUR`($S \leftarrow \theta$, $X \leftarrow V$)
- **while** $X \neq \theta$ **do**
  - *Choose* $v \in X$
  - **for** $j \leftarrow 1$**to**$|S|$ **do**
    - **if** $(S_j \cup \{v\})$ *is an independent set* **then**
      - $S_j \leftarrow S_j \cup \{v\}$
      - *break*
    - **else**
      - $j \leftarrow j + 1$
- **if** $j > |S|$ **then**
  - $S_j \leftarrow \{v\}$
  - $S \leftarrow S \cup S_j$
  - $X \leftarrow X - \{v\}$

Algorithm 3: Algorithme DSATUR[45].

3.6.3 Algorithme RLF(Recursive Largest First)

Bien que l'algorithme DSATUR de coloration de graphe soit similaire dans le comportement et la complexité de l'approche gourmande classique, la prochaine méthode constructive que nous examinons, le premier algorithme (RLF) récursif, suit une stratégie légèrement différente. L'algorithme RLF a été conçu à l'origine par Leighton (1979), en partie pour une utilisation dans la construction de solutions aux gros problèmes de calendrier. La méthode

fonctionne en colorant une couleur à la fois, par opposition à un sommet à la fois. À chaque étape, l'algorithme utilise des heuristiques pour identifier un ensemble de sommets indépendants dans le graphe, qui sont ensuite associés à la même couleur. Cet ensemble est ensuite retiré du graphe et le processus est répété sur le sous graphe résultant. Ce processus continue jusqu'à ce que le sous graphe soit vide[45].

Et voilà le pseudo code d' Algorithme RLF ci-dessus dans la algorithme 4 :

Function `RLF`($S \leftarrow \theta$, $X \leftarrow V$, $Y \leftarrow \theta$, $i \leftarrow 0$)
- **while** $X \neq \theta$ **do**
  - $i \leftarrow i + 1$
  - $S_i \leftarrow \theta$
  - **while** $X \neq \theta$ **do**
    - *Choose* $v \in X$
    - $S_i \leftarrow S_i \cup \{v\}$
    - $Y \leftarrow Y \cup \Gamma_x(v)$
    - $X \leftarrow X - (Y \cup \{v\})$
    - $S \leftarrow S \bigcup \{S_i\}$
    - $X \leftarrow Y$
    - $X \leftarrow \theta$

Algorithm 4: Algorithme RLF[45].

Dans chaque boucle extérieure du processus, la classe de couleur S_i est construite. L'algorithme utilise également deux ensembles : X, qui contient des sommets non collants qui peuvent actuellement être ajoutés à S_i sans causer un affrontement, et Y, qui détient les sommets non collants qui ne peuvent pas être ajoutés de manière géré à S_i. $\Gamma x(v)$ désigne le sous-ensemble de sommets dans le jeu Xqui sont adjacents au sommet V.

Exemple 3.6.1.

- Tout d'abord, il faut repérer le sommet de plus au degré : c'est le sommet 1(voir Figure 3.2(a)).
- Nous allons donc lui attribuer une couleur, et éliminer tous ses sommets adjacents. Ceux qui ne le sont pas prendront la couleur du sommet 1 (voir Figure 3.2(b)).
- Puis, nous allons étudier un sous-graphe où les deux premiers sommets coloriés sont exclus, et choisir un nouveau sommet (le sommet 5) et lui attribuer une autre couleur. On remarque qu'il faut aussi exclure les sommets adjacents de l'autre sommet (sommet 3) de même couleur que notre sommet choisi(voir Figure 4.3(c)).
- On réédite à nouveau les étapes ci-dessus, pour obtenir le graphe suivant (il ne reste plus que deux sommets indépendants)(voir figure 3.2(d)) .

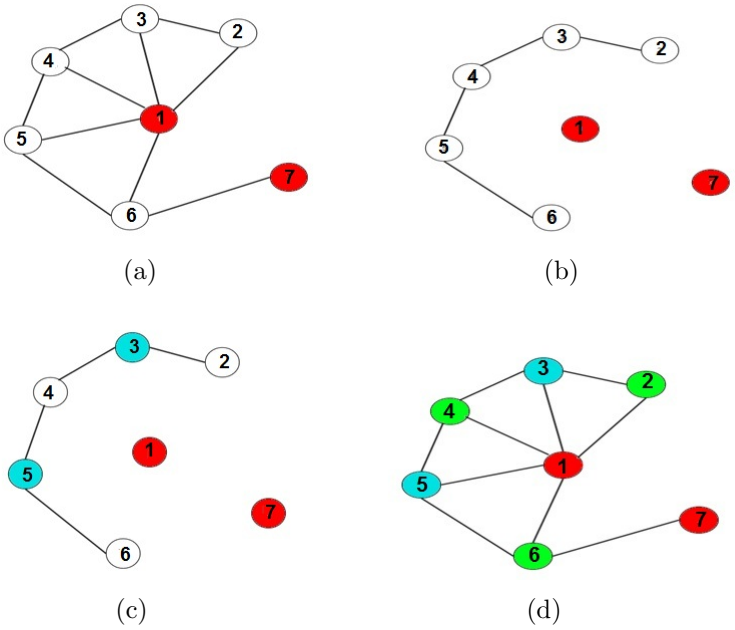


FIGURE 3.2 – Étapes l' agorithme RLF d'exécution

3.6.4 Algoithme du graphe Welsh Powell

Dans la théorie des graphes, la coloration des sommets est un moyen d'étiqueter chaque sommet individuel de telle sorte qu'aucun sommet adjacent n'a la même couleur. Mais nous devons trouver le nombre de couleurs dont nous avons besoin pour satisfaire la condition donnée. Il n'est pas souhaitable d'avoir une grande variété de couleurs ou d'étiquettes. Nous avons donc un algorithme appelé algorithme Welsh Powell qui donne le minimum de couleurs dont nous avons besoin. Cet algorithme est également utilisé pour trouver le nombre chromatique d'un graphe. Il s'agit d'une approche glouton itérative [21]. Les etapes de cet algoithme sont la suivante :

1. Trouvez le degré de chaque sommet.
2. Répertoriez les sommets par ordre décroissant de degrés.
3. Colorez le premier sommet avec la couleur 1.
4. Descendez dans la liste et colorez tous les sommets non connectés au sommet coloré, avec la même couleur.
5. Répétez l'étape 4 sur tous les sommets non colorés avec une nouvelle couleur, par ordre décroissant de degrés jusqu'à ce que tous les sommets soient colorés.

Et voilà le pseudo code d' Algorithme Welsh Powell ci-dessus dans la algorithme :

Input: G :Graph[V,E]
1. On note L la liste des sommets si casses suivant l'ordre décroissant de leur.
2. Degre :d(s1) d(s2)d(s3) : : d(sn).
3. Initialisation
4. L :liste des sommets dans l'ordre décroissant du degré.
couleur=0
while $L \neq \phi$ **do**
  couleur =couleur+1
  couleur(s)=couleur
  for *tout t dans L faire* **do**
    if $t \notin \Gamma s$ **then**
      couleur(t)=couleur
      $\Gamma s = \Gamma s \bigcup \Gamma t$
    end
  end
  Retirer les sommets colores de L
end

Algorithm 5: Algorithme Wesh et Powel [45].

Exemple 3.6.2. Le Figure 3.5 présente un exemple de coloration d'un graphe utilisant l'algorithme welsh powell. Les resultats de coloration sont aussi illustré dans le Tableau 3.1 :

Somment	A	B	C	D	E	F	G	H	I	J	K
Degré	2	2	1	4	2	2	3	5	3	3	5
(b)				Rouge	Rouge			Rouge			
(c)	verte		verte	Rouge	Rouge	verte		Rouge	verte		verte
(d)	verte	bleu		Rouge	Rouge		bleu	Rouge	verte	bleu	verte

TABLE 3.1 – Étapes de la agorithme d'exécution

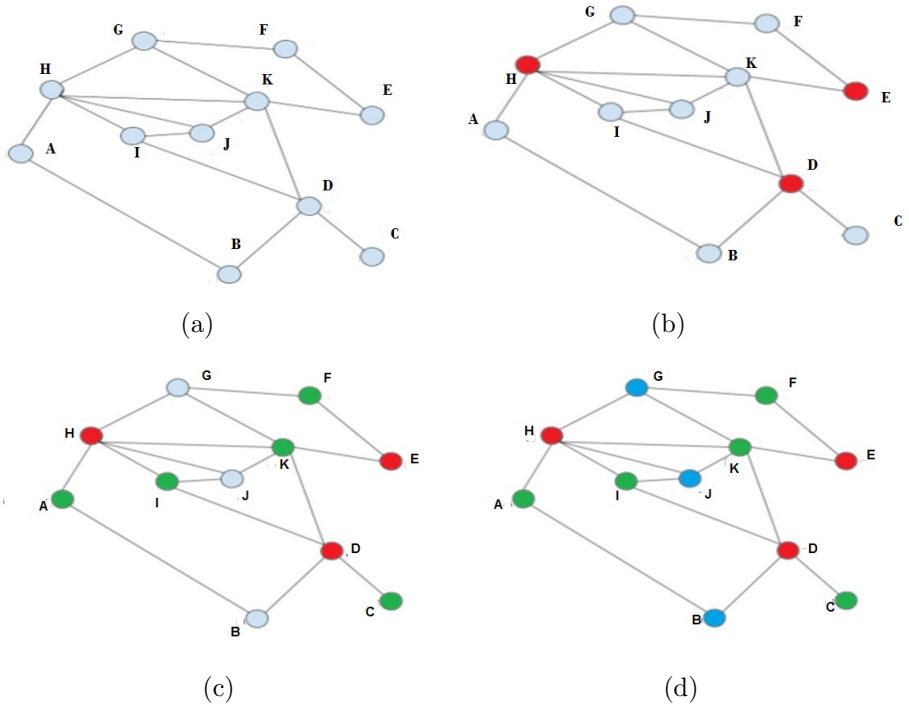


FIGURE 3.3 – Example Algorithme Welsh Powell

Maintenant, nous pouvons voir qu'en utilisant l'algorithme de Welsh Powell, nous pouvons colorer les sommets avec seulement 3 types de couleurs (le nombre chromatique de ce graphe est 3) ce qui est la solution optimale, puisque ce graphe contient au moins un triangle[21].

3.7 Proposition d'un nouvel algorithme de coloration des sommets d'un graphe sous Graphx

Dans cette section, nous proposons un algorithme de coloration parallèle de sommets d'un graphe sous Graphx, nommé CPSG (Coloration Parallèle de Sommets d'un Graphe). Cet algorithme est conçu pour les graphes non orienté. Dans ce qui suit, nous allons donner le pseudo code de l'algorithme proposé suivie par une description détaillée et un exemple illustratif.

3.7.1 Description de l'algorithme proposé

L'algorithme CPSG est un algorithme parallèle adapté à l'algorithme détaillé précédemment, l'algorithme de welsh et powell. CPSG est conçu pour l'environnement Graphx où nous avons basé sur les opérateurs de graphes, par exemple *aggregateMessages*, *distinct*...etc. Le pseudo code de l'algorithme CPSG est donnée dans l'algorithme 6.

Input: $graph1$: Graph direct
Output: $graph2$: Graph non direct
go←graph1.vertices.map{case (id, attr)≥(id, -1) }
graph4 ←Graph(go, graph1.edges)
d←graph4.degrees
gk2←graph4.outerJoinVertices(d)((vid,vd,u) ≥(-1, u.get))
vertex←gk2.vertices
graph2←Graph(vertex, graph4.edges ⋃ graph4.edges.reverse).cache()
$max \leftarrow graph4.degrees.reduce((a, b) => if(a._2 > b._2)\ a\ else\ b)$
$b \leftarrow max._2 + 1$
bkd←graph2.vertices
for $X \leftarrow 1\ to\ b$ **do**
  bkd←graph2.aggregateMessages(Coloration sommet max degre(i), reduce ())
  graph2←Graph(bkd, graph2.edges)
  bkd ←graph2.aggregateMessages(Colorer reste le sommet (i),reduce ())
  graph2←Graph(bkd, graph2.edges)

Algorithm 6: Algorithme CPSG (Coloration Parallèle de Sommets de Graphe) .

La première étape, nous ajoutons à chacune de sommet une variable *couleur* qui prend la valeur -1, d'ailleurs nous ajoutons à chaque sommet une variable appelée *degre* afin qu'il contienne le degré de chaque sommet. Ensuite, nous convertissons le graphe orienté en un graphe non orienté. Puis il utilise aggregateMessages qui fait appelle aux fonctions *Coloration sommet max degre*, *Colorer reste le sommet*, et *Reduce*. Le calcul se répète jusqu'à ce que tous les sommets soient colorés. Ces trois fonctions sont comme suit :

1. **Fonction Coloration sommet max degre** (voir Algorithme 7) : Dans cette fonction, le sommet envoie des messages à ses voisins. Si le sommet v a un degré supérieur à celui de son voisin u, ou si son degré est égal au degré de son voisin u et que l'ID du sommet v est plus petit que l'ID du voisin u, et si la couleur de u n'est pas égale à la couleur de v, il envoie à u un message contient le degré de son voisin u et sa couleur, de la forme $(coleur_voisin, degre_voisin)$. Et si c'est l'inverse, si la couleur se son voisin est égale à -1, elle lui envoie un message contenant -1, une couleur, de la forme suivante : $(coleur, -1)$. Sinon, le sommet v envoi un message contenant le degré et la couleur de son voisin, sous de la forme $(coleur_vosin, degre_vosin)$.
2. **Fonction Colorer reste le sommet** (voir Algorithme 8) : Dans cette fonction, le sommet envoie des messages à ses voisins. Si le sommet v a un degré intérieur à celui de son voisin u, ou si son degré est égal au degré de son voisin u et que l'ID du sommet v est plus grand que l'ID du voisin u, et si la couleur de u n'est pas égale à la couleur de v, il envoie à u un message contient le degré de son voisin u et sa couleur, de la forme $(coleur_voisin, degre_voisin)$. Et si c'est l'inverse, si la couleur se son voisin est égale à -1, elle lui envoie un message contenant -1, une couleur, de la forme suivante :

$(coleur, -1)$. Sinon, le sommet v envoi un message contenant le degré et la couleur de son voisin, sous de la forme $(coleur_vosin, degre_vosin)$.

3. **Fonction reduce** (voir Algorithme 9) : cette fonction est conçu pour traiter les messages reçus par un sommet v. La fonction reduce prend les messages reçus, les compare entre eux, et ensuite choisi le plus petit messages reçu. Par la suite selon ce message le sommet v, change sa couleur et son degré.

Function `Coloration sommet max degre`(*int couleur*)
- **for** *triplet* $\in$ *à tout les arêtes du sommet* **do**
  - **if** $(((triplet.srcAttr._2 > triplet.dstAttr._2)||((triplet.srcAttr._2 == triplet.dstAttr._2)\ \&\ (triplet.srcId < triplet.dstId))\ \&\ (triplet.dstAttr._1! = (couleur)))$ **then**
    - $triplet.sendToDst((triplet.dstAttr._1, triplet.dstAttr._2))$
  - **else**
    - **if** $triplet.dstAttr._1. == (-1)$ **then**
      - $triplet.sendToDst((couleur, -1))$
    - **else**
      - $triplet.sendToDst((triplet.dstAttr._1, triplet.dstAttr._2))$

Algorithm 7: Fonction Coloration sommet max degre.

Function `Colorer reste le sommet` (*int couleur*)
- **for** *triplet* $\in$ *à tout les arêtes du sommet* **do**
  - **if** $(((triplet.srcAttr._2 < triplet.dstAttr._2)||((triplet.srcAttr._2 == triplet.dstAttr._2)\ \&\ (triplet.srcId > triplet.dstId))\ \&\ (triplet.dstAttr._1! = (couleur)))$ **then**
    - $triplet.sendToDst((triplet.dstAttr._1, triplet.dstAttr._2))$
  - **else**
    - **if** $triplet.dstAttr._1. == (-1)$ **then**
      - $triplet.sendToDst((couleur, -1))$
    - **else**
      - $triplet.sendToDst((triplet.dstAttr._1, triplet.dstAttr._2))$

Algorithm 8: Fonction Colorer reste le sommet.

Function `reduce ()`
- recevoir des messages msg1,msg2
- **if** $(msg1._1 > msg2._1)$ **then**
  - return msg2
- **else**
  - return msg1

Algorithm 9: Fonction reduce.

3.7.2 Déroulement de l'algorithme proposé sur un exemple

Exemple 3.7.1. Appliquant l'algorithme CPSG sur le graphe de la Figure 3.4. Le résultat d'exécution de cet algorithme ce graphe est présenté par la Figure 3.5(e). Le déroulement de

cet algorithme sur ce graphe est comme suit :

- **Étape 1(voir la Figure 3.5(a)) :** chaque sommet envoie des messages à ses voisins, comme illustré par la Figure 3.5(a). Par la suite, chaque sommet reçoit les messages envoyés par ses voisins. Dans cette étape, le sommet 1 est le seul qui changera la valeur de la couleur de -1 à 1, puisque il a le plus grand degré par rapport à ses voisins. De plus, il change ses valeur de degré en -1, en fonction des messages qu'il a reçu de ses voisins, et selon le plus petit message qu'il a reçu de ses voisins.
- **Étape 2(voir Figure 3.5(b)) :** chaque sommet envoie des messages à ses voisins, comme illustré par la Figure 3.5(b). Par la suite, chaque sommet reçoit les messages envoyés par ses voisins. Dans cette étape, le sommet 5 est le seul qui changera la valeur de sa couleur de -1 à 1, puisque il a le plus petit degré par rapport à ses voisins. De plus, il change sa valeur de degré en -1, en fonction des messages qu'il reçu de ses voisins, et selon le plus petit message qu'il a reçu de ses voisins.
- **Étape 3(voir Figure 3.5(c)) :** chaque sommet envoie des messages à ses voisins, comme illustré par la Figure 3.5(c). Par la suite, chaque sommet reçoit les messages envoyés par ses voisins. Dans cette étape, le sommet 2 et le sommet 4 changeront la valeur de la couleur de -1 à 2, puisque ils ont le plus grand degré par rapport à ses voisins. De plus, ils changent ses valeur de degré en -1, en fonction des messages qu'ils ont reçu de leur voisins, et selon le plus petit message qu'ils ont reçu de leur voisins.
- **Étape 4(voir Figure 3.5(d)) :** chaque sommet envoie et reçoit des messages de ses voisins. Mais, dans cette étape il n'y a aucun changement dans l'état des sommets.
- **Étape 5(voir Figure 3.5(e)) :** chaque sommet envoie des messages à ses voisins, comme illustré par la Figure 3.5(e). Par la suite, chaque sommet reçoit les messages envoyés par ses voisins. Dans cette étape, le sommet 3 est le seul qui changera la valeur de la couleur de -1 à 3, puisque il est le seul sommet non coloré dans l'ensemble de ses voisins. Donc, il change sa valeur de degré en -1, en fonction des messages qu'il reçu de ses voisins, et selon le plus petit message qu'il a reçu de ses voisins.

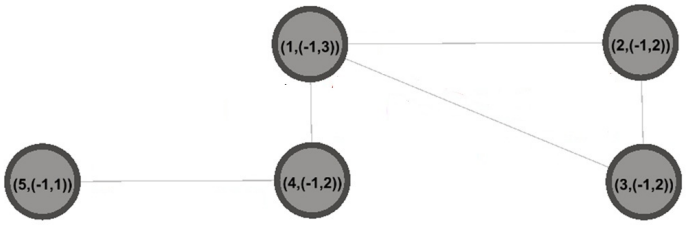
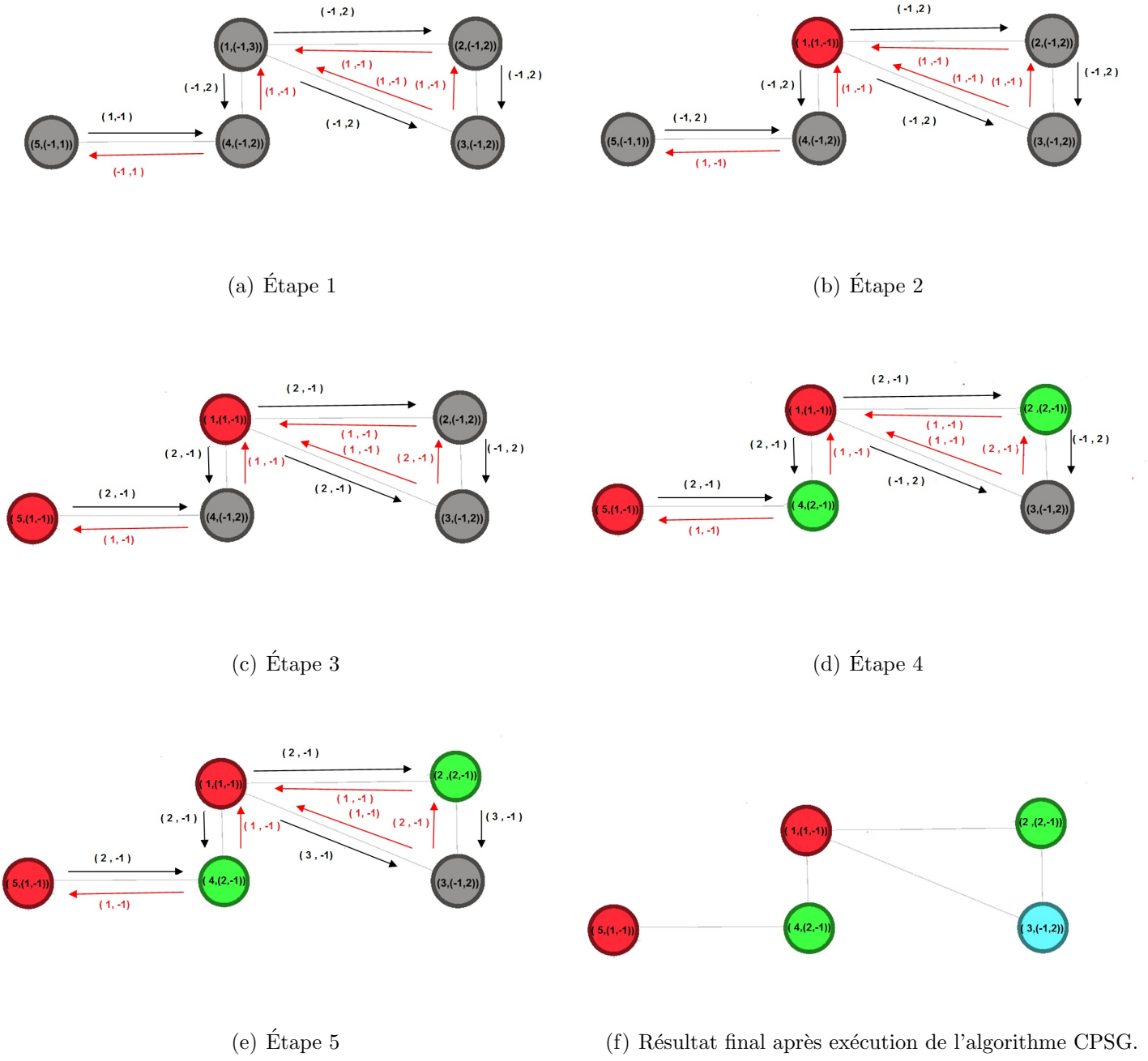


FIGURE 3.4 – Exemple d'un graphe.



(a) Étape 1

(b) Étape 2

(c) Étape 3

(d) Étape 4

(e) Étape 5

(f) Résultat final après exécution de l'algorithme CPSG.

FIGURE 3.5 – Exemple de l'algorithme CPSG dans Graphx

3.8 Conclusion

Le problème de coloration est un problème fondamental dans la théorie de graphes. Des exemples d'applications peuvent être trouvés dans la conception et l'analyse du réseau, les sciences sociales, ou bioinformatique . Dans ce chapitre nous avons vu quelques algorithmes de coloration de sommets de graphe. Ce algorithmes sont séquentiels des algorithmes sont heuristiques bien définit tels que algorithme Glouton, algorithme Welsh et Powell et algo-

rithme DSATURS. Par la suite, en se basant sur ces derniers nous avons proposé un nouvel algorithme parallèle pour la coloration de sommets des graphes nommé CPSG.

Dans le suivant chapitre, nous allons réaliser l'implémentation de l'algorithme proposé sous Graphx suivi par des évaluations expérimentales.

Chapitre 4

Mise en oeuvre et évaluation de l' algorithme

4.1 Introduction

Dans le chapitre précédent, nous avons proposé un nouveau algorithme parallèle pour la coloration de larges graphes sous Graphx, nommé CPSG. Cet algorithme est une adaptation de l'algorithme séquentiel welsh powell pour qu'il soit capable de s'exécute dans un environnement parallèle. Dans ce chapitre, nous allons implémenter et tester cet algorithme sur des benchmarks de données. Nous allons commencer par la préparation d'un environnement d'exécution équipé d'un système d'exploitation Windows 7, JDK, Eclipse, Spark/Graphx, et Scala. Par la suite, nous allons réaliser des expérimentations afin de tester la scalabiité et la qualité de solution de l'algorithme proposé.

4.2 Configuration de l'environnement de développement

Indépendamment du système d'exploitation Windows ou Linux, l'idée d'un environnement de développement Spark est la même, basée sur Eclipse, et développée en language Java, Scala ou Python. Avant l'installation, nous devons préparer de l'environnement JDK (Java Development Kit), Scala ou Python à l'avance, puis téléchargez et installez le plugin Scala ou Python dans Eclipse (Spark supporte Java, Python et d'autres langues). Notre environnement de développement, est illustré dans la Figure 4.1. L'installation de Spark nécessite l'installation et la configuration d'une version récente de Java (la version par défaut 8). Ensuite nous avons installé Graphx qui est à base de Spark, par la suite nous avons installé et configuré Eclipse avec Spark.

Nous avons utilisé le langage de programmation Scala, qui est un language de programmation à usage général créé en 2001 afin de fournir un soutien à la fois à la programmation orientée objet et à la programmation fonctionnelle. Il a été fortement influencé par Java et conçu pour remédier à certains de ses inconvénients. Le language Scala commence à être utilisé dans l'industrie, plusieurs entreprises ont annoncé le passage de certaines de leurs

applications au langage Scala, comme Twitter le Tabeau 4.1.

Logiciel	Version	Logiciel prérequis	Remarque
System d'exploitation	Windows 7		64bit
Spark	Spark1.6.1	Windows 7 Java version 8 (jdk8)	`https://spark.apache.org/news/spark-2-4-6.html`
Eclipce	Eclipce v4.5.2	Windows7	`https://www.eclipse.org/downloads/packages/release`
Scala	Scala v2.11.8	Windows7	`https://www.scala-lang.org/download/2.11.8.html`

TABLE 4.1 – Tableau descriptif de l'environnement du travail

FIGURE 4.1 – Pile des logiciels

Scala, Eclipse, Spark sont tous les deux sont installés dans machine physique contenant 2GO RAM et CPU 2.10 Hz dans un espace de stockage de 250 Go comme indiqué dans les Tableau4.1.

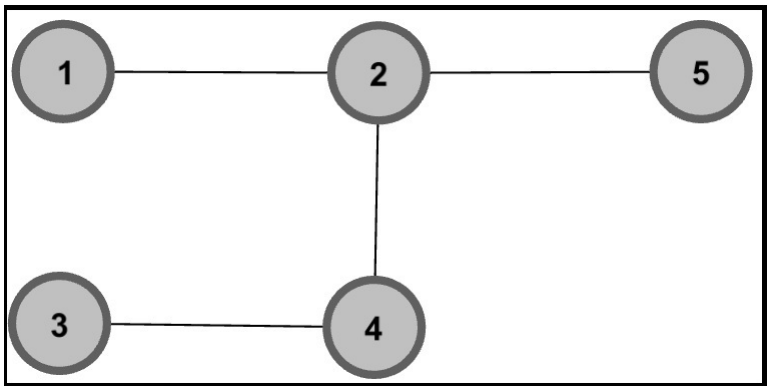
	PC	
Machine physique	RAM	2 GO
	Espace disque	250 GO
	Processeur	Inter (R) CPU B950 @2.10GHz

TABLE 4.2 – Tableau descriptif de matérielle et des machines

4.3 Exemples d'exécution de l'algorithme

Dans cette section, nous présentons deux exemples d'exécution du l'algorithme proposé utilisant les graphes de la Figure 4.2 et la Figure 4.3.

Exemple 4.3.1. Soit le graphe de la Figure 4.2(a). La liste d'adjacence de ce graphe est présenté dans la Figure 4.2(b). Le résultat d'exécution de l'algorithme CPSG est illustré par la Figure 4.2(d) .



(a) Exemple d'un graphe

*Sortie.txt *k1.txt

```
1# Directed graph (each unordered pair of nodes is saved once): k1.txt
2# Paper citation network of Arxiv High Energy Physics Theory category
3# Nodes: 5 Edges: 4
4# FromNodeId    ToNodeId
51   2
62   5
72   4
83   4
```

(b) Liste d'adjacence du graphe du Figure 4.2(a)

*Sortie.txt k1.txt

```
11   2
22   1
33   1
44   2
55   2
```

(c)

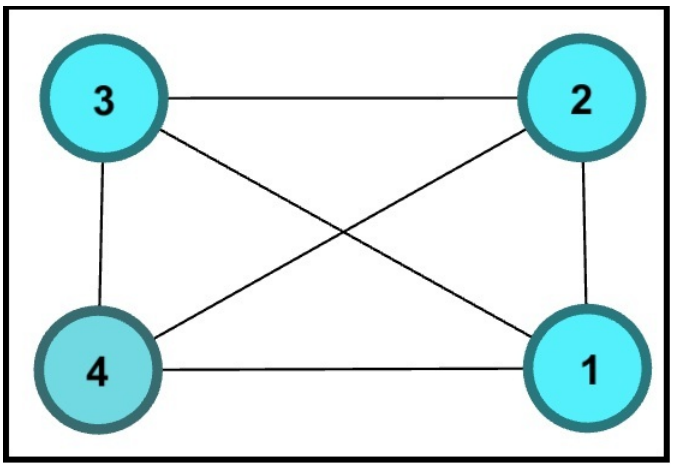
```
**********la couleur de chaque sommets **********
(4,(2,-1))
(1,(2,-1))
(3,(1,-1))
(5,(2,-1))
(2,(1,-1))
**********end**********
************************************************

le nombre de couleur utilisé : 2
************************************************
```

(d) Résultat de l'exécution de l'algorithme sur le graphe de le Figure 4.2(a)

FIGURE 4.2 – Exécution de graphe 1

Exemple 4.3.2. Soit le graphe de la Figure 4.3(a). La liste d'adjacence de ce graphe est présenté dans la Figure 4.3(b). Le résultat d'exécution de l'algorithme CPSG est illustré par la Figure 4.3(d).



(a) Exemple d'un graphe

*Sortie.txt *k1.txt

```
1# Directed graph (each unordered pair of nodes is saved once): k1.txt
2# Paper citation network of Arxiv High Energy Physics Theory category
3# Nodes: 5 Edges: 4
4# FromNodeId    ToNodeId
51   2
62   3
73   4
84   1
91   3
102  4
```

(b) Liste d'adjacence du graphe du Figure 4.3(a)

*Sortie.txt *k1.txt

```
1 1   1
2 2   2
3 3   3
4 4   4
5
```

(c)

Tasks Console Progress Coverage

```
<terminated> project$ [Scala Application] C:\Program Files\Java\jre1.8.0_141\bin\java
**********etape2**********4
**********la couleur de chaque sommets **********
(4,(4,-1))
(1,(1,-1))
(3,(3,-1))
(2,(2,-1))
**********end**********
**************************************************
le nombre de couleur utilisé : 4
**************************************************
```

(d) Résultat de l'exécution de l'algorithme sur le graphe de le Figure 4.3(a)

FIGURE 4.3 – Exécution de graphe 2

4.4 Benchmark de graphe de données utilisés

Pour bien tester et évaluer l'algorithme implémenté, nous avons généré trois types de graphes (voir le Tableau 4.3) en utilisant le logiciel Gephi qui génère de graphe aléatoires de façon aléatoire. Pour générer ces graphes nous avons, a chaque fois, changé le degré maximum des sommets et le nombre des arêtes de façon que nous avons obtenus 3 types de graphes : petit, moyen et large. Chaque type de graphe englobe 5 graphes contiennent un nombre d'arêtes varié de 100 à 2000.

	Nombre des arcs	Nombre des sommet	Max degrée
Petit-100	90	100	3
Petit-500	129	500	4
Petit-100	524	1000	7
Petit-1500	1099	1500	7
Petit-2000	1894	2000	8
Moyen-100	330	100	5
Moyen-500	624	500	10
Moyen-1000	2397	1000	13
Moyen-1500	5634	1500	20
Moyen-2000	9957	2000	21
Large-100	241	100	7
Large-500	1224	500	13
Large-1000	4972	1000	21
Large-1500	11260	1500	31
Large-2000	20012	2000	37

TABLE 4.3 – Tableau descriptif des graphes générés

4.5 Expérimentation et analyse des résultats :

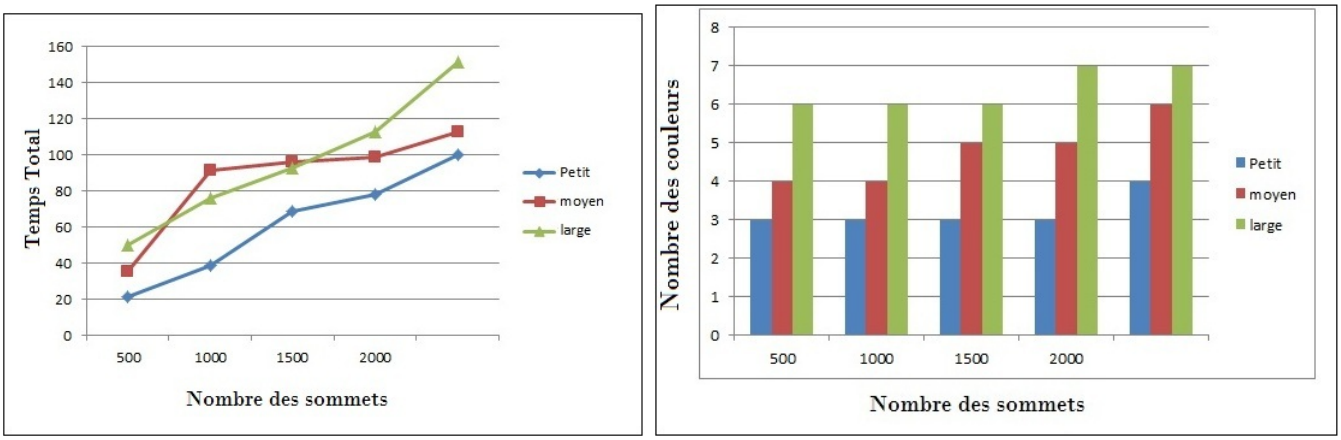
4.5.1 Test de l'algorithme CPSG

Dans cette section nous avons testé l'algorithme CPSG sous Graphx. Pour ce faire nous avons utilisé les trois ensembles de données illustrées par le Tableau 4.4 , chaque fois on augmente le nombre des sommets et le nombre des arcs puis on examinant le temps total et le nombre des couleurs :

Les résultats obtenus sont montrés par le tableau et illustres dans le Figure 4.4. A partir de ces résultats nous remarquons que l'algorithme CPSG est scalable et le temps d'exécution augmente linéairement avec la taille des données d'entrée (nombre de sommets).

	Nombre des arcs	Nombre des sommet	Nombre des couleurs
Petit-100	90	100	3
Petit-500	129	500	3
Petit-100	524	1000	3
Petit-1500	1099	1500	3
Petit-2000	1894	2000	4
Moyen-100	330	100	4
Moyen-500	624	500	4
Moyen-1000	2397	1000	5
Moyen-1500	5634	1500	5
Moyen-2000	9957	2000	6
Large-100	241	100	6
Large-500	1224	500	6
Large-1000	4972	1000	6
Large-1500	11260	1500	7
Large-2000	20012	2000	7

TABLE 4.4 – Tableau des résultats du générateur de graphes



(a) Temps d'exécution par rapport au nombre de sommets

(b) Nombre des couleurs utilisés par chaque graphe

FIGURE 4.4 – Test de scalabilité algorithme CPSG.

4.6 Conclusion

Dans ce chapitre nous avons décrit l'environnement du travail et les déférents benchmarks de donnée utilisées pour valider l'algorithme CPSG. Par la suite, nous avons testé la scalabilité des l'algorithme sous Graphx en terme du temps d'exécution et le qualité de la solution (nombre de couleurs). Les résultats obtenus montrent que l'algorithme proposé est scalable mais il ne donne pas toujours la solution optimale mais aussi il fonctionne bien même sur des graphes denses dans un temps raisonnable.

Conclusion générale et perspectives

Il a été reconnu que les graphes gagnent en popularité constituant une partie intégrante de nombreuses applications du monde réel. En raison de l'importance des graphes, de nombreuses études ont beaucoup investi dans les technologies de l'exploitation des graphes de données. Ces efforts ont abouti à un nombre croissant d'approches d'interrogation de graphes. L'objectif de notre projet était de proposer un algorithme de graphe séquentiel, l'adapter et l'implémenter dans un environnement parallèle. Parmi les environnements parallèles de traitement de larges graphes les plus efficaces nous avons choisi Graphx.

Pour atteindre cet objectif, nous avons étudié les algorithmes de coloration des sommets dans un graphe non orienté les plus célèbres qui se trouvent dans la littérature. Par la suite, nous avons choisi l'heuristique Welsh et Powell. L'algorithme choisi est adapté et implémenté sous Graphx et testé sur des benchmarks de données de graphes. Les résultats obtenus montrent que l'heuristique implémentée permet une coloration de toutes les sommets dans un graphe donné. Ainsi, cette heuristique est très rapide.

Ce projet ou cette expérience a été pour nous une occasion pour découvrir un environnement aussi complexe de la mise en place d'une solution Cloud, ce qui nous a permis d'avoir des connaissances dans le domaine de la virtualisation, Cloud Computing et ses nouvelles technologies de calcul distribué, ainsi que les graphes parallèles.

Finalement, comme perspectives on peut proposer :

- Étude expérimentale de l'algorithme proposé sur des larges clusters avec des larges graphes.
- Valider l'algorithme implémenté sur une plate forme Cloud publiques tels qu' Amazon.
- Adapter d'autres algorithmes de coloration de sommets d'un graphe.
- Réaliser des expérimentations afin de comparer l'algorithme proposé avec d'autres algorithmes de graphes parallèle ou séquentiel existant dans la littérature, utilisant des graphes de données réels.
- Une autre étude pourrait évaluer l'évolutive des améliorations mises en oeuvre pour déterminer la manière dont elles fonctionnent sur des jeux de données plus importants ou sur différents types de graphes.

1. **Problèmes rencontrés :**

Ce projet était très compliqué, surtout il était pour nous une nouvelle expérience de programmation avec un nouvel environnement de parallélisme. Donc nous avons rencontre de nombreux problèmes lors de l'installation et la configuration des outils qui nécessitent un débit élevé, notamment en ce qui concerne l'installation de Graphx et le manque de la documentation.

Bibliographie

[1] Understanding MapReduce in Hadoop, May 2021. https://www.section.io/engineering-education/understanding-map-reduce-in-hadoop (le consulter 27/05/2021).

[2] Mouhamed Mourchid Adio Adegbindin. un algorithme constructif efficace pour le pour le problème de coloration de graphe. Master's thesis, ÉCOLE POLYTECHNIQUE DE MONTRÉAL, 2013.

[3] Khaled Ammar and M. Tamer Özsu. Experimental analysis of distributed graph systems. *Proceedings of the VLDB Endowment*, 11(10) :1151–1164, Jun 2018.

[4] Tim Hunter Ankur Dave, Joseph Bradley and Xiangrui Meng. Introducing Graph Frames, March 2016. https://databricks.com/blog/2016/03/03/introducing-graphframes.html (Consulter 5/04/2021).

[5] Khadidjatou bamba. Comprendre le big data. Technical report, 2013.

[6] Jean-Yves Baudot. Graphes orientés, Avril 2021. http://www.jybaudot.fr/Optimisations/orientes.html (le consulter 3/06/2021).

[7] Maha boussabbeh. *Preuves d'algorithmes distribués par composition et raffinement.* PhD thesis, de l'université l'université de bordeaux et de l'université de bordeaux et de l'université de sfax, 2017.

[8] Venkatesan T. Chakaravarthy, Fabio Checconi, Fabrizio Petrini, and Yogish Sabharwal. *Scalable Single Source Shortest Path Algorithms for Massively Parallel Systems.* May 2014.

[9] Aditya Chatterjee. A quick introduction to Google's Pregel graph processing system. *Medium*, May 2018.

[10] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges : graph processing at facebook scale. *ResearchGate*, pages 1804–1815, Jan 2015.

[11] Samir El Houari Consultant. Hadoop - composants, May 2021. https://sites.google.com/site/selhconsultant/hadoop-composants (le consulter 27/05/2021).

[12] Beyond corner. Apache Spark components - Spark GraphX - beyond corner, Jul 2021. https://beyondcorner.com/learn-apache-spark/apache-spark-components-spark-graphx (le consulter 4/06/2021).

[13] Apache dans github.com. Spark GitHub, Aug 2021. https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples (le consulter 28/06/2021).

[14] DataFlair. Spark GraphX features - An introductory guide - DataFlair, Sep 2018. https://data-flair.training/blogs/apache-spark-graphx-features (le consulter 5/06/2021).

[15] Ripon Patgiri Dhananjay Kumar Singh. Big graph : tools techique issues , challenges and future directions, 2016. https://airccj.org/CSCP/vol6 (le consulter 25/04/2021).

[16] Arturo Diaz-Perez, Alberto Garcia-Robledo, and Jose-Luis Gonzalez-Compean. Graph processing frameworks. In *Encyclopedia of Big Data Technologies*, pages 875–883. Springer, Cham, Switzerland, Feb 2019.

[17] Alain Fernandez. Qu'est-ce que Hadoop ? *Management et Performance, piloter*, Dec 2018.

[18] Alain Fernandez. Qu'est-ce que le big data ? *Management et performance, piloter*, novembre 2020.

[19] G.Dilan. Hadoop - quels sont les inconvénients de mapreduce ?, September 2013. https://askcodez.com/quels-sont-les-inconvenients-de-mapreduce.html (le Consulter 27/05/2021).

[20] G.Dilan. hadoop - quels sont les inconv énients de mapreduce ?, Mar 2019. https://askcodez.com/quels-sont-les-inconvenients-de-mapreduce.html (le consulter 27/05/2021).

[21] Geeksforgeeks. Welsh Powell graph colouring algorithm - GeeksforGeeks, Octobre 2019. https://www.geeksforgeeks.org/welsh-powell-graph-colouring-algorithm (le consulter 30/05/2021).

[22] Giraph. Giraph - Welcome To Apache Giraph !, Aug 2020. https://giraph.apache.org (le consulter 27/05/2021).

[23] IGI Global. What is Pregel | IGI Global, Aug 2021. https://www.igi-global.com/dictionary/pregel/62442 (le consulter 15/04/2021).

[24] Nicolas Grevet. Le cloud computing : évolution ou révolution ? Master's thesis, M2IRT 2009, spécialité SIIC, 2009.

[25] Gulati, S. and Kumar, S. *Apache Spark 2.x for Java Developers*. Packt Publishing, 2017.

[26] Hadi Hashem. *Modélisation intégratrice du traitement Big Data*. PhD thesis, l'universite paris-saclay.

[27] Boukhatem Fatima Zohra Hadjari Imane, Benbachir Meriem. Big data : conceptions, architectures, fonctionnements et applications. Master's thesis, Université Abou Bakr Belkaid - Tlemcen, 2016.

[28] Halluxio. Introduction to Hadoop distributed File System (HDFS) Alluxio, May 2021. `https://www.alluxio.io/learn/hdfs` (le consulter 27/05/2021).

[29] Minyang Han, Khuzaima Daudjee, Khaled Ammar, M. Tamer Ö zsu, and Tianqi Jin. An experimental comparison of Pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 7(12) :1047–1058, Aug 2014.

[30] Mark Horeni. Graphx and spark. Technical report, 2016.

[31] Intellipaat. Algorithm - Intellipaat, December 2020. `https://intellipaat.com/blog/tutorial/mapreduce-tutorial/algorithm-mapreduce`(le consulter 28/05/2021).

[32] Intellipaat. Definition of MapReduce - Intellipaat, December 2020. `https://intellipaat.com/blog/tutorial/mapreduce-tutorial/definition-of-mapreduce` (le consulter 28/06/ 2021).

[33] Hausmane Issarane. 6 Logiciels Big Data Open Source - analytics & insights, Feb 2019. `https://analyticsinsights.io/6-logiciels-big-data-open-source` (le consulter 27/05/2021).

[34] Hausmane Issarane. Apache Hadoop : avantages et inconv énients - Analytics & Insights, Feb 2019. `https://analyticsinsights.io/apache-hadoop-avantages-et-inconvenients` (le consulter 27/05/2021).

[35] JDN. Hadoop et son écosystème, juillet 2013. `https://www.journaldunet.com/web-tech/developpeur/1125768-panorama-des-solutions-de-big-data/1125771-hadoop-et-son-ecosysteme` (le consulter 27/05/2021).

[36] JDN. E-business, FinTech, Big Data, IoT, tendances média, d écideurs.., May 2021. `https://www.journaldunet.com` (le consulter 27/05/2021).

[37] Rohan Joseph. what is spark, 2021. `https://chartio.com/learn/data-analytics/what-is-spark` (le consulter 26/06/2021).

[38] Bastien L. MapReduce : tout savoir sur le framework Hadoop de traitement Big Data - LeBigData.fr, Octobre 2017. `https://www.lebigdata.fr/mapreduce-tout-savoir` (le consulter 27/05/2021).

[39] Bastien L. Hadoop – Tout savoir sur la principale plateforme Big Data LeBigData.fr, Mar 2018. `https://www.lebigdata.fr/hadoop` (le consulter 27/05/2021).

[40] LaMaStEx. Introduction to graph frames scalable data science, Jul 2021. https://lamastex.gitbooks.io/scalable-data-science/content/db/week8/15_GraphX/026_GraphFramesUserGuide.html (Consulter 5/06/2021).

[41] Lamkien. Cloud Computing en Afrique Situation et perspectives viewCloud Computing en afrique : Situation et perspectives Cloud Computing en Afrique : Situation et perspectives Regulation Global. *Vdocuments*, Apr 2018.

[42] Ryan P. Langewisch. A performanc study of an implementation of the pushrelabel maximum flow algorithm in apache spark. Master's thesis, the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment, 2015.

[43] Erell Le Gall. Qu'est-ce que la virtualisation en informatique ?, May 2021. https://blog.hubspot.fr/marketing/virtualisation-informatique (le consulter27/05/2021).

[44] R. M. R. Lewis. *A guide to graph colouring*. Jan 2016.

[45] R.M.R. Lewis. *A Guide to Graph Colouring : Algorithms and Applications*. Springer International Publishing, 2015.

[46] Rédha LOUCIF. Parallélisation d'algorithmes d'optimisation combinatoire. Master's thesis, Université colonel HADJ LAKHDAR ?BATNA, 2014.

[47] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed GraphLab : A Framework for Machine Learning in the Cloud. *arXiv*, Apr 2012.

[48] Yi Lu, James Cheng, Da Yan, and Huanhuan Wu. Large-scale distributed graph computing systems. *Proceedings of the VLDB Endowment*, 8(3) :281–292, Nov 2014.

[49] M. Malak and R. East. *Spark GraphX in Action*. Manning Publications, 2016.

[50] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, and Grzegorz Czajkowski. Pregel : a system for large-scale graph processing. *Pregel : a system for large scale graph processing*, page 48, Jan 2009.

[51] Teddy Mantoro, Media A. Ayu, and Amir Borovac. GPS Based Tracking Framework for Walking Pedestrian. In *Informatics Engineering and Information Science*, pages 337–348. Springer, Berlin, Germany, Nov 2011.

[52] Nick McKenna. The relationship between IoT, Big Data and Cloud Computing, May 2021. https://www.mckennaconsultants.com/relationship-between-iot-big-data-and-cloud-computing / (le consulter 27/05/ 2021).

[53] D. Müller. *Introduction à la théorie des graphes*. Les Cahiers de la CRM. Commission romande de mathématique, 2011.

[54] Capital one. What is a cluster ? an introduction to clustering in the cloud | Capital One, July 2020. https://www.capitalone.com/tech/cloud/what-is-a-cluster (le consulter 27/05/2021).

[55] Raj R Parmar, Sudipta Roy, Debnath Bhattacharyya, Samir Kumar Bandyopadhyay, and Tai-Hoon Kim. Large scale encryption in the hadoop environment : challenges and solutions. *IEEE Access*, 5 :7156–7163, 2017.

[56] Srini Penchikala. Facebook's comparison of apache Giraph and Spark GraphX for Graph data processing. *InfoQ*, Dec 2016.

[57] Srini Penchikala. Big Data processing using Apache Spark - part 6 : graph data Analytics with Spark Graphx. *InfoQ*, Mar 2017.

[58] Prasenjit Choudhury Pijush Kanti, Dhananjay Kumar Singh. Big graph analytics : techniques, tools, challenges, and applications, 2018. https://www.researchgate.net/profile/Pijush-Dutta-Pramanik/publication/327802620_Big_Graph_Analytics_Techniques_Tools_Challenges_and_Applications (le consulter 28/06/ 2021).

[59] Matroid Reza Zadeh and Stanford. Distributed algorithms , Sept 2016. https://www.coursehero.com/file/16092746/cme323-lec9 (le consulter 30/05/2021).

[60] Benoît Robillard. Complexitéde la coloration avec préférences dans les graphes de conitbipartis. Technical report, Lab oratoireCEDRIC,292rueSaint- Martin,75141Pariscedex03, 2020.

[61] Seyed H Roosta. Parallel graph algorithms. In *Parallel Processing and Parallel Algorithms*, pages 259–318. Springer, New York, NY, New York, USA, 2000.

[62] Sherif Sakr. Processing large-scale graph data : a guide to currenttechnology. Technical report, National ICT Australia, 10 June 2013.

[63] SAS. What is Hadoop ?, Aug 2019. https://www.sas.com/nl_nl/insights/big-data/hadoop.html#hadoopusers (le consulter 27/05/2021).

[64] Gril Dylan Schuimer jordan. Partie II : la coloration, Décmbre 2015. https://m6colorationgraphes.wordpress.com/2015/11/12/lhistoire-de-la-coloration-des-graphes (le consulter 19/05/2021).

[65] R. Shaposhnik, C. Martella, and D. Logothetis. *Practical graph analytics with Apache Giraph*. Apress, 2015.

[66] Bledi Shaqiri. The 5V of Big Data characteristics, Novembre 2017. https://www.researchgate.net/figure/The-5V-of-Big-Data-Characteristics_fig1_321050765 (le consulter 19/06/2021).

[67] Rohit Sharma. Top 6 Major challenges of Big Data & simple solutions to solve them | upGrad blog, May 2020. https://www.upgrad.com/blog/major-challenges-of-big-data (le consulter 27. May 2021).

[68] Spark. GraphX - Spark 3.1.2 documentation, May 2021. https://spark.apache.org/docs/latest/graphx-programming-guide.html (le consulter 28/05/2021).

[69] Abbas Taher. Pagerank-example-spark2.0-deep-dive, Aug 2021. https://github.com/abbas-taher/pagerank-example-spark2.0-deep-dive/blob/master/SparkPageRank.scala (le consulter 2/07/2021).

[70] Maher Turifi. *Optimisation techniques for finding connected components in large graphs using GraphX*. PhD thesis, University of Salford School of Computing, Science and Engineering, 2017.

[71] Tutorialspoint. Cloud Computing Tutorial - Tutorialspoint, May 2021. https://www.tutorialspoint.com/cloud_computing/index.htm (le consulter 27/05/2021).

[72] Unit.eu. Introduction à la théorie des graphes - Coloration des graphes, Jan 2017. http://www.unit.eu/cours/EnsROtice/module_de_base_voo7/co/coloration.html / (le consulter 1/06/2021).

[73] Qiumin Xu, Hyeran Jeon, and Murali Annavaram. Graph processing on GPUs : where are the bottlenecks ? *ResearchGate*, pages 140–149, Oct 2014.

[74] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. Pregel algorithms for graph connectivity problems with performance guarantees. *Proceedings of the VLDB Endowment*, 7(14), Oct 2014.

[75] Saadna Yassmina. Cours big data et deep learning. Technical report, Universite Batna 2 faculté mathématique et informatique Département de Mathématique Master1 SAD, 2020.

[76] YouScribe. Le Cloud Computing : évolution ou révolution ?, May 2021. https://www.youscribe.com/BookReader/Index/408177/?documentId=379663 (le consulter 27/05/2021).

RÉSUMÉ

Aujourd'hui, les graphes sont devenus un moyen de stockage des informations et des données. ils sont partout. Ils sont utilisés pour modéliser divers types de réseaux : réseau internet, réseau de transport, réseaux sociaux, etc. Cependant, ces graphes peuvent être de grande taille (Big-graphs) et il devient difficile de les traiter efficacement sur une seule machine. Cela signifie qu'il ya un besoin pour les algorithmes qui peuvent être facilement parallélisés. Par conséquent, pour simplifier la programmation des algorithmes de graphe dans un environnement distribué, un certain nombre des outils open source ont été développé, tels que : Pregel, Giraph, Graphx, GraphLab, etc. La plupart de ces systèmes s'appuient sur le modèle BSP, popularisé par le projet Pregel de Google. Ils sont conçus pour exécuter d' algorithmes de graphes itératifs à travers des clusters de machines.

Dans ce mémoire nous avons basé sur le framwork Graphx. Graphx est une API de calcul de graphes construite sur Apache Spark. Il est un moteur rapide et agrégé pour le traitement de données à grande échelle dans le cloud. Alors que la popularité de Spark et Graphx augmente, la technologie relativement jeune n'a pas encore exploré les problèmes de graphes existants dans ce domaine. Afin d'examiner et de mieux comprendre les capacités de Graphx, ce memoire aborde le paradigme avec l'intention de mise en oeuvre du problème de colorisation, qui est un problème graphe complexe. .

Mots clés : Cloud Computing, Pregel, Graphx, Spark, Big-Graph, Algorithme de graphes.

ABSTRACT

Today, graphs have become a way to store information and data, they are everywhere. They are used to model various types of networks : Internet network, transport network, social networks, etc. However, these graphs can be large (Big-graphs) and it becomes difficult to treat them effectively on a single machine. This means that there is a need for algorithms that can be easily parallelized. Therefore, to simplify the programming of graph algorithms in a distributed environment, a number of open-source tools have been developed, such as : Giraph, Graphx, Graphlab, etc. Most of these systems are based on the BSP model, popularized by the Google's Pregel project. They are designed to execute the algorithms of iterative graphs through machines. In this memory we use the Framwork Graphx. Graphx is a graph compute API built on top of Apache Spark, it is a fast and aggregated engine for large scale data processing in the cloud. While the popularity of Spark and Graphx increases, relatively young technology has not yet explored the graph problems that exist in this area. In order to review and better understand Graphx's capabilities, this thesis addresses the framework with the intention of implementation of a colorization problem, which is a complex graph problem.

Key words : Cloud Computing, Pregel, Graphx, Spark, Big-Graph, Graph Algorithm .