République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
Et de la Recherche Scientifique
Université de - Jijel - Mohamed Seddik 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 graphes sous Spark/Graphx : Algorithmes et Implémentation

Réalisé par :

- Ammara Riad
- Khelili Houssem Eddine

Encadré par :

- Melle : Brighen Assia

Promotion : 2016

Remerciement

Le plus grand merci revient tout d'abord à Dieu qui nous a guidés dans le bon sens durant notre vie et qui nous a donné la volonté, l'aide, la patience et le courage pour élaborer ce modeste travail.

Nous tenons à remercier tout particulièrement Mlle Brighen Assia d'avoir accepté d'être notre encadreur. Nous lui sommes extrêmement reconnaissantes pour l'aide, pour les encouragements et pour les conseils qu'elle nous a donnés.

Nous tenons à remercier vivement les membres de jury qui ont accepté de juger notre travail. Qu'ils trouvent ici, l'expression de notre profond, respect et notre sincère gratitude.

Nous ne terminons pas sans avoir exprimé nos vifs remerciements à tous les enseignants qui ont contribué à notre formation

Enfin nous remercions toutes les personnes qui nous ont donné un soutien moral lors de la réalisation de ce travail.

02
02

Dédicace

Je dédie ce modeste travail :

A mes chers parents

A tous mes frères et sœurs

A toute ma famille

A mon binôme et sa famille

A toutes mes amies et collègues

A tous ceux qui méritent mon respect et ma bienveillance

A tous ceux que j'aime et tous ceux qui m'aiment

A tous ceux qui me connaissent.

Riad

Dédicace

Je dédie ce modeste travail :

A mes chers parents

A tous mes frères

A toute ma famille

A mon binôme et sa famille

A toutes mes amies et collègues

A tous ceux qui méritent mon respect et ma bienveillance

A tous ceux que j'aime et tous ceux qui m'aiment.

A tous ceux qui me connaissent.

Houssem

Table des Matières

4. Mise en œuvre et évaluation de l'algorithme

Liste des Figures

Liste des Tableaux

Liste des abréviations

ACID	Atomique Cohérente Isolée et Durable
API	Application Programming Interface
BI	Business Intelligence
BSP	Bulk Synchronous Processing
DAG	Directed Acyclic Graph
EC2	Elastic Compute Cloud
EIGRP	Enhanced Interior Gateway Routing Protocol
EPFL	École Polytechnique Fédérale de Lausanne
HDFS	Hadoop Distributed File System
IaaS	Infrastructure as a Service
IBM	International Business Machines
IDE	Integrated Development Environment
IGRP	Interior Gateway Routing Protocol
JDK	Java Development Kit
JSON	JavaScript Object Notation
JVM	Java Virtuelle Machine
MCP	Maximum Clique Problem
NoSQL	Not Only Structured Query Language
Paas	Platform as a Service
RDD	Resilient Distributed Dataset
REST	Representational State Transfer
RIP	Routing Information Protocol
S3	Simple Storage Service
SaaS	Software as a Service
SDS	Software Defined Storage
SGBDR	Système de Gestion de Bases de Données Relationnelles
SI	Système Information
SQL	Structured Query Language
SSH	Secure Shell

Introduction générale

Partout dans le monde, le Cloud Computing devient une réalité pour de nombreuses entreprises. Deux grandes questions préoccupent actuellement les entreprises en matière d'informatique : le Cloud Computing et l'analyse Big Data. Le Cloud Computing, en tant que modèle de distribution des services informatiques, peut potentiellement améliorer l'agilité métier et la productivité de l'entreprise, renforcer son efficacité et réduire ses coûts. L'analyse Big Data peut potentiellement faire émerger de précieux éléments de compréhension, qui permettront de mieux comprendre les besoins des clients et d'affiner l'offre.

Ces deux technologies ne cessent d'évoluer. La valeur réelle du Big Data réside dans les informations résultant de leur analyse. Aujourd'hui, le stockage des Big Data n'est plus un problème en soi pour les entreprises, qui cherchent plutôt à déterminer comment elles peuvent conduire des analyses qui apporteront des réponses pertinentes et concrètes à leurs besoins.

Durant ces dernières années, l'explosion quantitative des données, conséquence de la flambée de l'usage d'Internet, au travers des réseaux sociaux, des appareils mobiles…etc., ont tous conduit à une génération importante de masses de Big Data. Face à cette croissance, les entreprises sont confrontées à certaines problématiques qui sont celles de savoir comment collecter, analyser et exploiter ces grands volumes de données pour créer de la valeur ajoutée. En effet, les systèmes classiques ne sont plus capables de répondre à ces problématiques.

D'un point de vue recherche, les verrous scientifiques liés au Big Data sont nombreux et essentiellement liés à l'exploitation et l'analyse de ces données. Il s'agit de découvrir de nouveaux ordres de grandeur concernant la capture, la recherche, le partage, le stockage, l'analyse et la présentation des Big Data. Les solutions à apporter doivent être efficaces en termes de temps et d'espace et doivent surtout garantir le passage à l'échelle.

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 naturellement 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.

Objectifs du travail

Le traitement de Big Graph a 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. De plus, un algorithme de traitement typique suit les arcs du graphe, la raison pour laquelle une implémentation séquentielle naïve n'est plus efficace, et le temps de calcul peut vite exploser exponentiellement sur ces grands graphes.

Le recours se trouve dans le parallélisme et la distribution des algorithmes au sein d'un cluster de serveurs ou d'un Cloud. Il est donc nécessaire de redéfinir l'algorithme pour qu'il fonctionne sur un environnement distribué. C'est-à-dire, décomposer le graphe en plusieurs partitions et les manipuler en parallèle sur plusieurs machines pour accélérer le traitement et réduire alors la limitation des ressources fixée par une seule machine.

Pour mettre en œuvre cette solution, de nombreux frameworks spécifiques au développement d'algorithmes de graphes distribués existent, tel que GraphLab et Giraph. Le framework MapReduce/Hadoop été le premier framework considéré pour être utilisé sur des applications de traitement des graphes en raison de la popularité du modèle de programmation MapReduce. Les récentes additions à l'écosystème sont le projet GraphX, qui s'exécute au-dessus de framework Spark et qui a été dévoilé en 2013 par la fondation Apache. Certes, il est encore jeune, mais il dispose d'une communauté énorme. C'est aussi l'un des projets ayant une vitesse de développement rapide et qui s'avère être le successeur du fameux MapReduce.

L'objectif de notre travail est de démontrer l'adaptation et l'élégance d'implémentation d'un algorithme parallélisés pour traiter l'un des fameux problèmes de graphes, c'est bien celui de « la clique maximale », en utilisant le nouveau framework de calcul distribué, Spark GraphX.

Organisation du mémoire

Afin d'atteindre les objectifs sollicités auparavant, nous avons organisé ce mémoire en deux parties :

Partie 1 : Etat de l'art

Cette partie constitue l'aspect théorique du mémoire. Elle inclut 2 chapitres :

- **Chapitre 1 :** Il est consacré à des notions fondamentales et généralités à propos de Cloud Computing et de Big Data. On abordera aussi quelques concepts de base sur les graphes dans le contexte de Big Data (i.e. Big Graph), avant de présenter un bref aperçu sur les outils existants de traitement parallèle des graphes sur le Cloud.
- **Chapitre 2 :** Nous allons étudier GraphX, une librairie du framework Spark qui étend les fonctionnalités de Spark avec des algorithmes parallélisés pour traiter les problèmes de graphes. Nous détaillons son architecture, son modèle de donnée et de traitement, ainsi qu'une brève description de quelques algorithmes de graphes implémentés dans GraphX.

Partie 2 : Implémentation et mise-en-œuvre

Cette partie représente l'aspect pratique du mémoire. Deux chapitres sont inclut :

- **Chapitre 3 :** Il est destiné à la description détaillée de problème d'énumération des cliques maximales et la clique maximum dans un graphe donné. Nous allons parler de quelques algorithmes utilisés dans la théorie de graphe pour la résolution de ces deux problèmes.
- **Chapitre 4 :** Ce 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 et durée d'exécution.

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

Chapitre 1 :

Notions introductifs et concepts de base

1.1 Introduction

Depuis quelques années, le Cloud computing est devenu la nouvelle technique dans le monde, en grande partie, au marketing et aux offres de services proposés par de grands groupes comme Google, IBM et Amazon…etc. Cloud computing est la prochaine étape dans l'évolution d'Internet. Le Cloud computing donne la possibilité de faire des calculs et de disposer d'une capacité de stockage très performante [1].

D'un autre côté, l'évolution du système d'information (SI) amène les entreprises à traiter de plus en plus de données issues de sources toujours plus variées. Les prévisions de taux de croissance des volumes de données traitées dépassent les limites des technologies traditionnelles. On parle de pétaoctet (milliard d'octets) voir de zettaoctet (trilliard d'octets). Dans ce cas la nouvelle solution utilisée pour résoudre ces problèmes est la technologie de BIG DATA. [2]

Dans ce chapitre nous allons présenter un état de l'art sur le Cloud computing, les Big data, les graphes et l'utilisation des graphes. Par la suite, nous allons aborder des concepts lies a cette nouvelle technique et quelque outilles de traitement parallèle permettant de traiter un gigantesque nombre d'opération dans un temps réduit. Ainsi, l'utilisation du Cloud computing dans les Big graphes, et les outils de traitement parallèle permettent de traiter des informations de manière simultanée.

1.2 Cloud computing

1.2.1 Historique

Comme tous les concepts relevant autant de l'économie que de la technologie, il est difficile de dire avec précision quand a été inventé le Cloud computing. Selon certains, il faut remonter à 1960, avec les travaux de l'Américain John McCarthy (1927-2011), un des pionniers de l'intelligence artificielle qui considérait d'emblée l'informatique comme un service. Selon une autre source, c'est l'avènement des réseaux dans les années 1970 qui a rendu possible l'exécution déportée des tâches informatiques.

D'autres enfin mentionnent le fait qu'Amazon, un site de commerce électronique de dimension mondiale, a trouvé dans le Cloud computing la solution élégante à la sous-utilisation de son parc de serveurs informatiques en dehors des périodes de fête (qui représentent en termes de commandes un pic temporel ponctuel d'utilisation). En louant ses serveurs à la demande et

en proposant à ses clients ses outils S3 (Simple Storage Service) et EC2 (elastic compute Cloud), qui offrent respectivement des services de stockage de données et de calcul, Amazon a pu rentabiliser ses propres investissements en matériel informatique. L'expression « Cloud computing » a, quant à elle, été citée pour la première fois en 1997 par un professeur en systèmes de l'information, Ramnath Chellappa, qui a défini les limites de l'informatique non en termes techniques mais en termes économiques. D'autres sociétés, comme Salesforces, Google, ou IBM ont commencé dès 1999, à développer une économie numérique fondée sur ces principes. Toutes ces entreprises de dimension mondiale participent de manière active à la création de centres (data-centers ou clusters) offrant une puissance de calcul et de stockage inégalée. Ces data-centers sont un des enjeux stratégiques majeurs de la décennie 2015-2025 [3].

1.2.2 Définition

Le Cloud Computing traduit en français par informatique dans les nuages, informatique dématérialisée ou encore informatique virtuelle.

Le Cloud computing est une expression imagée désignant un ensemble de technologies matérielles et logicielles qui offrent à un utilisateur ou à une entreprise le moyen d'accéder en libre-service, n'importe quand et n'importe où, à des fichiers personnels, des applications logicielles opérationnelles ou toute autre ressource numérique au travers d'une infrastructure réseau fiable et sécurisée.

Le Cloud Computing rend possible l'externalisation de la puissance de calcul et de stockage. Il permet de déporter des serveurs distants, des traitements informatiques traditionnellement exécutés sur la machine locale de l'utilisateur. Le cloud caractérisé par [3] :

- Un accès en libre-service à la demande aux capacités de calcul.
- Un accès ubiquitaire au réseau.
- Une mise en commun des ressources.
- Une « Elasticité rapide ».
- Un service mesuré en permanence.

1. 2.3 Principe de Cloud Computing

Le Cloud computing est une forme particulière de gérance de l'informatique fondée sur le modèle client-serveur. Dans ce modèle, le serveur distant est un ordinateur performant, fiable, sécurisé, avec un système d'exploitation et un ensemble d'applications logicielles

toujours à jour. Ce serveur est sous le contrôle direct du prestataire de Cloud. Ce dernier propose un ensemble de services que le client peut utiliser à distance, en général via Internet ou des réseaux privés, après une étape nécessaire d'authentification.

L'amélioration de la bande passante (débit des données numériques) des réseaux, filaires et non filaires, a en effet permis de passer d'un modèle où les postes de travail des utilisateurs sont relativement autonomes et qui doivent, pour être efficaces, être équipés localement de ressources de calcul et de stockage importantes, à un modèle de 'client fin' caractérisé par une infrastructure matérielle et logicielle beaucoup plus légère et complètement découplée des données de l'utilisateur, qui restent stockées sur le serveur. L'emplacement physique des données et des applications n'est en pratique pas connu de l'utilisateur, ce dernier n'ayant pas à se soucier de l'installation des logiciels et de la configuration proprement dite de son poste de travail. En pratique, le Cloud revient à simplifier la vie de l'utilisateur (ou de l'administrateur des machines) en lui évitant l'installation locale répétée des logiciels professionnels ou de la configuration de chaque poste individuel [3].

1. 2.4. Modèles de déploiement de Cloud Computing

Le Cloud Computing est décomposé en trois types :

- **Public**, il est loué à un acteur, hors du pare-feu et plutôt réservé à des parties non Cœur de métier, moins transactionnelles du SI (Google, Amazon, Salesforce, etc.).
- **Prive**, il est dans ce cas réalisé en interne. Le Cloud Computing devient une pratique Architecturale.
- **Hybride**, il est une combinaison de Clouds privés et publics. Le schéma idéal est une Architecture de Cloud privé qui permet de recevoir des Clouds publics, mais cela peut également être des Clouds 'dédiés' à une entreprise chez un hébergeur.

Dans un tel modèle de Cloud, le Cloud privé joue le rôle de passerelle et de mécanisme de contrôle pour les services du Cloud public [4].

1. 2. 5. Les niveaux de service offerts par le Cloud computing

Il y a trois principaux niveaux [4] :

- IaaS (Infrastructure as a Service) où on loue de la capacité informatique, que ce soit du calcul ou du stockage. Par exemple on trouve Amazon EC2, GoGrid, Sun GRID.

- PaaS (Platform as a Service) où on loue une plateforme de développement, de test, de recette ou d'exécution. Par exemple on trouve Force.com, Google App Engine, Azure Services Platform.
- SaaS (Software as a Service) où l'on loue une application. Par exemple on trouve CRM Salesforrce.com, Google Docs, Adobe Photoshop Express Online.

1.2.6. Concept lies aux Cloud Computing

1. 2.6.1. Cluster

Une grappe (cluster) est un réseau d'ordinateurs, indépendants et interconnectés, gérés tous ensemble comme une seule ressource informatique. Cette structure permet d'additio nner les ressources de toutes les machines ainsi connectées dans un seul système [5].
L'avantage de cluster est :

- d'augmenter la disponibilité ;
- de faciliter la montée en charge ;
- de permettre une répartition de la charge ;
- de faciliter la gestion des ressources (processeur, mémoire vive, disques dur, bande passante réseau).

Le Cloud Computing permet de construire un nuage de Clusters et permet la connexion entre un ensemble de machines sur un réseau (internet ou un réseau locale). Par conséquent, les utilisateurs peuvent déployer des machines virtuelles dans ce nuage, ce qui permet d'utiliser un certain nombre de ressources (par exemple de l'espace disque, de la mémoire vive, ou encore du CPU) [6].

1.2.6. 2. Virtualisation

La virtualisation est une technologie logicielle éprouvée offrant la possibilité d'exécuter simultanément plusieurs systèmes d'exploitation et applications sur une même machine. Celle-ci bouleverse rapidement le paysage informatique en modifiant fondamentalement nos usages de la technologie. La virtualisation peut s'appliquer à des ordinateurs, à des systèmes d'exploitation, à des périphériques de stockage, à des applicatio ns ou à des réseaux [7].

1.2.6. 2.1. Les avantage de la virtualisation

La virtualisation possède plusieurs avantages, à savoir [7] :

- Réduire les dépenses d'investissement et les coûts d'exploitation.
- Assurer une haute disponibilité des applications.
- Minimiser ou éliminer les interruptions de service.
- Renforcer la productivité, l'efficacité, la flexibilité et la réactivité du département informatique.
- Accélérer et simplifier le-provisionnement des applications et des ressources.
- Optimiser la continuité et la reprise d'activité.
- Déployer une gestion centralisée.
- Concevez un véritable 'Software Defined Data Center'.

1.2.6.2.2. Les différents types de virtualisation

Il y a quatre types de virtualisation [8] :

- **Virtualisation des serveurs**

La plupart des serveurs fonctionnent à moins de 15 % de la capacité, ce qui conduit à la prolifération des serveurs et de la complexité. La virtualisation des serveurs répond à ces problèmes en permettant à plusieurs systèmes d'exploitation de fonctionner sur un seul serveur physique que les machines virtuelles.

- **Virtualisation du réseau**

La virtualisation de réseau est la reproduction intégrale d'un réseau physique. Le fonctionnement des applications dans les réseaux virtuels est le même qu'avec les réseaux physiques et offrant les mêmes caractéristiques du réseau physique en plus des avantages opérationnels et l'indépendance matérielle de la virtualisation.

- **Virtualisation de stockage**

Les volumes de données et les applications en temps ont besoin d'un nouveau niveau de stockage. La virtualisation de stockage est l'abstraction des disques et des lecteurs flash à l'intérieur de vos serveurs ; elle les combine dans des pools de stockage de haute performance, et les délivre en tant que logiciel. Le Stockage définie par logiciel SDS (software defined storage) est une nouvelle approche de stockage plus efficace. Les avantages de ces

système de stockage est la possibilité d'une meilleure connexion du matériel existant, une gestion jointe des systèmes de stockage connectés, des coûts réduits et un meilleur rendement [9].

- **Virtualisation des postes de travail**

Consiste à afficher sur un écran, des dizaines, des centaines voire des milliers de postes physiques, une image virtuelle du poste utilisateur qui est en fait réellement exécutée sur un serveur distant. Cela peut réduire les coûts et améliorer le service rapidement et facilement fournir des postes de travail et des applications virtualisés aux succursales [10].

1.2.6.3. Les acteurs du Cloud Computing

Amazon, Citrix, Google, HP, IBM, Intel, Microsoft ou SalesForce figurent parmi les principales entreprises du secteur. En France, les principaux acteurs sont représentés par Orange Business Services, et SFR Business Team ainsi que de plus petites entités parmi lesquelles des SSII, des fournisseurs de services en mode SaaS tels que Dassault Systèmes, Oodrive et des fournisseurs d'hébergement comme Gandi, Ozitem, Ikoula, OVH, PHPNET ou Sigma Services [15]. Dans ce qui suit nous allons parler de l'acteur IBM [11].

- **Le Cloud Computing dans IBM**

IBM a développé un modèle de Cloud computing pour optimiser la qualité de service qu'a délivré aux clients internes ou externes. IBM a aussi la capacité de proposer des solutions de type Cloud public ou privé, sans privilégier ou imposer un type d'approche, mais uniquement en tenant compte de la solution la plus adaptée aux besoins des entreprises.

- **IBM Smart Business Services**

C'est la gamme de services de Clouds sécurisés et prêts à l'emploi, pour des Clouds de type public ou privé :

- ✓ **IBM Smart Business on the IBM Cloud** est la gamme de solutions standardisées de Clouds publics, basée sur l'infrastructure IBM et accessible via un modèle de souscription.
- ✓ **IBM Smart Business Cloud** est la gamme de services de déploiement de Clouds privés dans les entreprises, Clouds administrés, ou non, par IBM.

- **IBM Smart Business Systems** est la gamme de solutions d'infrastructure pré intégrées permettant de démarrer rapidement un Cloud privé.

- **Avec IBM, toutes les combinaisons sont possibles** Les solutions et les services proposés permettent de déployer des Clouds de type IaaS, PaaS Ou SaaS, en mode public, privée, hybride [12].

1.2.6.4. Les avantages de Cloud Computing [13]

Les avantages de Cloud Computing sont :

- La possibilité de déployer et de rendre disponibles des applications majeures et des environnements de travail de manière immédiate
- Les données peuvent être partagées
- Possibilité de calculs particulièrement puissants
- Permettre un accès libre et ouvert au client
- Garantie d'un suivi constant du développement d'espace Cloud computing
- Possibilité de réduire le coût pour plusieurs utilisateurs utilisant le même service

1.2.6. 5. Les limite de Cloud Computing

L'inconvénient major de Cloud Computing se situe dans la confidentialité et la sécurisation des données des clients. En effet, le vol des données personnel d'un utilisateur est un risque permanent. Il y a d'autres limites de Cloud Computing telles que [14] :

- Le coût de Cloud Computing : Beaucoup d'entreprises ne regardent que les frais de stockage, mais il faut également prendre en compte les frais de transferts, qui peuvent s'avérer être importants, selon l'utilisation que l'entreprise faite du cloud.

- Le piratage : Certaines applications comme Facebook et Twitter sont très sujets aux attaques.

- L'optimisation des applications : Malgré une connexion internet rapide, avec un débit garanti, certaines applications web peuvent s'avérer être très lentes. Elles peuvent s'avérer être plus limitées que des applications fonctionnant sur les propres ordinateurs de l'entreprise.

- L'utilisation de la connexion pour le transfert des données : il peut avoir une connexion très performante.

1. 3. Big Data

1. 3.1. Définition

Les Big data, littéralement les 'grosses données', consiste en l'accumulation d'un nombre très important de données (bien plus grand que les capacités d'un ordinateur personnel), puis en l'analyse en temps réel de ces données. Ces analyses sont tellement gourmandes en puissance que l'on a souvent recours à plusieurs machines "parallélisées" pour permettre l'étude de toutes ces données dans un temps acceptable [15].

1. 3. 2. Caractéristiques des Big Data

Big Data n'est pas caractérisé uniquement par la taille des données, mais inclut également la variété de données et la vitesse de traitement de ces données. Ensemble, ces trois attributs forment les trois Vs de Big Data (Velocity-Volume-Variety) (voir la Figure 1.1) [15].

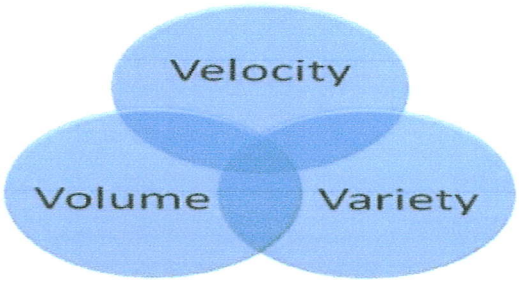


Figure 1.1 Les trois Vs de Bige Data [15].

- **Volume** Le volume de données traitées est considéré comme le premier critère pour qu'un ensemble de données relève du Big Data. Pourtant, ce premier **V** est le moins opérant et le plus variable en fonction du secteur et de l'organisation concernés. Certaines solutions de stockage et d'utilisation de ressources sont aujourd'hui disponibles via le Cloud Computing (voir la Figure 1.2) [16].

Dans la figure 1.2 en note la possibilité d'accès à des ressources de stockage et des infrastructures, permettant de traiter le volume, et diminuer le temps de traitement. On remarque aussi la possibilité d'accéder aux Cloud à partir de n'importe quel objet connecté [17].

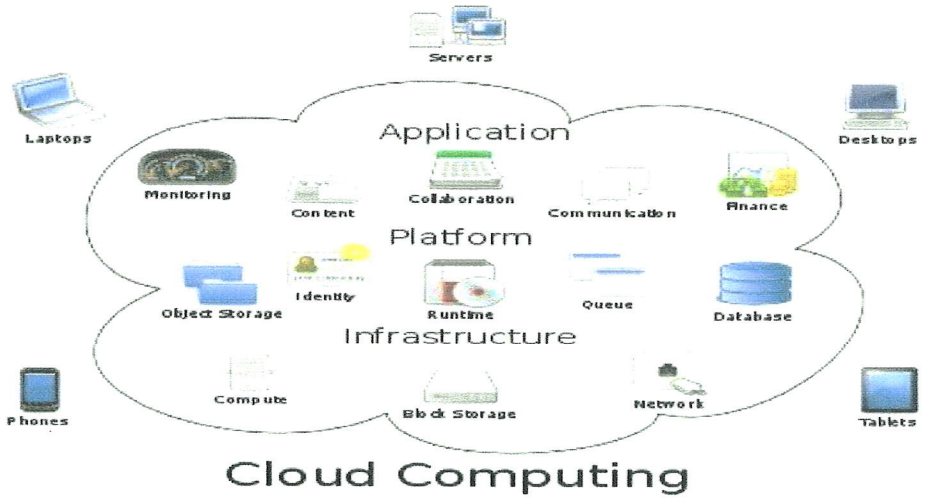


Figure 1.2 Ressources et infrastructures informatiques partagées en Cloud computing [17]

- **Vitesse (Vélocité)** Ce critère de vitesse renvoie à la faculté de traiter les jeux de données en un temps record, le plus souvent, en temps réel. Cela permet de créer des services directement fondés sur les interactions présentes [16].

- **La variété** Ces données arrivent de différentes sources et sont de nature variées : SMS, Tweets, réseaux sociaux, messageries, vidéo, …etc. Elles représentent une somme d'informations conséquentes et sont issues de différents domaines [17].

1.3.3. Analyse et découverte de Base de Donnée

La technologie de Big Data peut être décomposée en deux composantes majeures le composant matériel et le composant logiciel. Le composant matériel se réfère à la couche de composants et l'infrastructure. Le composant logiciel peut être divisé en organisation et gestion des données, l'analyse et découverte des données, et aide à la décision et de logiciels d'automatisation [15].

1.3.4. Les technologie de Big Data

- **MapReduce :** on trouve "Map Réduce", une technologie de traitement massivement parallèle issue des laboratoires 'Google Corp' avec gestion de la tolérance aux pannes et système de gestion de fichiers spécifiques (Google File System). On parle de traitement sur des milliers de machines réparties en clusters (plus de détaille voir la section 1.6.1).

- **Hadoop :** Hodoop c'est un 'Framework' mis au point par l'Apache Software Fondation afin de mieux généraliser l'usage du stockage et du traitement massivement parallèle de 'Map Reduce' et de 'Google File System'. Bien entendu, Hadoop possède ses limites. Quoi qu'il en soit, c'est une solution de Big data très largement utilisée pour effectuer des analyses sur de très grands nombres de données.

- **Bases NoSQL :** Les bases de données relationnelles ont une philosophie d'organisation des données bien spécifiques, avec notamment le langage d'interrogation SQL, le principe d'intégrité des transactions (ACID), et les lois de normalisation. Bien utiles pour gérer les données qualifiées de l'entreprise, elles ne sont pas du tout adaptées au stockage de très grande dimension et au traitement ultra rapide. Les bases NoSQL autorisent la redondance pour mieux servir les besoins en matière de flexibilité, de tolérance aux pannes et d'évolutivité.

- **Stockage "In-Memory" :** Pour des analyses encore plus rapides, les traitements directement en mémoire sont une solution, une technologie bien qu'encore trop coûteuse, il est vrai, pour être généralisée.

- **Cloud Computing :** Le Big Data exige une capacité matérielle hors du commun, que ce soit pour le stockage, comme pour les ressources processeurs nécessaires au traitement. Nul besoin de s'équiper outre mesure, le "Cloud" est là pour cela [19].

1.3.5. Les défis de Big Data

- Trouver un langage pour le Big Data : Toutes les sciences, chimie et mathématiques en tête, ont connu un formidable coup d'accélérateur en adoptant des systèmes de notation et un langage spécifique. Ne faut-il donc pas suivre la même voie dans le domaine du Big Data et inventer une notation algébrique et un langage informatique adaptés pour mieux partager et faciliter son analyse.

- Travailler sur des données fiables : Face à l'explosion du volume de données disponibles, l'enjeu est de savoir séparer le 'signal' du 'bruit' et l'information du hasard. La lutte contre le 'spam de données' et pour la qualité des données va devenir un enjeu crucial pour préserver la valeur du Big Data.

- Incorporer des données de plus en plus complexes Si le Big Data a d'abord concerné des données 'simples' (tableaux de chiffres, graphiques...), les données traitées sont

aujourd'hui plus complexes et plus variées (images, vidéos, représentations du monde physique et du monde vivant... etc.) il est donc nécessaire de repenser et de réinventer les outils et architectures du Big Data pour mieux capturer, stocker et analyser cette diversité de données.

- Mieux intégrer la variable temps La dimension temporelle est également un challenge important de développement pour le Big Data, tant pour analyser des causalités sur le long terme que pour traiter en temps réel des informations précises dans un large flux de données. Enfin, le problème se pose également en termes de stockage. Le volume de données créées va dépasser les capacités de stockage et nécessiter une sélection attentive.

- Ne pas délaisser ce qui ne peut encore se quantifier Les avancées récentes du Big Data, aussi impressionnantes soient-elles, ne permettent de quantifier, et d'intégrer dans les modèles d'analyse, qu'une très petite part du monde physique et social. Le risque est de réduire le réel à ces phénomènes quantifiés et représentés sous forme de données, et d'écarter des analyses tout ce qui n'a pas encore pu être évalué de la même façon.

- S'adapter aux capacités d'analyse du Big Data Le changement d'échelle offert par les technologies du Big Data a engendré des changements de paradigme profonds dans les champs scientifique, économique et politique. Mais il impacte également le champ de l'humain. Nos capacités cognitives se sont en effet développées pour traiter et se représenter un faible nombre de données. Le Big Data met donc à l'épreuve nos capacités d'analyse et notre perception du monde [20].

1. 4. Les graphes

1. 4.1. Définition de graph

Un graphe est un groupe d'objets associés représentés par un réseau de sommets et des arêtes, où un sommet est un objet et un arc reliant un sommet à l'autre sommet pour désigner leur relation par paires. Mathématiquement le graphe est un ensemble de sommets (ou points) et d'arcs (ou lignes orientées) ou d'arêtes liant certains couples de points. Formellement un graphe G est un couple (V, E) où [21]

- V est un ensemble (fini) d'objets. Les éléments de V sont appelés les sommets du graphe.

- E est sous-ensemble de VxV. Les éléments de E sont appelés les arêtes du graphe.

1.4.2 Utilités des graphes

Les graphes sont utilisés dans plusieurs domaines tels que [22] :

- L'organisation territoriale et Réseaux de transports routiers
- Réseaux de transports de données (téléphonie fixe, wifi …)
- Réseaux d'informations (bases de données, web, réseaux sociaux …)
- Réseau électrique qui transfère l'électricité à nos maisons, et les trajectoires de vol entre les aéroports.
- Systèmes biologiques présentent également des graphes, tels que les interactions entre les protéines et la topologie de conformation de polymères. Les neurones dans notre cerveau envoient des signaux sur les synapses, formant l'un des plus grands réseaux naturels dans l'existence.
- Nous concevons également des réseaux de circuits électroniques minutes dans les microprocesseurs au réseau numérique massive de l'Internet, ce qui facilite la communication entre les ordinateurs du monde entier.
- Les réseaux sociaux.

• Un exemple de Google

L'utilisation des graphes dans Google est importante, car à l'époque des premiers moteurs de recherche, les réponses aux recherches pointaient sur les pages les plus visitées, et non sur les pages les plus intéressantes. Le problème majeur était l'utilisation par certaines personnes de programme qui augmentaient le nombre de visites sur leurs pages afin d'être en haut dans les moteurs de recherches. Google a utilisé un nouveau système de recherche, au lieu de prendre les pages les plus visitées, il crée un arbre reliant les pages entres elles par des arcs représentant la redirection d'une page sur une autre. La page qui ressort est celle qui est pointée par le plus de pages différentes, elles-mêmes pointées par d'autres pages [22].

1.4.3. Définition de Big graphe

Les Big graphe sont des graphes contiennent un grand nombre des sommets et un grand nombre de quantité d'information. Par exemple, dans les dernières années, le web contiens des miliaires de pages web et chaque page contient plusieurs URL, le graphe de web aura un miliaire de miliaires de sommets, chaque sommet contiens un nombre définie d'informations.

Par exemple : En 2008, Google a sauvegardé dans le répertoire un très grand nombre de pages (un billion de pages). En Octobre 2012, Facebook a annonça que leur site de média social avait atteint un milliard d'utilisateurs actifs par mois, et depuis 2004, il y eut 140,3 milliards de connexions d'amis.

1.5. Les graphes sur le Cloud

Les graphes sont largement utilisés pour modéliser les Big Data, mais ils rencontrent des problèmes pour gérer des quantités des grosses données telles que le stockage et le traitement de ces données. Donc, il nécessaire d'utiliser une technique qui permet de résoudre ces problèmes. Le cloud computing est une technologie qu'on peut mettre à profit pour résoudre les problèmes de graphes et l'utiliser pour le développement des algorithmes parallèles de traitement de Big graphes.

1. 6. Outils de traitement parallèle des graphes sur le Cloud

Avec la croissance exponentielle et la complexité des réseaux provenant de diverses applications telles que les réseaux sociaux, World Wide Web, les réseaux de transport ...etc. les graphes sont largement utilisés pour modéliser ce type de réseaux.

Toutefois, la taille des Big graphes qui consistent en des millions (ou même des milliards) de sommets et des centaines de millions (ou milliards) d'arcs pose un grand défi pour les traiter. Cela signifie qu'il y a un besoin de développer de nouvelles techniques qui permettent de traiter les donnes de manière simple est efficace. L'idée informatique est de diviser le travail informatique en plusieurs tâches plus petites et de les distribuer sur plusieurs machines informatiques pour les traiter en parallèle. Il existe de nombreux paradigmes informatiques parallèles tels que le Framework Hadoop, Sparks, Apache Giraph, Graphx, Pregel, BSP...etc. Dans cette section nous allons présenter quelques paradigmes de traitement parallèle [23].

1.6.1. MapReduce

1.6.1.1. Définition

MapReduce est un modèle de programmation associé à une implémentation proposée initialement par Google. Il permet le traitement et la génération de données volumineuses.

Les données sont définies par l'utilisateur et dépendent de l'application. La sortie est un ensemble de couples (clé, valeur). L'utilisateur décrit son algorithme en utilisant les deux

fonctions Map et Reduce. La fonction 'Map' prend un ensemble de données et produit une liste intermédiaire de couples (clé, valeur). La fonction 'Reduce' est appliquée à toutes les données intermédiaires et fusionne les résultats ayant la même clé. La figure 1.3 présente un pseudocode d'un exemple de ces fonctions qui permet de compter le nombre d'occurrence de chaque mot dans un ensemble de documents données [24].

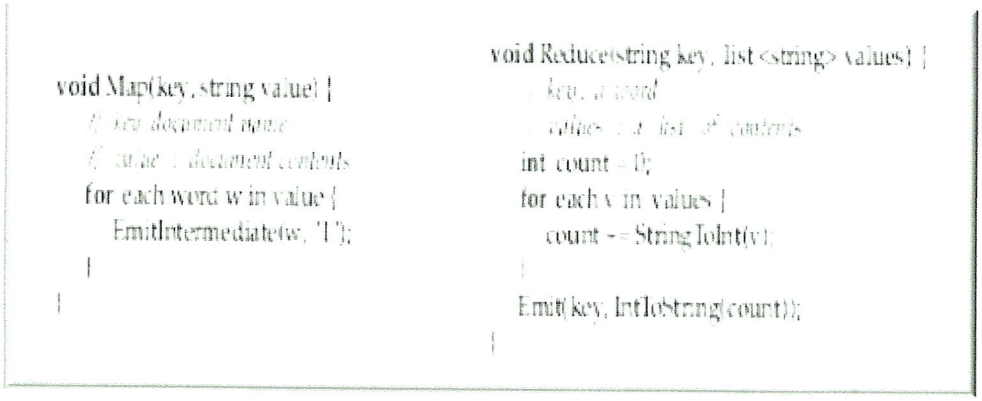


Figure 1.3 Exemple d'algorithme MapReduce

1.6.1.2. Caractéristiques de MapReaduce [25]

- La répartition de la charge sur un grand nombre de serveurs ;
- Distribution de haut niveau avec une abstraction quasi-totale de la couche matérielle (scalable friendly).
- MapReduce gère entièrement le cluster et la répartition de la charge. Cela permet de faire du calcul distribué dans un environnement Cloud.
- Plusieurs implémentations de ce Framework dans différents langages (C++, Java, Python, etc.) et utiliser par de nombreux organismes (Google, Yahoo, etc.).

1.6.1. 3. Principe de fonctionnement de MapReduce

MapReduce joue un rôle majeur dans le traitement de grandes quantités de données. La distribution des données au sein de nombreux serveurs permet le traitement parallélisé de plusieurs tâches portant chacune sur des morceaux de fichiers. La fonction 'Map' accomplit une opération spécifique sur chaque élément. L'opération 'Reduce' combine les éléments selon un algorithme particulier, et fournit le résultat. le principe de délégation peut être récursif : les

nœuds à qui sont confiées des tâches peuvent aussi déléguer des opérations à d'autres nœuds (voir la Figure 1.4) [26].

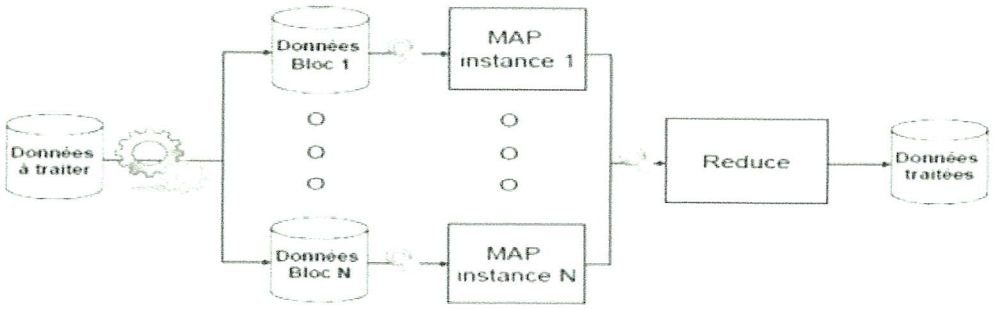


Figure 1.4 Principe de fonctionnement de MapReduce [26]

1.6.1.4. Exemples d'applications [27]

- Calcul de la taille de plusieurs milliers de documents.
- Trouver le nombre d'occurrence d'un pattern dans un très grand volume de données.
- Classifier de très grands volumes de données provenant des paniers d'achats de clients.

1.6.1.5. Avantages

- Fourni une abstraction totale des mécanismes de parallélisassions sous-jacents ;
- Peu de tests sont nécessaires. Les librairies MapReduce ont déjà été testées et fonctionnent correctement ;
- L'utilisateur se concentre sur son propre code ;
- Largement utilisé dans les environnements de Cloud Computing [27].

1.6.1.6. Inconvénients

- Une seule entrée pour les données ;
- Deux primitives de haut-niveau seulement ;
- Le flux de données en deux étapes le rend très rigide ;
- Le système de fichiers distribués (HDFS) possède une bande passante limitée en entrée/sortie ;
- Les opérations de tris limitent les performances du Framework (implémentation Hadoop).
- Utilisation interne chez Google [27].

1.6.2. Hadoop

Hadoop est une implémentation 'open source', en Java, de MapReduce distribuée par la fondation Apache. Il a été mis en avant par des grands acteurs du web tels que Yahoo! et Facebook. Les deux caractéristiques principales de Hadoop sont le Framework MapReduce et le HDFS (Hadoop Distributed File System) qui s'inspire du Google File System. Il permet de stocker et de traiter de grands ensembles de données de façon distribuée sur des clusters standards. Il permet aussi aux entreprises d'exploiter rapidement les informations contenues dans de grandes quantités de données, que celles-ci soient structurées ou non [28].

1.6.3. Spark

Spark originellement a été développé par AMPLab, de l'Université UC Berkeley, en 2009, et passé 'open source' sous forme de projet Apache en 2010. Apache Spark [29] est un Framework de traitements de grosses données 'open source' construit pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation. Il permet de partager les données en mémoire entre les graphes, de façon à ce que plusieurs tâches puissent être travaillées sur le même jeu de données. Spark s'exécute sur des infrastructures HDFS et propose des fonctionnalités supplémentaires. À côté des API principales, Spark contient des librairies additionnelles qui permettent de travailler dans le domaine des analyses Big data. Parmi ces librairies, on trouve : Graphx, ...etc. ce dernier est la nouvelle API (en version alpha) pour les traitements de graphes et de parallélisassions de graphes.

1.6.4. Apache Giraph

Giraph est un Framework conçu pour exécuter des algorithmes de graphes itératifs qui peuvent être facilement parallélisés sur des centaines de machines. Giraph est tolérant aux pannes et facile à programmer [30]. Facebook utilise Giraph avec quelques améliorations de performance pour analyser un billion d'arcs à l'aide de 200 machines en 4 minutes. Giraph est basé sur un document publié par Google autour de son propre système de traitement de graphe appelé Pregel [31].

1.6.5. Pregel et BSP

Pregel est un système de traitement de graphes à grande échelle. Il permet aux développeurs d'écrire des algorithmes de vertex-centric pour le traitement de graphes. Cela signifie qu'on aura juste besoin d'écrire une fonction qui reçoit des messages à partir des

sommets et qui envoie des messages vers d'autres. BSP est un modèle de calcul pour la mise en œuvre des algorithmes parallèles. Son objectif est de répliquer, dans un calcul parallèle, l'universalité du modèle Von Neumann [32]. Dans ce modèle, le graphe est décomposé en plusieurs « partitions ». Chaque partition contient un grand nombre de nœuds, et dont le traitement s'exécute d'une manière séquentielle. Le modèle d'exécution utilisé est le BSP (Bulk Synchronous Processing). Dans ce modèle, une multitude de tâches s'exécute en parallèle sous forme de séquences appelées « supersteps ». Au sein d'une superstep, chaque instance de programme reçoit tous les messages de la superstep précédente et envoie des messages à ses voisins. Une barrière est imposée entre les supersteps pour faire en sorte que toutes les instances du programme de traitement terminent les messages de la superstep précédente avant de passer à la suivante. Le programme se termine si la condition d'arrêt soit vraie. Le cœur de la programmation BSP est l'ordinateur BSP qui se compose d'un ensemble de processeurs reliés par un réseau de communication. Chaque processeur peut avoir différents files de calculs, chacune qui comprend une série de superstep. Chaque superstep se compose de 3 composants de calculs simultanés : plusieurs calculs indépendants se produisant de manière asynchrone ayant lieu sur chaque processeur participant [33].

1.7. Conclusion

Dans ce chapitre nous avons représenté le concept du Cloud Computing. Nous avons abordé les model de déploiement, les caractéristique, le principe de Cloud Computing et quel que concept liés au Cloud Computing, et nous avons présenté un ensemble d'avantage et d'inconvénient de Cloud Computing.

Le Cloud peut présenter des intérêts évidents pour une entreprise. Grâce à un tel système, elle peut proposer un ensemble d'applications à ses clients sans avoir à se soucier de sa maintenance ou son administration, tout en ayant l'assurance d'un système efficace et plus économe. Certaines peuvent même utiliser le Cloud pour commercialiser un service à travers des d'applications web.

Le Cloud Computing est une solution efficace pour résoudre le problème de stockage de données et les Big Graphes de données. Ce dernier est utilisé largement pour Modéliser les différents types des réseaux tels que réseaux sociaux, World Wide Web, les réseaux de transport ...etc. [34].

Nous avons aussi expliqué le terme 'Big data', ses caractéristiques, ses techniques, ses technologies, ses défis, ses avantages et ses inconvénients qui y sont liés.

Le Big Data va devenir un élément important dans la prise de décision des dirigeants d'entreprises et le traitement de données en temps réel [17]. Les graphes présente une technique efficace pour les modélisations des Big data .les graphes engendrée par des Big data sont des grande taille et nécessite des technique de traitement parallèle sur le cloud, ou travers Spark, Hadoop… . Dans le prochain chapitre, nous allons : présenter un paradigme de traitement des graphes largement utilisé sur le cloud : Spark et Graphx.

Chapitre 2 :

Graphx et les algorithmes de graphes

2.1 Introduction

Allant des réseaux sociaux aux réseaux pair-à-pair en passant par le World Wide Web, l'ampleur croissante et l'importance des données de graphes ont entrainé le développement de nombreux nouveaux systèmes de graphe-parallèle (ex., Pregel, GraphLab).

En limitant le calcul qui peut être exprimé et introduisant de nouvelles techniques de partitionnement et distribution des graphes, ces systèmes peuvent efficacement exécuter des algorithmes de graphe itératives d'ordre de grandeur plus rapide que pas mal de systèmes de données-parallèles générales.

Cependant, les mêmes restrictions qui permettent ce gain de performance rendent également difficile d'exprimer beaucoup d'étapes importantes dans un pipeline typique d'analyse de graphes : la construction du graphe, la modification de sa structure et l'expression de calculs qui peuvent couvrir plusieurs graphes.

En conséquence, les pipelines de l'analyse de graphes existants composent les systèmes de graphe-parallèle et ceux de donnée-parallèle en utilisant des systèmes de stockage externes, conduisant ainsi à des mouvements importants de données et un modèle de programmation plus compliqué.

Pour relever ces défis, L'Apache Software Foundation (fondation Apache) a introduit Spark GraphX, une plateforme distribuée pour le traitement de graphes, qui unifie le traitement de graphe-parallèle et le traitement de donnée-parallèle. GraphX fournit un ensemble noyau d'opérateurs de graphe-parallèle suffisamment expressif pour mettre en œuvre les abstractions Pregel et PowerGraph. GraphX utilise aussi un ensemble de techniques d'optimisation des requêtes telles que la réécriture automatique de jointure afin de mettre en œuvre efficacement les opérateurs de graphe-parallèle [35].

Dans ce chapitre, nous commençons d'abord par une introduction vers Spark, un framework de traitement distribué de Big Data, sur lequel se base GraphX et d'autres composants essentiels dans le domaine de traitement et d'analyse de Big Data. Par la suite, nous allons détailler GraphX, l'outil de traitement parallèle de graphes, en exposant son modèle de données et de traitement, son architecture, son mécanisme de tolérance aux pannes, ainsi que ses avantages et limites. Et pour mieux comprendre ce nouveau paradigme, nous allons présenter quelques algorithmes de graphes les plus utilisées dans GraphX, illustrés par des codes source en langage Scala, ainsi que des simples exemples d'application.

2.2 Apache Spark

Dans cette section, nous proposons un aperçu de ce qu'est Spark, de la suite d'outils mis à disposition pour les traitements de Big Data et nous expliquons comment Spark se positionne par rapport aux solutions classiques de MapReduce [29].

2.2.1 Qu'est-ce que Spark ?

Spark (ou Apache Spark) est un framework open source de calcul distribué et de traitements de Big Data, construit pour effectuer des analyses sophistiquées et conçu pour la rapidité et la facilité d'utilisation. Celui-ci a originellement été développé en 2009 par AMPLab, de l'Université UC Berkeley, puis passé open source sous une licence BSD en 2010, et maintenant un projet Apache depuis 2013.

Spark présente plusieurs avantages par rapport aux autres technologies Big Data et MapReduce comme Hadoop et Storm. D'abord, Spark propose un framework complet et unifié pour répondre aux besoins de traitements de Big Data pour divers jeux de données : divers par leur nature (texte, graphe, etc.) aussi bien que par le type de source (batch ou flux temps-réel). Ensuite, Spark permet à des applications d'être exécutées sur des clusters Hadoop jusqu'à 100 fois plus vite en mémoire, 10 fois plus vite sur disque. Il permet d'écrire rapidement des applications en Java, Scala ou Python et inclut un jeu de plus de 80 opérateurs haut-niveau. De plus, il est possible de l'utiliser de façon interactive depuis un shell.

Enfin, en plus des opérations de Map et Reduce, Spark supporte les requêtes SQL et le streaming de données et propose des fonctionnalités de machine learning et de traitements orientés graphe. Les développeurs peuvent utiliser ces possibilités en mode stand-alone ou en les combinant en une chaîne de traitement complexe [29].

2.2.2 L'écosystème de Spark

À côté des API principales de Spark (Spark Core), l'écosystème contient des librairies additionnelles qui permettent de travailler dans le domaine des analyses Big Data, Big Graph et de la machine learning (voir la Figure 2.1). Parmi ces librairies, on trouve [29] :

- **Spark Streaming** : Spark Streaming peut être utilisé pour le traitement temps-réel des données en flux. Il s'appuie sur un mode de traitement en "micro batch" et utilise pour les données temps-réel DStream, c'est-à-dire une série de RDD (voir Section 2.4).

- **Spark SQL** : Spark SQL permet d'exposer les jeux de données Spark via une API JDBC et d'exécuter des requêtes de type SQL en utilisant les outils BI (Business Intelligence, en français Informatique Décisionnelle) et de visualisation traditionnels. Spark SQL permet d'extraire, transformer et charger des données sous différents formats (JSON, Parquet, base de données) et les exposer pour des requêtes ad-hoc.
- **Spark MLlib** : MLlib est une librarie de machine learning qui contient tous les algorithmes et utilitaires d'apprentissage classiques, comme la classification, la régression, le clustering, le filtrage collaboratif, la réduction de dimensions, en plus des primitives d'optimisation sous-jacentes.
- **Spark GraphX** : GraphX est la nouvelle API pour les traitements de graphes et de parallélisation de graphes. GraphX étend les RDDs de Spark en introduisant le « Resilient Distributed Dataset Graph », un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arêtes. Pour le support de ces traitements, GraphX expose un jeu d'opérateurs de base (comme subgraph, joinVertices, aggregateMessages), ainsi qu'une variante optimisée de l'API Pregel. De plus, GraphX inclut une collection toujours plus importante d'algorithmes et de builders pour simplifier les tâches d'analyse de graphes.

En plus de ces librairies, on peut citer BlinkDB et Tachyon : BlinkDB est un moteur de requêtes approximatif qui peut être utilisé pour exécuter des requêtes SQL interactives sur des volumes de données importants. Tachyon est un système de fichiers distribué qui permet de partager des fichiers de façon fiable à la vitesse d'accès en mémoire à travers des frameworks de clusters comme Spark et MapReduce.

Il existe aussi des adaptateurs pour l'intégration à d'autres paradigmes comme Cassandra (Spark Cassandra Connector) et R (SparkR). Avec le connecteur Cassandra, on peut utiliser Spark pour accéder aux données stockées dans Cassandra et réaliser des analyses sur ces données.

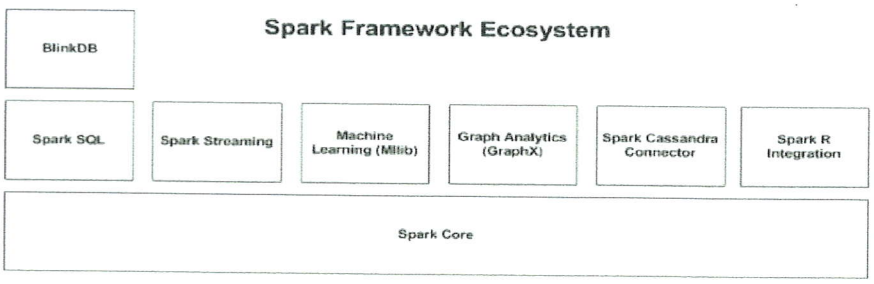


Figure 2.1 Les librairies du framework Spark [29]

2.2.3 L'architecture de Spark

L'architecture de Spark comprend les trois composants principaux suivants (voir la Figure 2.2) [29] :

- **Le stockage distribué de données** : Spark utilise le système de fichiers HDFS pour le stockage de données. Il peut fonctionner avec n'importe quelle source de données compatible avec Hadoop, dont HDFS, HBase, Cassandra, etc.
- **L'API** : L'API permet aux développeurs de créer des applications Spark en utilisant une API standard. L'API existe en Scala, Java, Python et R.
- **Gestion des ressources** : Spark peut être déployé comme un serveur autonome (standalone) ou sur un framework de traitement distribué comme Mesos ou YARN.

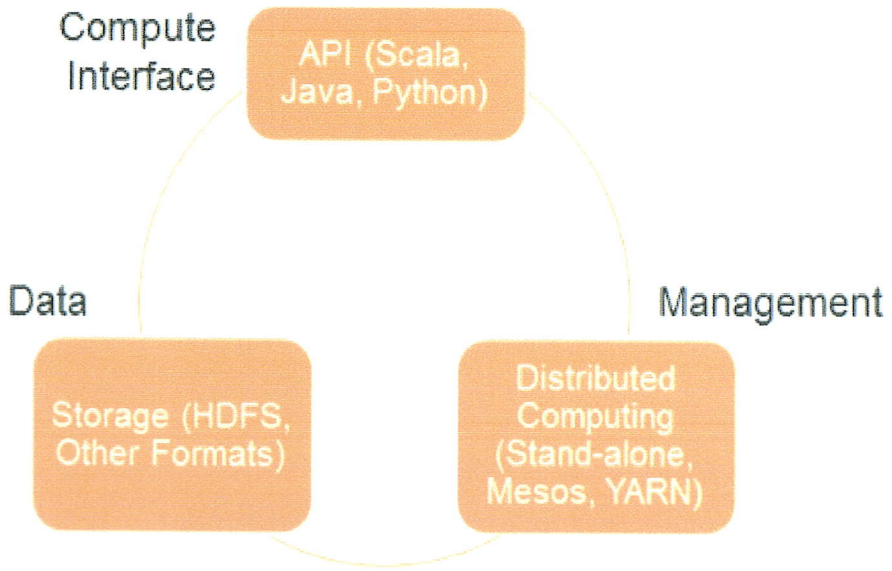


Figure 2.2 Les composants du modèle d'architecture de Spark [26]

2.2.4 Les RDDs "Resilient Distributed Datasets"

Les Resilient Distributed Datasets (ou RDD) sont un concept au cœur du framework Spark. Un RDD peut être vu, par analogie, comme une table dans une base de données. Celui-ci peut porter tout type de données et est stocké par Spark sur différentes partitions. Les RDDs permettent de réarranger les calculs et d'optimiser le traitement. Ils sont aussi tolérants aux pannes car un RDD sait comment recréer et recalculer son ensemble de données en cas de reprise. Les RDDs sont immutables : pour obtenir une modification d'un RDD, il faut y

appliquer une transformation, qui retournera un nouveau RDD, l'original restera inchangé. Les RDDs supportent deux types d'opérations [29] :

- **Les transformations** : les transformations ne retournent pas de valeur seule, elles retournent un nouveau RDD. Rien n'est évaluée lorsque l'on fait appel à une fonction de transformation, cette fonction prend juste un RDD et retourne un nouveau RDD. Les fonctions de transformation sont par exemple : map, mapToPair, filter, flatMap, reduce, reduceByKey, union, intersection... etc.
  - map(func) permet de transformer un élément en un autre élément.
  - mapToPair() permet de transformer un élément en un tuple clé-valeur.
  - filter(func) permet de filtrer les éléments en ne conservant que ceux qui correspondent à une expression.
  - flatMap(func) permet de découper un élément en plusieurs autres éléments.
  - reduce() et reduceByKey() permet d'agréger des éléments entre eux.
  - union(otherDataset) and intersection(otherDataset) permet de retourner un nouveau dataset contenant l'union ou l'intersection des éléments du dataset source et dataset argument.
- **Les actions** : les actions évaluent et retournent une nouvelle valeur. Au moment où une fonction d'action est appelée sur un objet RDD, toutes les requêtes de traitement des données sont calculées et le résultat est retourné. Les actions sont par exemple : reduce, collect, count, first, take, saveAsTextFile et foreach.
  - reduce(func) pour agréger les éléments d'un jeu de données (un dataset) en utilisant une fonction func.
  - collect() pour récupérer les éléments d'un dataset sous une forme d'un tableau.
  - count() pour compter le nombre des éléments.
  - first() pour retourner le premier élément, similaire à take(1).
  - take(n) pour retourner un tableau avec les premier n éléments d'un dataset.
  - saveAsTextFile() pour sauver le résultat dans un fichier texte.
  - foreach(func) pour exécuter une fonction func sur chaque élément du dataset.

Enfin, l'API permet de conserver temporairement un résultat intermédiaire grâce aux méthodes cache() (stockage en mémoire) ou persist() (stockage en mémoire ou sur disque, en fonction d'un paramètre).

2.2.5 Les variables partagées

Spark fournit deux types de variables partagées pour permettre d'exécuter de façon efficace les programmes Spark sur un cluster [29] : Broadcast et Accumulators.

- **Les variables Broadcast** : ces variables permettent de maintenir des variables en cache, en lecture seule, sur chaque machine, plutôt que d'avoir à les envoyer avec les tâches. Elles peuvent être utilisées pour mettre à la disposition des nœuds du cluster des copies de jeux de données volumineux de façon plus efficace. L'extrait de code suivant montre comment utiliser les variables Broadcast :

```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))
scala> val v = broadcastVar.value
v: Array[Int] = Array(1, 2, 3)
```

- **Les accumulateurs** : les accumulateurs peuvent être ajoutés lors de l'utilisation d'opérations associatives ; leur support est donc efficace dans le cadre de traitements parallèles. Ils peuvent être utilisés pour implémenter des compteurs (comme avec MapReduce) ou des sommes. Les tâches exécutées sur le cluster peuvent ajouter un accumulateur avec la méthode add. Cependant, ils ne peuvent pas lire sa valeur. Seul le programme pilote peut lire la valeur d'un accumulateur. L'extrait de code suivant montre comment utiliser un accumulateur :

```
scala> val accum = sc.accumulator(0, "My Accumulator")
scala> sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)
scala> val v = accum.value
v: Int = 10
```

2.2.6 Les fonctionnalités de Spark

Spark apporte des améliorations à MapReduce grâce à des étapes de shuffle moins coûteuses. Avec le stockage en mémoire et un traitement proche du temps-réel, la performance peut être plusieurs fois plus rapide que d'autres technologies Big Data. Spark supporte également les évaluations paresseuses des requêtes (dites "lazy evaluation"), ce qui aide à l'optimisation des étapes de traitement. Il propose une API de haut-niveau pour une meilleure productivité et un modèle d'architecture cohérent pour les solutions Big Data.

Spark maintient les résultats intermédiaires en mémoire plutôt que sur disque, ce qui est très utile en particulier lorsqu'il est nécessaire de travailler à plusieurs reprises sur le même jeu de données. Le moteur d'exécution est conçu pour travailler aussi bien en mémoire que sur disque. Les opérateurs réalisent des opérations externes lorsque la donnée ne tient pas en mémoire, ce qui permet de traiter des jeux de données plus volumineux que la mémoire agrégée d'un cluster. Spark essaye de stocker le plus possible en mémoire avant de basculer sur disque. Il est capable de travailler avec une partie de données en mémoire, une autre sur disque.

Il est nécessaire d'examiner ses données et ses cas d'utilisation pour évaluer ses besoins en mémoire car, en fonction du travail fait en mémoire, Spark peut présenter d'importants gains de performance. Les autres fonctionnalités proposées par Spark comprennent [29] :

- Des fonctions autres que Map et Reduce,
- L'évaluation paresseuse des requêtes ce qui optimise le workflow global de traitement,
- Des APIs concises et cohérentes en Scala, Java, Python et R,
- Un shell interactif pour Scala et Python (non disponible encore en Java).

2.2.7 Hadoop et Spark

Les Big Data sont trop gros pour tenir sur une seule machine. Hadoop et Spark [29] sont des technologies qui distribuent les Big Data sur un cluster de nœuds. Hadoop est positionné en tant que technologie de traitement de données depuis 10 ans et a prouvé être la solution de choix pour le traitement de gros volumes de données. MapReduce est une très bonne solution pour les traitements à passe unique mais n'est pas la plus efficace pour les cas d'utilisation nécessitant des traitements et algorithmes à plusieurs passes. Chaque étape d'un workflow de traitement étant constituée d'une phase de Map et d'une phase de Reduce, il est nécessaire d'exprimer tous les cas d'utilisation sous forme de patterns MapReduce pour tirer profit de cette solution. Les données en sortie de l'exécution de chaque étape doivent être stockées sur un système de fichier distribué avant que l'étape suivante commence. Cette approche a tendance à être peu rapide à cause de la réplication et du stockage sur disque.

De plus, les solutions Hadoop s'appuient généralement sur des clusters, qui sont difficiles à mettre en place et à administrer. Elles nécessitent aussi l'intégration de plusieurs outils pour les différents cas d'utilisation Big Data (comme Mahout pour le Machine Learning et Storm pour le traitement par flux). Si on souhaite mettre en place quelque chose de plus

complexe, on devra enchaîner une série de jobs MapReduce et les exécuter séquentiellement, chacun de ces jobs présentant une latence élevée et aucun ne pouvant commencer avant que le précédent n'ait tout-à-fait terminé.

Spark permet de développer des pipelines de traitement de données complexes, à plusieurs étapes, en s'appuyant sur des graphes orientés acycliques (DAG). Contrairement à Hadoop qui utilise le patron d'architecture MapReduce sur des disques, Spark travaille en mémoire vive ce qui est potentiellement 100 fois plus rapide (voir la Figure 2.3). Spark permet aussi de partager les données en mémoire entre les graphes, de façon à ce que plusieurs jobs puissent travailler sur le même jeu de données. Il s'exécute sur des infrastructures HDFS (Hadoop Distributed File System) et propose des fonctionnalités supplémentaires. Il est possible de déployer des applications Spark sur un cluster Hadoop existant.

Plutôt que de voir Spark comme un remplaçant d'Hadoop, il est plus correct de le voir comme une alternative au MapReduce d'Hadoop. Spark n'a pas été prévu pour remplacer Hadoop mais pour mettre à disposition une solution complète et unifiée permettant de prendre en charge les différents cas d'utilisation et besoins dans le cadre des traitements Big Data.

Figure 2.3 Spark clustering et la distribution de données sur la RAM [38]

2.2.8 Les avantages de Spark

Spark présente plusieurs avantages par rapport à Hadoop MapReduce [36] :

- **Performance** : Selon Apache, Spark offre des performances jusqu'à 100 fois plus rapides en mémoire et jusqu'à 10 fois plus rapides sur disque.
- **Productivité** : Spark inclut un mode interactif et supporte au moins 80 opérateurs de haut-niveau et une variété de langages de programmation puissants (Java, Scala, Python, notamment).

- **Streaming** : MapReduce est un framework qui se limite essentiellement au traitement d'un batch de données stockées à la fois, Spark permet également de traiter en temps-réel des flux continus de données, grâce à son extension Spark Streaming.
- **Polyvalence** : En plus des opérations traditionnelles de types Map et Reduce, Spark supporte le traitement de graphes de données, le scripting SQL (via Spark SQL, ex-Shark), le Machine Learning (via la librairie incluse MLlib), là où Hadoop MapReduce doit passer par des technologies complémentaires (Mahout pour le Machine Learning, par exemple).
- **Ergonomie** : Spark est beaucoup moins complexe à administrer qu'un cluster Hadoop.

2.3 Spark et GraphX

2.3.1 Motivation

Allant des réseaux sociaux à la publicité ciblée en passant par la modélisation des protéines et de l'astrophysique, les big graphes capturent la structure des données et sont au cœur des récents progrès dans l'apprentissage machine et le fouille de données. Ainsi, l'application directe des outils existants de données-parallèles (ex., Hadoop et Spark) dans les tâches de traitement de graphes peut être inefficace. Le besoin de nouveaux outils évolutifs et intuitifs pour le traitement de graphes a conduit à la mise au point de nouveaux systèmes de graphe-parallèle (i.e., Pregel et GraphLab) qui sont conçus pour exécuter efficacement des algorithmes de graphes (voir la Figure 2.4). Malheureusement, ces systèmes n'abordent pas les défis de la construction du graphe et de la transformation et par contre fournissent une tolérance aux pannes et un support limité pour l'analyse interactive [37].

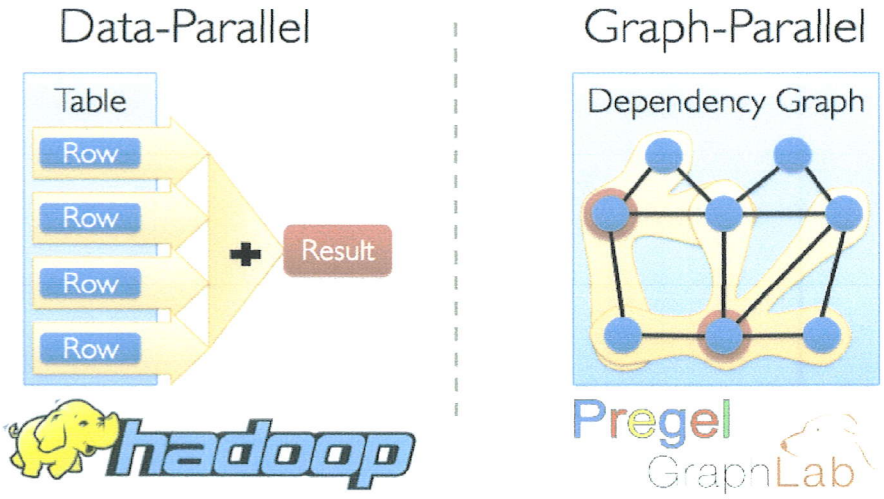


Figure 2.4 Données-parallèles vs. graphes-parallèles [37]

Le projet GraphX combine les avantages des deux systèmes de graphe-parallèle de donnée-parallèle (voir Figure 2.4) en exprimant efficacement le traitement de graphe au sein du framework Spark. Des nouvelles idées pour la représentation des graphes distribués ont été mises en place pour distribuer efficacement les graphes sous forme des structures de données tabulaires, et aussi pour exploiter le traitement en mémoire et augmenter la tolérance aux pannes. Des nouvelles opérations ont été introduites pour simplifier la construction des graphes et les taches de transformations. En utilisant ces primitives, l'équipe Apache responsable du projet GraphX a réussi de mettre en œuvre les abstractions PowerGraph et Pregel dans moins de 20 lignes de code. Enfin, en exploitant la fondation Scala de Spark, GraphX permet aux utilisateurs de créer facilement et de manière interactive, charger, transformer, raisonner et effectuer des calculs à grande échelle sur des données structurées de graphes [37].

2.3.2 Définition et architecture

GraphX [38] est une mince couche au-dessus de Spark (voir la Figure 2.5) qui fournit une structure de données de graphe composé de Spark RDD, et une API permettant de fonctionner sur ces structures. GraphX est livré avec la distribution standard de Spark, et peut être utilisé à l'aide d'une combinaison de l'API spécifique de GraphX et l'API régulière de Spark.

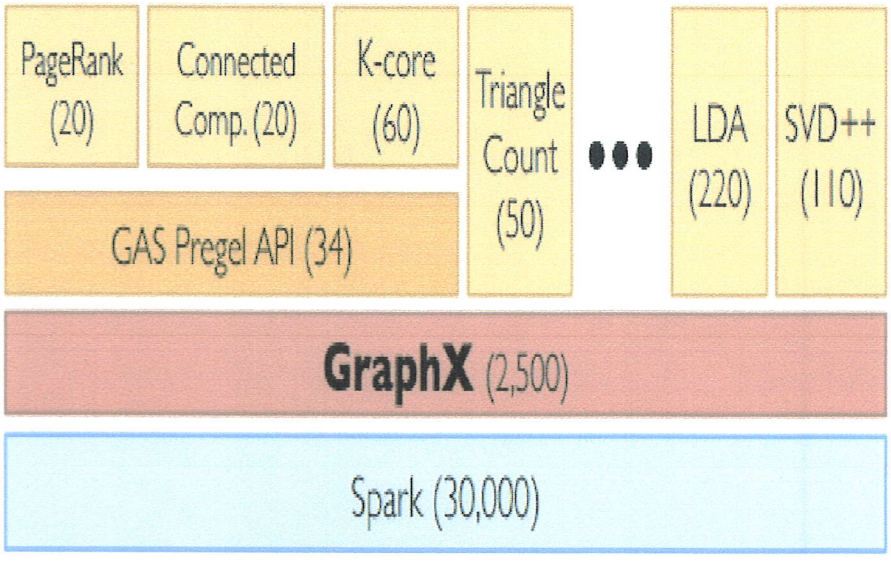


Figure 2.5 GraphX est une mince couche au-dessus de Spark (nombre de lignes de code) [44]

GraphX étend le concept de Spark RDD en introduisant le « Resilient Distributed Property Graph » : un multi-graphe orienté avec des propriétés attachées à chaque sommet et

arête. Pour prendre en charge le traitement sur les graphes, GraphX fournit un ensemble d'opérateurs de graphes fondamentaux, ainsi qu'une variante optimisée de l'API Pregel. En outre, GraphX contient une collection croissante d'algorithmes de graphes et des constructeurs pour simplifier les tâches d'analyse de graphe.

Comme GraphX fait entièrement partie de la distribution Spark d'Apache, les numéros de version pour le noyau Spark ainsi que ses composants de base, y compris GraphX, sont tous synchronisés.

2.3.3 Modèle de données : propriété de graphe

L'abstraction GraphX unifie le calcul de donnée-parallèle et de graphe-parallèle grâce à un modèle de données [35] qui représente les graphes et les collections comme des objets de première classe composables, ainsi qu'un ensemble d'opérateurs primitifs qui permet leurs compositions.

Le modèle de données GraphX se compose de collections immuables et de graphes dîtes « propriétés » (voir la Figure 2.6). La contrainte de l'immutabilité simplifie l'abstraction et permet la réutilisation de données et la tolérance aux pannes.

Les collections dans GraphX se composent de tuples non ordonnées (c.-à-d., paires de clé-valeur) et représentent des données non structurées. La clé peut être nulle et n'a pas besoin d'être unique, et la valeur peut être un objet arbitraire. Le vue de collection non ordonnée de données est essentielle pour le traitement des entrées brutes, l'évaluation des résultats de traitement de graphes, et certaines transformations de graphes. Par exemple, lors du chargement des données à partir d'un fichier, nous pourrions commencer par une collection de chaînes de caractères (avec des clés null) et ensuite appliquer les opérateurs relationnels pour obtenir un ensemble des « arêtes propriétés » (indexés par arête), construire un graphe et exécutez un algorithme de graphe, et enfin voir les valeurs résultants (indexés par identifiant de sommet) comme une collection pour des analyses supplémentaires.

Le graphe propriété G (P) = (V, E, P) combine des informations structurelles, V et E, avec des propriétés P = (P V, P E) décrivant les sommets et les arêtes. Les identificateurs de sommet $i \in V$ peuvent être arbitraires, mais le système GraphX utilise actuellement des entiers 64-bit. Ces identifiants peuvent provenir de l'extérieur (par exemple, les identifiants d'utilisateurs) ou en appliquant une fonction de hachage à un sommet propriété (par exemple, URL d'une page). Le graphe propriété regroupe les collections de sommets et arêtes propriétés

consistant de paires clé-valeur (i, P V (i)) et ((i,j), P E (i,j)) respectivement. Les sommets et les arêtes, comme déjà indiqué, peuvent chacun avoir un ensemble des attributs arbitraires. Chaque attribut peut être une valeur simple (i.e. l'âge d'une personne), ou un objet complexe (i.e. un document XML, image, vidéo…etc.)

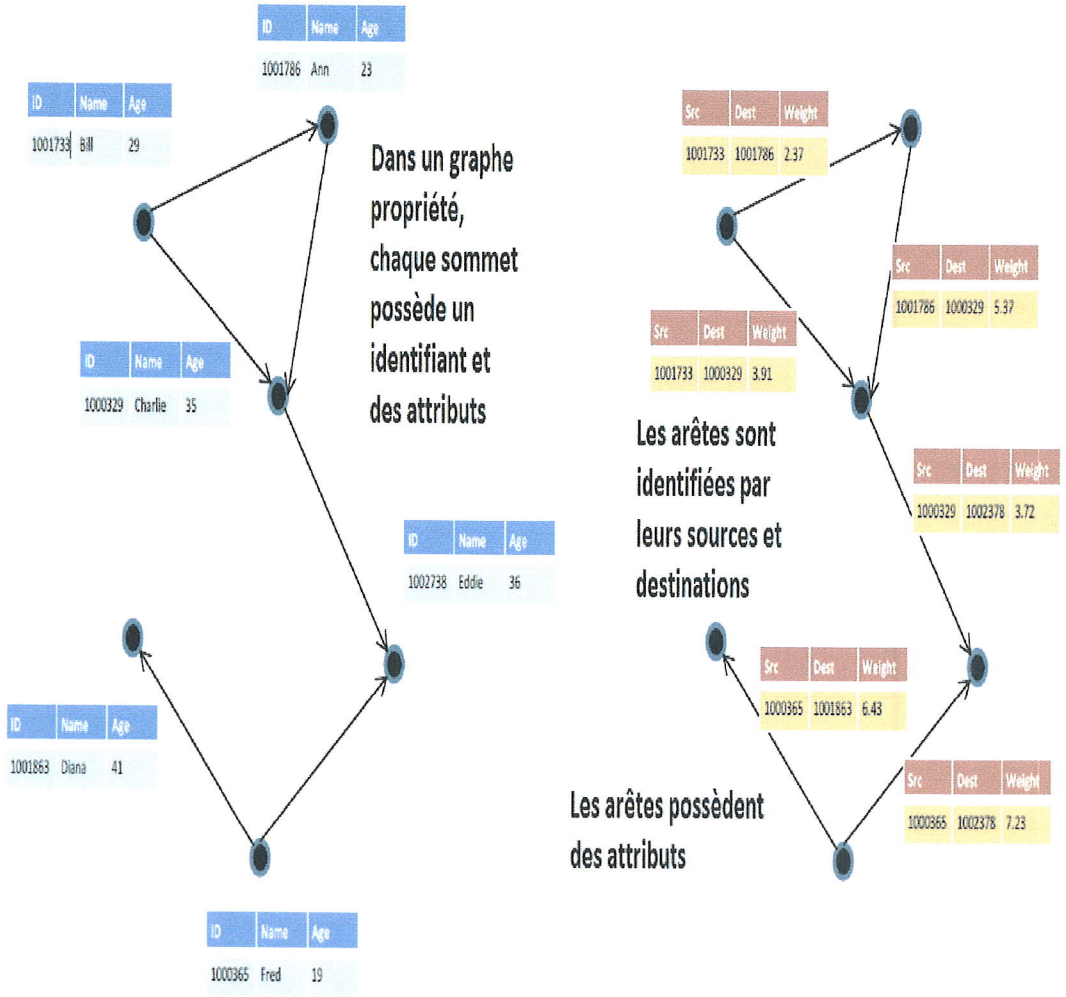


Figure 2.6 Exemple d'un graphe propriété [44]

GraphX stocke les sommets et les arêtes dans des tables séparées (voir la Figure 2.7). Cela permet à des algorithmes de graphes implémentés dans GraphX de traverser efficacement les graphes, soit sous forme de graphe (i.e. le travers d'un sommet à un autre) ou sous forme de tableaux de sommets et arêtes. Ce dernier mode d'accès permet des transformations efficaces sur les données de sommets et d'arêtes.

Bien que GraphX sauvegarde les arêtes et les sommets dans des tables séparées (ce qui est exactement le même principe à suivre si on voudrait concevoir un schéma SGBDR), dans son implémentation, GraphX possède des indexes spéciaux pour traverser rapidement le graphe,

il expose même une API qui rend facile l'interrogation et le traitement du graphe, plus facile que si on essaye de le faire avec SQL.

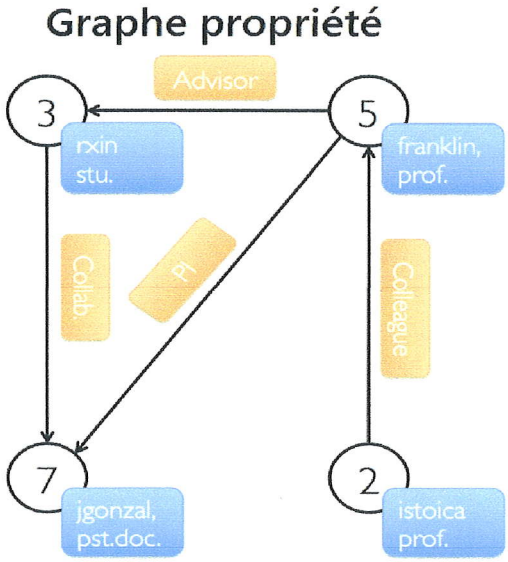


Table des sommets

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Table des arêtes

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Figure 2.7 GraphX sauvegarde les arêtes et les sommets dans des tables séparées [38]

2.4 Modèle de traitement

La solution de Spark GraphX est de conserver les résultats intermédiaires dans la mémoire vive de chaque nœud de cluster, et de garder l'historique des opérations ayant permis d'obtenir ces données. En cas de panne, l'historique permet de recalculer les données perdues (car stockées dans la mémoire vive d'un nœud de cluster tombé en panne) à partir des dernières données encore disponibles [39].

Spark GraphX permet de développer et d'exécuter des programmes en Scala, en Java, en Python et en R. Un programme Spark tourne dans une machine virtuelle Java (JVM). Le code Scala peut appeler de façon native des librairies écrites en Java.

La figure 2.8 présente le déploiement sur un cluster d'ordinateurs (ensemble d'ordinateurs interconnectés et gérés de façon unifiée) :

- **Le programme Driver** contrôle la logique de l'application et distribue les tâches aux nœuds de calcul (dites **Workers**). En raison des communications nécessaires entre le driver et

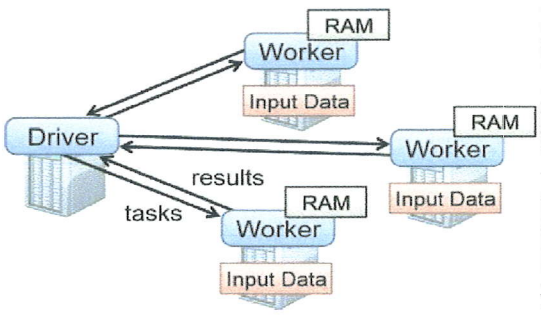


Figure 2.8 Déploiement d'un cluster Spark [42]

les workers, mieux vaut que le driver tourne sur un des ordinateurs du cluster.

- **Le gestionnaire de cluster (Cluster Manager)** veille à la bonne gestion des ressources du cluster. Spark propose son propre gestionnaire de cluster (mode autonome, aussi dite stand-alone, pas d'intégration avec d'autres opérations), mais permet aussi d'utiliser Apache Mesos (intégration possible avec d'autres opérations basées sur MapReduce) ou encore Hadoop YARN (intégration possible avec d'autres opérations basées sur MapReduce ou non-MapReduce).

Plusieurs applications Spark différentes (i.e. GraphX) exécutées sur les mêmes nœuds de cluster sont complètement isolées entre elles car chacune tourne dans une machine virtuelle Java (JVM) spécifique et est donc isolée des autres. Ces applications peuvent communiquer entre elles uniquement à travers des fichiers. Pour des raisons d'efficacité, le driver devrait tourner sur un nœud du cluster.

L'ordonnanceur de job de Spark GraphX utilise la représentation des RDDs déjà présentée. Lorsqu'un utilisateur exécute une action sur un RDD, l'ordonnanceur examine le graphe d'origine des RDDs pour construire un DAG (Directed Acyclic Graph) des étapes à exécuter. Chaque étape contient une série de transformations chainées avec autant de dépendances étroites que possible. L'ordonnanceur exécute alors les tâches pour calculer les partitions manquantes sur chacune des étapes jusqu'à l'obtention du RDD cible.

L'ordonnanceur assigne les tâches aux machines en fonction des données locales. Si une tâche nécessite une partition qui est disponible en mémoire sur un nœud, cette dernière est rapatriée. Sinon, si une tâche calcule une partition pour laquelle le RDD fourni une localisation souhaitée, alors la partition est envoyée à ce nœud. Si la tâche échoue, elle est ré-exécutée sur un autre nœud tant que l'étape parente est disponible. Si certaines étapes deviennent inaccessibles, la tâche est resoumise en parallèle afin de recalculer la partition manquante.

2.5 Stockage distribué des données

Parce que GraphX est strictement un système de traitement basé-mémoire, on a donc besoin d'un endroit pour stocker les données de graphes (voir la Figure 2.9). Spark espère un système de stockage distribué, tels que HDFS, Cassandra ou Amazon S3, ce qui est la façon habituelle pour le stockage de telles données. Cependant, certains préfèrent utiliser GraphX en conjonction avec une base de données orientée graphe, dite « base graphe » (i.e. Neo4j) afin de bénéficier des avantages de ces deux technologies. L'utilisation des bases de données graphes

séparées ajoutent la possibilité de réaliser des opérations de transactions sur les données de graphe [38].

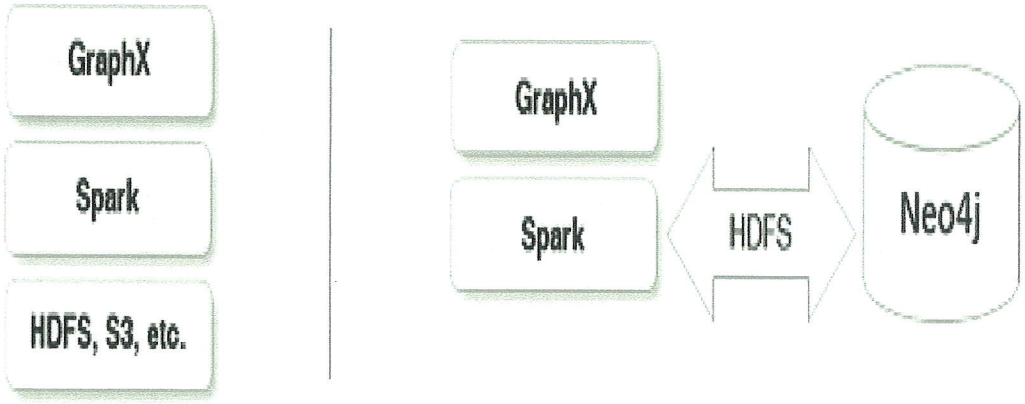


Figure 2.9 Stockage des données de graphes [45]

2.6 Tolérance aux pannes

Dans Spark (GraphX), les RDDs sont des collections de données partitionnées en lecture seule et créés à travers des opérations déterministes sur des données stables (ou sur d'autres RDDs). Ils comprennent des informations sur la traçabilité des données avec des instructions pour la transformation des données et pour leur persistance. Ils constituent une collection d'objets distribués qui peuvent être mis en mémoire cache sur plusieurs nœuds de cluster [40].

Les RDDs sont conçus pour l'ambition de prévenir la détérioration des opérations : si une opération échoue, elle est automatiquement reconstruite. La principale difficulté dans la conception des RDDs a été de définir une interface de programmation qui pouvait palier efficacement à la panne. Les abstractions de stockage en mémoire pour les clusters telles que les mémoires partagées distribuées, les stockages clé/valeur et les bases de données offrent une interface basée sur de petites mises à jours d'état mutables. Avec ces interfaces, les seules manières de fournir de la tolérance à la panne est de répliquer les données entre les machines ou de « logger » les mises à jours entre les machines. Ces deux approches sont consommatrices pour une charge de travail intensive sur des données puisque cela nécessite un fort transfert de données lors de la copie des informations entre les nœuds (le réseau est beaucoup plus lent que la RAM).

Si une partition d'un RDD est perdue, le RDD dispose de suffisamment d'informations sur la manière dont il a été produit pour recalculer la partition manquante. Ainsi, les données perdues peuvent être récupérées souvent rapidement sans avoir à recourir aux mécanismes de réplication souvent couteux [41].

2.7 Algorithmes de graphes

Spark GraphX contient un ensemble d'algorithmes open-source [42] qui simplifient et facilitent les tâches d'analyse de graphes. Ces algorithmes et ses codes sources sont contenus dans le package org.apache.spark.graphx.lib de la distribution source de Spark, et sont accessibles directement en tant que méthodes sur des objets Graph.

GraphX contient aussi un exemple d'un simple réseau social sur lequel on peut directement tester ces algorithmes. Le fichier "graphx/data/followers.txt", représente le graphe de ce réseau et contient les informations sur les relations (les arêtes) entre les utilisateurs de ce réseau (voir la Figure 2.10).

Dans cette section, nous avons sélectionné quelques algorithmes les plus populaires dont, pour chacun, nous expliquerons le principe et nous citons l'implémentation en langage Scala, ainsi qu'un simple exemple d'application utilisant comme entrée le graphe « followers ».

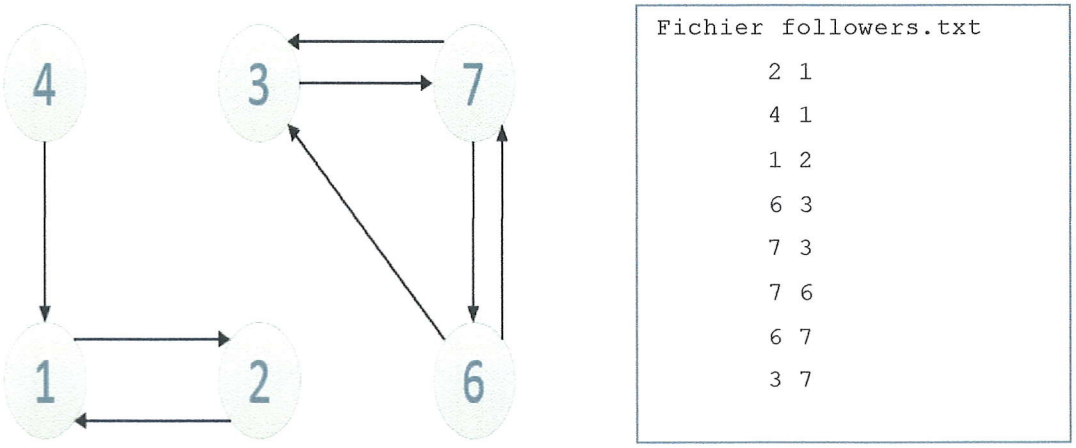


Figure 2.10 Exemple d'un simple réseau d'utilisateurs fournit avec GraphX

Code source 1. Implémentation de l'algorithme de PageRank statique [43].

```
object PageRank extends Logging {
  def runWithOptions[VD: ClassTag, ED: ClassTag](
      graph: Graph[VD, ED], numIter: Int, resetProb: Double = 0.15,
      srcId: Option[VertexId] = None): Graph[Double, Double] ={
    val personalized = srcId isDefined
    val src: VertexId = srcId.getOrElse(-1L)
    var rankGraph: Graph[Double, Double] = graph
      .outerJoinVertices(graph.outDegrees) { (vid, vdata, deg) => deg.getOrElse(0) }
      .mapTriplets( e => 1.0 / e.srcAttr, TripletFields.Src )
      .mapVertices { (id, attr) =>
        if (!(id != src && personalized)) resetProb else 0.0}
    def delta(u: VertexId, v: VertexId): Double = { if (u == v) 1.0 else 0.0 }
    var iteration = 0
    var prevRankGraph: Graph[Double, Double] = null
    while (iteration < numIter) {
      rankGraph.cache()
      val rankUpdates = rankGraph.aggregateMessages[Double](
        ctx => ctx.sendToDst(ctx.srcAttr * ctx.attr), _ + _, TripletFields.Src)
      prevRankGraph = rankGraph
      val rPrb = if (personalized) {
        (src: VertexId , id: VertexId) => resetProb * delta(src, id)
      } else (src: VertexId, id: VertexId) => resetProb  }
      rankGraph = rankGraph.joinVertices(rankUpdates) {
        (id, oldRank, msgSum) => rPrb(src, id) + (1.0 - resetProb) * msgSum
      }.cache()
      rankGraph.edges.foreachPartition(x => {})
      logInfo(s"PageRank finished iteration $iteration.")
      prevRankGraph.vertices.unpersist(false)
      prevRankGraph.edges.unpersist(false)
      iteration += 1} rankGraph}
```

- **PageRank dynamique** : cette implémentation (voir le Code source 2) utilise l'interface Pregel de GraphX et continue de s'exécuter jusqu'à la convergence (c.-à-d. quand toutes les améliorations de rang soient inférieures à une certaine tolérance) (PageRank.runUntilConvergence(graph, tol)).

Code source 2. Implémentation de l'algorithme de PageRank dynamique [43].

```
  def runUntilConvergenceWithOptions[VD: ClassTag, ED: ClassTag](
      graph: Graph[VD, ED], tol: Double, resetProb: Double = 0.15,
      srcId: Option[VertexId] = None): Graph[Double, Double] =
  {
    val personalized = srcId.isDefined
    val src: VertexId = srcId.getOrElse(-1L)
    val pagerankGraph: Graph[(Double, Double), Double] = graph
      .outerJoinVertices(graph.outDegrees) {
        (vid, vdata, deg) => deg.getOrElse(0)
      }
      .mapTriplets( e => 1.0 / e.srcAttr )
      .mapVertices { (id, attr) =>
        if (id == src) (resetProb, Double.NegativeInfinity) else (0.0, 0.0)
      }
      .cache()
    def vertexProgram(id: VertexId,attr: (Double,Double), msgSum: Double): (Double,
Double) = {
      val (oldPR, lastDelta) = attr
      val newPR = oldPR + (1.0 - resetProb) * msgSum
      (newPR, newPR - oldPR)
    }
    def personalizedVertexProgram(id: VertexId, attr: (Double, Double),
      msgSum: Double): (Double, Double) = {
      val (oldPR, lastDelta) = attr
      var teleport = oldPR
      val delta = if (src==id) 1.0 else 0.0
      teleport = oldPR*delta
      val newPR = teleport + (1.0 - resetProb) * msgSum
      val newDelta = if (lastDelta==Double.NegativeInfinity) newPR else newPR - oldPR
      (newPR, newDelta)
    }
    def sendMessage(edge: EdgeTriplet[(Double, Double), Double]) = {
      if (edge.srcAttr._2 > tol) {
        Iterator((edge.dstId, edge.srcAttr._2 * edge.attr))
      } else {
        Iterator.empty
      }
    }
    def messageCombiner(a: Double, b: Double): Double = a + b

    val initialMessage = if (personalized) 0.0 else resetProb / (1.0 - resetProb)

    val vp = if (personalized) {
      (id: VertexId, attr: (Double, Double), msgSum: Double) =>
        personalizedVertexProgram(id, attr, msgSum)
    } else {
      (id: VertexId, attr: (Double, Double), msgSum: Double) =>
        vertexProgram(id, attr, msgSum)
    }

    Pregel(pagerankGraph, initialMessage, activeDirection = EdgeDirection.Out)(
      vp, sendMessage, messageCombiner)
      .mapVertices((vid, attr) => attr._1) }}
```

-Dans l'exemple qui suit, nous sommes concentrés sur ce dernier algorithme car il est plus utile, parce qu'il est difficile de déterminer le nombre d'itérations que l'algorithme doit exécuter

- avant la convergence, et le nombre de calculs réduit d'une manière significative dans les dernières itérations lorsque la plupart des arêtes ne sont plus actifs.

- **Exemple d'application**

Nous calculons le PageRank de chaque utilisateur (de l'exemple précédant) comme suit :

```
// Construire un graphe à partir de fichier des arêtes
val graph = GraphLoader.edgeListFile(sc, "graphx/data/followers.txt")
// Exécuter PageRank (tol = 0.0001)
val ranks = graph.pageRank(0.0001).vertices
// Afficher les résultats
println(ranks.collect().mkString("\n"))
```

```
(Utilisateur, PageRank)
(4,      0.15)
(6,      0.7013599933629602)
(2,      1.390049198216498)
(1,      1.4588814096664682)
(3,      0.9993442038507723)
(7,      1.2973176314422592)
```

2.7.2 Enumération des triangles (Triangle Count)

- **Définition**

Dans un graphe, on dit qu'un sommet fait partie d'un triangle lorsqu'il a deux sommets adjacents avec une arête entre eux. GraphX implémente un algorithme d'énumération des triangles défini dans l'objet TriangleCount.scala qui détermine le nombre de triangles passant par chaque sommet, fournissant ainsi une mesure de clustering.

- **Domaine d'application**

L'énumération des triangles est très utile dans le domaine d'analyse des réseaux sociaux et la détection des communautés. Un triangle est un sous-graphe à trois nœuds, où chaque deux nœud sont connectés. Supposons, par exemple, qu'une personne possède deux amis qu'ils lui suivent sur Facebook, et ces deux amis sont suivis l'un par l'autre. Ces trois personnes ensemble composent « un triangle ».

De même, pour mesurer la connectivité d'un réseau social, généralement le plus le nombre de triangles qu'il possède, le plus ses connexions sont étroites.

- **Principe**

L'algorithme est relativement simple et peut être calculée en trois étapes :

- Calculer l'ensemble des voisins pour chaque sommet.
- Pour chaque arête, calculer le nombre des intersections entre ces ensembles puis envoyer le résultat aux deux sommets extrémités.

- A chaque somment, calculer la somme des nombres reçus et diviser le résultat par deux (car chaque triangle est compté deux fois).

- **Implémentation**

L'implémentation de cet algorithme (voir le Code source 3) exige que le graphe soit partitionné en utilisant Graph.partitionBy, et que les conditions suivantes soient satisfaites :

- L'absence des boucles (self-edges),
- L'absence des arêtes dupliquées (multi-edges),
- Toutes les arêtes sont en forme canonique (c.-à-d., srcID < dstID)

Code source 3. Implémentation de l'algorithme TriangleCount [43].

```
object TriangleCount {
  def run[VD: ClassTag, ED: ClassTag](graph: Graph[VD, ED]): Graph[Int, ED] = {
    val g = graph.groupEdges((a, b) => a).cache()
    val nbrSets: VertexRDD[VertexSet] =
      g.collectNeighborIds(EdgeDirection.Either).mapValues { (vid, nbrs) =>
        val set = new VertexSet(4)
        var i = 0
        while (i < nbrs.size) {
          if (nbrs(i) != vid) {
            set.add(nbrs(i))
          } i += 1
        } set }
    val setGraph: Graph[VertexSet, ED] = g.outerJoinVertices(nbrSets) {
      (vid, _, optSet) => optSet.getOrElse(null) }
    def edgeFunc(ctx: EdgeContext[VertexSet, ED, Int]) {
      assert(ctx.srcAttr != null)
      assert(ctx.dstAttr != null)
      val (smallSet, largeSet) = if (ctx.srcAttr.size < ctx.dstAttr.size) {
        (ctx.srcAttr, ctx.dstAttr)
      } else {
        (ctx.dstAttr, ctx.srcAttr)
      }
      val iter = smallSet.iterator
      var counter: Int = 0
      while (iter.hasNext) {
        val vid = iter.next()
        if (vid != ctx.srcId && vid != ctx.dstId && largeSet.contains(vid)) {
          counter += 1 }
      }
      ctx.sendToSrc(counter)
      ctx.sendToDst(counter)  }
    val counters: VertexRDD[Int] = setGraph.aggregateMessages(edgeFunc, _ + _)
    g.outerJoinVertices(counters) {
      (vid, _, optCounter: Option[Int]) =>
        val dblCount = optCounter.getOrElse(0)
        assert((dblCount & 1) == 0)
        dblCount / 2 }
}
```

- **Exemple d'application**

Nous calculons le nombre de triangles pour chaque utilisateur de réseau social de l'exemple précédant :

```
        // Construire le graphe en ordre canonique puis le partitionner
        val graph = GraphLoader.edgeListFile(sc, "graphx/data/followers.txt", true)
.partitionBy(PartitionStrategy.RandomVertexCut)
        // Calculer le nombre de triangles pour chaque utilisateur
        val triCounts = graph.triangleCount().vertices
        // Afficher les résultats
        println(triCounts.collect().mkString("\n"))
```

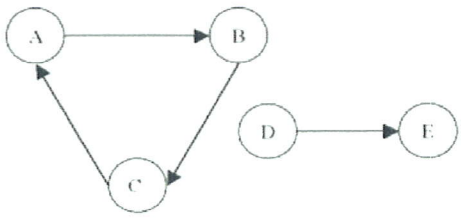
```
        (Utilisateur, Nombre de triangles)
        (4,      0)
        (6,      1)
        (2,      0)
        (1,      0)
        (3,      1)
        (7,      1)
```

2.7.3 Composantes connexes (Connected Components)

La connexité étant une propriété importante. Une des choses élémentaires à regarder dans un graphe pour commencer d'en comprendre la structure est de regarder « sa connexité ». Dans un graphe connexe, il est possible d'atteindre un sommet depuis n'importe quel autre en suivant un chemin (une séquence de sommets). Si le graphe n'est pas connexe, il est souvent divisé en composantes connexes, des sous-parties du graphe qui sont connexes, que l'on peut examiner individuellement.

- **Définition**

- Un graphe est dit connexe si tous ses nœuds ont deux à deux la relation de connexité.
- On appelle « composante connexe » un ensemble de nœuds qui ont deux à deux la relation de connexité. De plus, tout nœud en dehors de la composante n'a pas de relation de connexité avec aucun des éléments de la composante.



Exemple d'un graphe non connexe. {A,B,C} et {D,E} sont des composantes connexes.

- **Principe**

Dans un réseau, deux nœuds sont connectés s'il existe un chemin entre eux dans le graphe. Un réseau est dit « connecté » si tous ses pairs de nœuds sont connectés. Toutefois, un réseau « non connecté » contient plusieurs composantes dont chacune est connectée. Les

sommets appartenant à une même composante possèdent le même attribut dont la valeur est l'ID du sommet le plus petit dans la composante. En d'autre terme, l'attribut de chaque sommet identifie sa composante.

L'algorithme de composantes connexes marque chaque composante connexe du graphe avec l'ID de son sommet le plus petit. Par exemple, dans un réseau social, les composantes connexes peuvent représenter des clusters.

- **Implémentation**

GraphX contient une implémentation de cet algorithme (voir le Code source 4) dans l'objet ConnectedComponents.scala.

Code source 4. Implémentation de l'algorithme ConnectedComponents [43].

```
object ConnectedComponents {
  def run[VD: ClassTag, ED: ClassTag](graph: Graph[VD, ED]): Graph[VertexId,
ED] = {
    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
    Pregel(ccGraph, initialMessage, activeDirection = EdgeDirection.Either)(
      vprog = (id, attr, msg) => math.min(attr, msg),
      sendMsg = sendMessage,
      mergeMsg = (a, b) => math.min(a, b))
```

- **Exemple d'application**

Toujours avec le même premier exemple, on calcule ses composantes connexes comme suit :

```
// Construire le graphe
val graph = GraphLoader.edgeListFile(sc, "graphx/data/followers.txt")
// Trouver les composantes connexes
val cc = graph.connectedComponents().vertices
// Afficher les résultats
println(cc.collect().mkString("\n"))
```

```
(Utilisateur, ID de la composante (i.e. ID du sommet le plus petit) )
(4,      1)
(6,      3)
(2,      1)
(1,      1)
(3,      3)
(7,      3)
```

2.7.4 Plus court chemin (Shortest Path)

- **Principe**

L'algorithme calcule les plus courts chemins vers un sommet choisi au préalable (ou un ensemble de sommets) dit « landmark », et retourne un nouveau graphe où l'attribut de chaque sommet est un « map » contenant « la distance » du plus court chemin vers le sommet landmark (ou vers chaque sommet landmark atteignable).

- **Implémentation**

GraphX contient une implémentation de cet algorithme (voir le Code source 5) dans l'objet ShortestPaths.scala qui s'appuie sur une version distribuée de l'algorithme de Bellman-Ford, ce dernier qui est utilisé dans les protocoles de routage à vecteur de distances (i.e., RIP, IGRP et EIGRP).

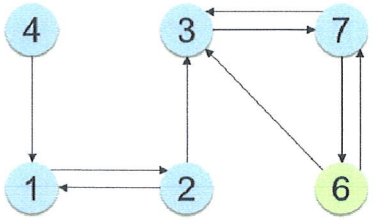
L'algorithme reçoit comme entrée le graphe d'entrée et l'ensemble de sommets landmarks (peut contenir un seul sommet ou plusieurs). L'appel à cet algorithme s'effectue à l'aide de la fonction ShortestPaths.run(graph, landmarks).

Code source 5. Implémentation de l'algorithme ShortestPath [44].

```
        object ConnectedComponents {
          def run[VD: ClassTag, ED: ClassTag](graph: Graph[VD, ED]): Graph[VertexId,
ED] = {
            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
            Pregel(ccGraph, initialMessage, activeDirection = EdgeDirection.Either)(
              vprog = (id, attr, msg) => math.min(attr, msg),
              sendMsg = sendMessage,
              mergeMsg = (a, b) => math.min(a, b))
```

- **Exemple d'application**

On considère le graphe suivant (à droite) pour lequel on calculera les plus courts chemins, et le sommet landmark n°6 vers lequel ces calculs seront effectués.



On fait appel à l'algorithme dans GraphX comme suit :

```
// Construire le graphe
val graph = GraphLoader.edgeListFile(sc, "graphx/data/followers.txt")
// Spécifier le sommet n°6 comme un sommet landmark
val landmark : Seq[VertexID] = Seq(6)
// Calculer les plus courts chemins vers le sommet n°6
val sp = ShortestPaths.run(graph, landmark).vertices
// Afficher les résultats
println(sp.collect().mkString("\n"))
```

```
(Sommet, Map(landmark -> distance))
(4,  Map(6 -> 5))
(6,  Map(6 -> 0))
(2,  Map(6 -> 3))
(1,  Map(6 -> 4))
(3,  Map(6 -> 2))
(7,  Map(6 -> 1))
```

2.8 Avantages et limites de GraphX

Les systèmes de traitement de graphes sont utiles pour, par exemple, répondre aux requêtes des services web, ou l'exécution de longs calculs autonomes. Spark GraphX facilite la composition des pipelines complexes utilisant à la fois les données-parallèles et le traitement de graphe-parallèle, ce qui accélère le développement des applications et des algorithmes d'analyse de graphes.

Supposant qu'on a un système qui utilise déjà Spark pour d'autres choses, et on a aussi besoin de traiter des données de graphe, alors Spark GraphX est une façon de le faire efficacement sans avoir à apprendre et à administrer une technologie de cluster complètement différente, tel qu'une base graphe séparée. Ou, et grâce à la rapidité de traitement de GraphX, on peut même coupler ce dernier avec une base graphe, tel que Neo4j, et réaliser le meilleur des deux mondes : les transactions sur la base graphe et le traitement rapide quand on en a besoin.

GraphX est encore jeune, et certaines de ses limites proviennent des limitations de Spark. Par exemple, les jeux de données GraphX, comme tous les jeux de données Spark, ne peuvent normalement pas être partagés par plusieurs programmes Spark, sauf si un serveur REST (comme Spark JobServer) est utilisé. Et jusqu'à ce que la fonctionnalité « IndexedRDD » soit ajoutée à Spark, qui est effectivement une version mutable d'un RDD (Resilient Dataset Distributed), GraphX reste limitée par l'immutabilité de Spark RDD, ce qui pose un problème pour les grands graphes.

Bien qu'il est plus rapide pour certaines utilisations, GraphX est souvent plus lent que les systèmes écrits en C ++, comme GraphLab/PowerGraph en raison de GraphX comptant sur la JVM (Java Virtual Machine) [38].

2.9 Conclusion

Dans ce chapitre, nous avons introduit GraphX, un framework distribué pour le traitement et l'analyse des graphes qui unifie le traitement de graphe-parallèle et de donnée-parallèle dans un seul système. GraphX est capable d'exprimer de façon succincte et d'exécuter efficacement le pipeline complet de l'analyse de graphe. L'abstraction GraphX unifie l'abstraction du graphe-parallèle et celle de donnée-parallèle dans un seul modèle de données en représentant les graphes et les collections comme des objets de première classe, avec un ensemble d'opérateurs primitifs permettant leur composition.

GraphX représente les graphes comme des ensembles d'arêtes et de sommets ainsi que des structures d'index auxiliaires, et modélise les traitements sur les graphes comme une séquence de jointures relationnelles et des agrégations. GraphX intègre aussi des techniques qui permettent d'exploiter les caractéristiques communes des charges de travail. Le résultat est un système qui permet d'atteindre des performances comparables aux celles des systèmes de graphe-parallèle contemporains, tout en conservant l'expressivité des systèmes de donnée-parallèle contemporains.

Nous s'intéressons dans le chapitre suivant à l'étude et l'adaptation d'un algorithme de graphe dans un environnement GraphX parallèle. Nous fournissons une description détaillée de cet algorithme, l'intérêt et les différents domaines d'applications, illustrés par des exemples.

Chapitre 3 :

Proposition d'un algorithme de graphe sous Graphx

3.1. Introduction

De nombreux réseaux peuvent être représentés par un graphe non orienté. On peut citer comme exemple les réseaux sociaux où la relation entre deux individus peut être représentée par une arête d'un graphe dont les sommets sont les individus. De même, en biologie l'interaction entre gènes, métabolites ou protéines peut être représentée par des arêtes d'un graphe [46]. L'analyse et l'exploration des graphes de données obtenus s'appuies sur les théories des graphes qui fournissent des algorithmes de graphes bien définissent. Parmi les algorithmes de graphes les plus utilisés on trouve l'énumération des cliques. Cette dernière est très utilisée dans de nombreux domaines applications, par exemple dans la biologique et la conception de codes correcteurs d'erreurs,... Etc.

Une clique maximale d'un graphe est un sous ensemble de sommets tous en relation deux à deux. La recherche des cliques d'un graphe représentant un réseau est d'un intérêt majeur puisqu'elle consiste à rechercher les sous-groupes d'individus tous en relation. Dans ce chapitre, dans un premier temps, nous allons présenter la description des cliques maximales (maximums), ses domaines d'applications et quelques algorithmes utilisés pour trouver toutes les cliques maximales dans un graphe donnée. Par la suite, nous allons adapter quelques célèbres algorithmes d'énumération des cliques maximales (maximums) pour être capable de s'exécuter sous GraphX.

3.2. Définition

Une clique dans un graphe non orienté est un sous graphe maximal complet. Formulée de façon plus explicite, une clique est un ensemble de sommets entre lesquels tous les liens possibles sont présents (complet), et il n'est pas possible d'ajouter un sommet sans que la propriété précédente ne disparaisse (d'où l'adjectif maximal). Un même sommet peut être membre de plusieurs cliques distinctes [47].

Étant donné un graphe non orienté $G = (V, E)$, une clique S est un sous ensemble de V tel que pour deux éléments quelconques $u, v \in S$ il existe un lien entre u et v. la figure 3.1 montre un exemple d'un graphe avec deux cliques maximales

3. 3. Application et utilité des cliques maximales

Permis les applications des cliques maximales on peut citer [48] :

- Modélisation des trafics de télécommunications.
- La conception de codes correcteurs d'erreurs.
- Le diagnostic de pannes sur les grands systèmes multiprocesseurs.
- Des problèmes de géométrie de carrelage (la conjecture de Keller).
- Vision par ordinateur et reconnaissance de formes.
- Le domaine biologique moderne.

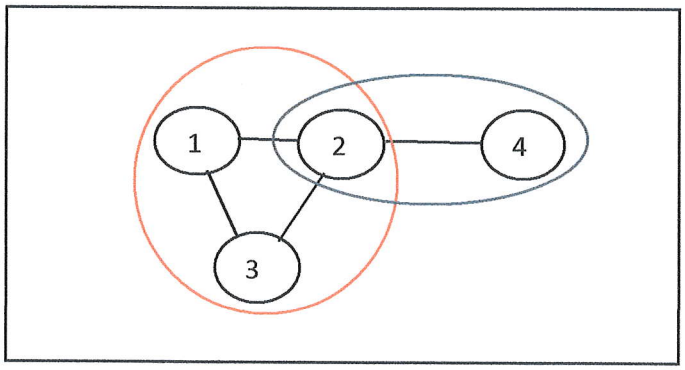


Figure 3.1 Exemple graphe avec 2 cliques maximales : (1, 2,3), (2,4)

3.4. Les algorithmes des cliques maximales

Parmi les algorithmes proposés pour l'énumération des cliques maximales dans un graphe donnée, on peut citer l'algorithme de Bron-Kerbosch :

3.4.1. Algorithme de Bron-Kerbosch

L'algorithme de Bron-Kerbosch a été décrit en 1973 par deux chercheurs hollandais, Joep Kerbosch et Coenraad Bron. Leur algorithme consiste à énumérer toutes les cliques maximales d'un graphe. Il y a trois variantes de cet algorithme : l'algorithme classique (sans pivot), avec pivot et Avec vertex commande. Dans cette partie nous avons choisi deux algorithmes avec pivot et sans pivot [49].

3.4.1.1. Algorithme sans pivot

L'algorithme Bron-Kerbosch sans pivot, (voir l'algorithme 1) s'appuie sur trois ensembles P, R, X. L'algorithme commence par R et X sont ∅ et P contient tous les sommets du graphe. R est utilisé pour sauvegarder le résultat temporaire, P contient l'ensemble des sommets des candidats possibles et X contient l'ensemble des sommets exclu des recherches fautives. N (v) indique les voisins d'un sommet v.

A chaque appel récursif un sommet v de P est choisir et ajouter v à R et retirer ses non voisins de P et X. Ensuite, choisir un autre sommet de la nouvelle série P et répétez le processus. Continuez jusqu'à ce que P est vide, une fois que P est X sont vides, R est signalé comme une nouvelle clique maximale. Maintenant revenir en arrière au dernier sommet choisi et restaurer P, R et X comme ils étaient choisir au début, retirez les sommets de P et ajouter à X. S'il n'y pas de sommet à l'intérieur de P on revenir en arrière au niveau supérieur [50].

```
Procédure BronKerboschsansPivot (R, P, X)
Début
    Si (P U X =∅) alors
        Signaler R comme une clique maximale
    Fin si
    Pour chaque sommet v ∈ P faire
    BronKerboschsansPivot (R ∪{v}, P∩N (v), X∩N (v))
        P ← P\ {v}
        X ← X ∪{v}
    Fin pour
Fin
```

Algorithme 1 : Algorithme sans pivot [6].

3.4.1.2. Algorithme Tomita et al (avec pivot)

Algorithme Bron-Kerbosch avec pivot (voir Algorithme 2) est une version modifiée par Tomita et al. Cet algorithme utilise une politique de pivotement spécifique pour éliminer des branches de calcul.

Un appel récursif à l'algorithme de Bron-Kerbosch fournit trois ensembles disjoints des sommets R, P et X, R est une clique (peut-être non maximale) et P ∪ X =N(R) sont des sommets adjacents à chaque sommet dans R.

Les sommets de P peuvent être ajoutés à la clique R, tandis qu'ensemble des sommets de X doivent être exclus de la clique, dans chaque appel récursif l'algorithme répertorie toutes les cliques dans $P \cup R$ qui sont maximales dans le graphe induit par $P \cup X \cup R$ [51].

On choisit un sommet appelé pivot $u \in (P \cup X)$, ce dernier est utilisé pour minimiser la taille de $P \backslash N(u)$ et $N(u)$ représenter les voisin de pivot u [52].Par conséquent, nous retardons l'ajout des sommets de $P \cap N(v)$ à la clique jusqu'à les prochains appels récursifs.

```
Procédure BronKerboschPivot (R, P, X)
Début
    Si (P U X =∅) faire
        Signaler R comme une clique maximale
    Fin si
    Choisir un pivot U∈P ∪ X
    Pour chaque sommet v ∈ P\N(u) do
        BronKerboschPivot (R ∪{v}, P∩N(v),  X∩N(v))
        P ← P\ {v}
        X ← X ∪{v}
    Fin pour
Fin
```

Algorithme 2 : Algorithme avec pivot [6]

3.4.1.2.1. Exemple :

Appliquant l'algorithme avec pivot sur le graphe G de la figure 3.2. Les étapes d'exécution sont les suivantes :

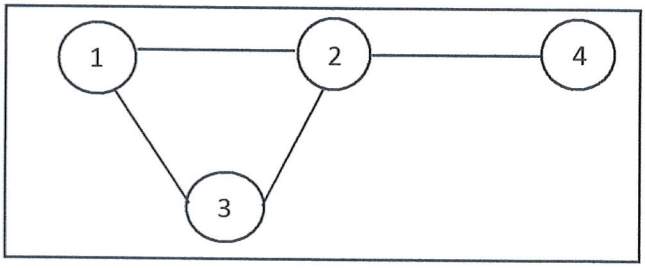
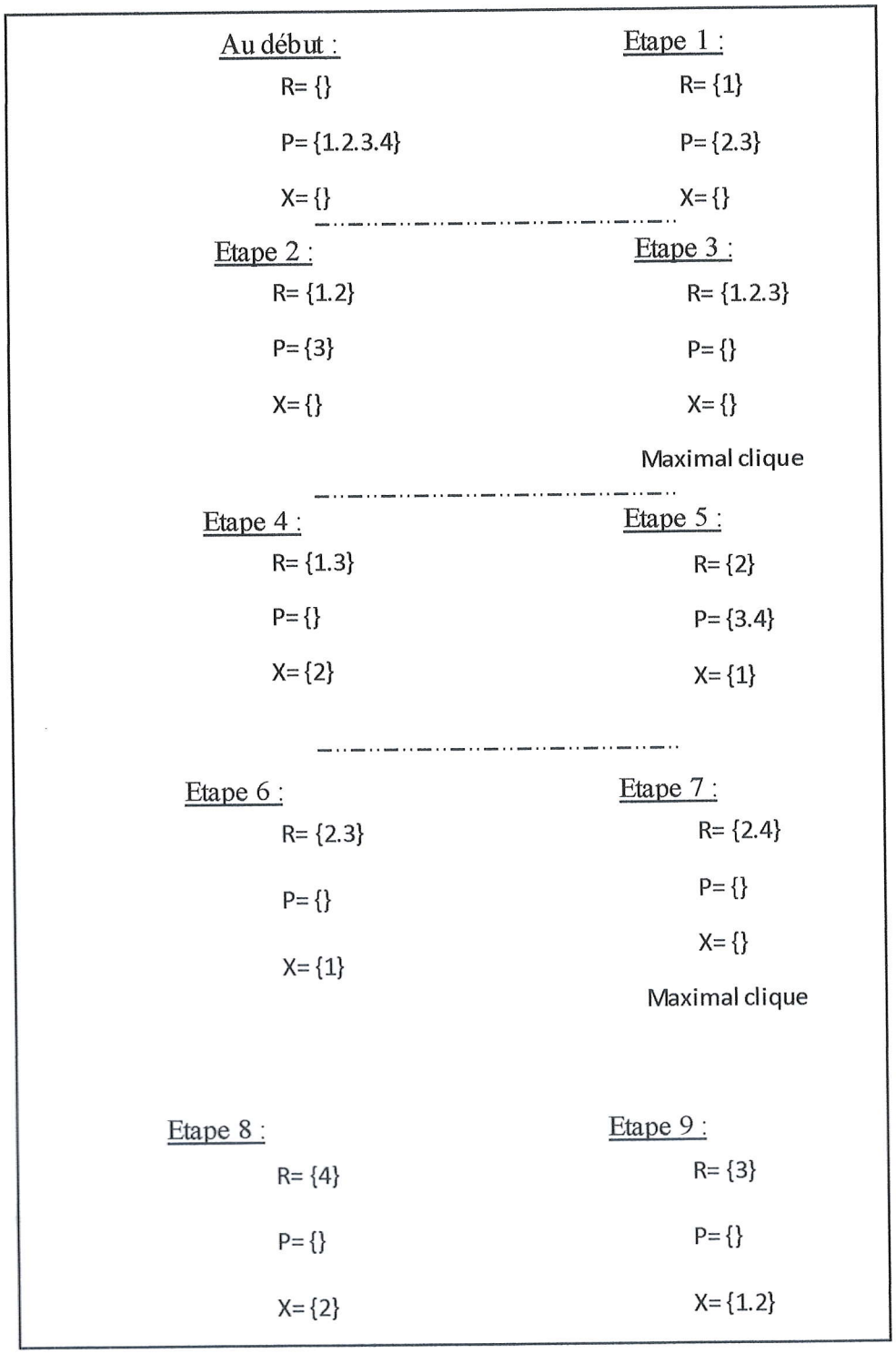


Figure 3.2 Un graphe G avec 2 cliques (1, 2,3), (2,4)



Dans cet exemple nous avons trouvé deux cliques maximales : {1.2.3} et {2.4}.

3.5. Proposition de l'algorithme d'énumération de cliques maximales sous GraphX

Dans cette section nous allons adapter les deux algorithmes de Bron-Kerbosch, avec pivot et sans pivot, pour être s'exécutent sous GraphX. Pour ce faire nous allons utiliser les

opérateurs de graph préfinis de Graphx. Par exemple subgraph, collectNeighborIds, reduce ... etc. Une description détaillée de ces opérateurs est présentée dans l'annexe A.2.

3.5.1. Algorithme de Bron-Kerbosch avec pivot

```
Tableau X, R;   Graph P;
BronKerboschavecpivot (Tableau: R, Graph P, Tableau X) {
  Choisir le pivot X∪P
  Voisin <- p. collectNeighborIds (pivot)
 If (P.vertices = ∅  AND x =∅ ) {
  Ecrire (« le clique maximal est :» R)}
  If (P.vertices ≠ ∅ ) {
    Pour (chaque sommet v dans p) {
      R.add (v)
      Voisin <- p. collectNeighborIds (v)
      P. subgraph (P∩ Voisin)
      x. intersection(Voisin)
      BronKerboschavecpivot (R, P, X)
     R. retirer (v)
     P<- P.subgraph (P/V)
      X.add(v)  }
}
```

Algorithme 3 : Algorithme de Bron-Kerbosch avec pivot sous Graphx

3.5.2. Algorithme de Bron-Kerbosch sans pivot :

```
Tableau X, R; Graph P;
BronKerboschavecpivot (Tableau: R, Graph P, Tableau X)
If (p. vertices = ∅  AND X =∅ )
 {
  Ecrire (« le clique maximal est :» R)
 }
If ((P. vertices ≠ ∅ )
{
  Pour (chaque sommet v dans p)  {
    R.add (v)
    Voisin <- p. collectNeighborIds (v)
    P. subgraph (P∩ Voisin)
    x.intersiction(Voisin)
    BronKerboschavecpivot (R, P, X)
    R. retirer (v)
    P<- P.subgraph (P/V)
    X.add(v)
  }
}
```

Algorithme 4 : Algorithme de Bron-Kerbosch sans pivot sous Graphx

3.6. Proposition des algorithmes d'énumération de clique maximum sous GraphX

La recherche de plus grande clique dans un graphe est un problème NP-dur appelé le problème de la clique maximum (MCP).Étant donné un graphe non orienté G = (V, E), une clique maximum S est un sous ensemble de V tel que pour deux éléments quelconques u, v ∈

S'il existe un lien entre u et v et représenter le clique qui contient le plus grande nombre d'arrêt ou de sommet. La figure 3.3 montre un exemple de clique maximum [53].

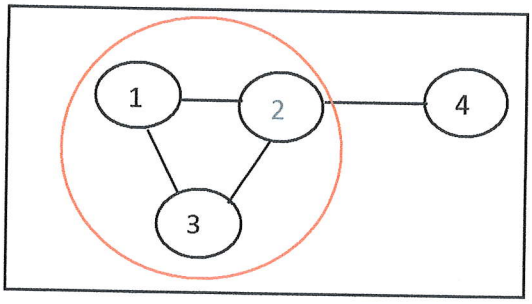


Figure 3.3 Exemple graphe avec une clique maximum : (1,2,3)

3.6.1 Algorithmes gloutons

La recherche d'une clique maximum dans un graphe G = (S, A) utilisant une heuristique gloutonne du « sommet de plus fort degré » (voir l'algorithme 6). On appelle heuristique un algorithme qui très puissant mais en générale ne détermine pas la valeur optimale [53].Sous GraphX nous allons implémenter cet algorithme en s'appuient sur les opératoires de graphes.

Debut

Tableau C, Cand ;

$C = \varnothing$

Cand = S

Tant que Cand $\neq \varnothing$ faire

Soit S_j le sommet de Cand ayant le plus fort degré dans

Le sous-graphe induit par Cand.

$C = C \cup \{S_j\}$

Cand = Cand $\cap$ $\{S_i / (S_i, S_j)\} \in A$

Fin Tant que

Fin

Algorithme 5 : Algorithme glouton

Voilà le pseudocode de cet algorithme sous Graphx :

```
Graph subgr ;
Tableau maxclique, cand, neighbours ;
Subgr <- graphe .subgraph
Cand <- graphe .vertices
Tan que (cand ≠ ∅)
  {
  Sj <- subgr.degree.reduce (max)
  maxclique.add(Sj)
  neighbours <- Subgr. collectNeighborIds(Sj)
  Subgraph <- Subgr. Subgraph (neighbours)
  Cand.add(Subgr)
}
Ecrire (« le clique maximum est :» maxclique)
```

Algorithme 6 : Algorithmes Glouton

3.6.2 Exemple

En appliquant l'algorithme glouton sur le graphe G de la figure 3.4, les étapes d'exécution sont les suivantes

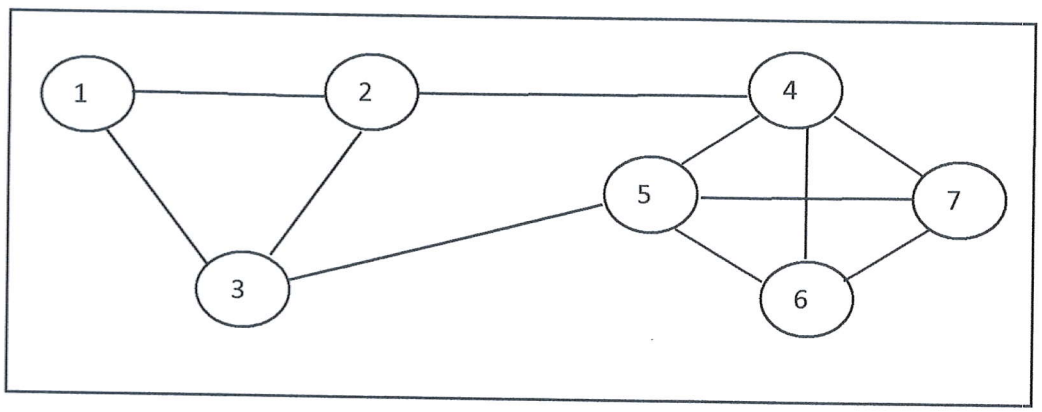
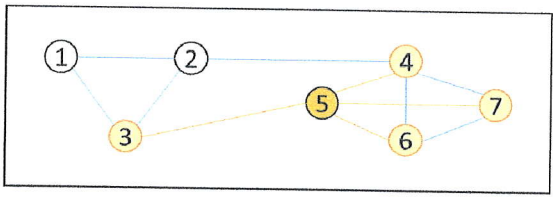


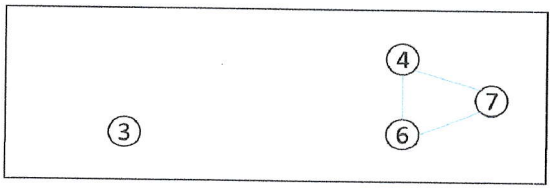
Figure 3.4 Exemple d'un graphe avec la clique maximum : (4, 5, 6, 7)

Les étapes d'exécutions :

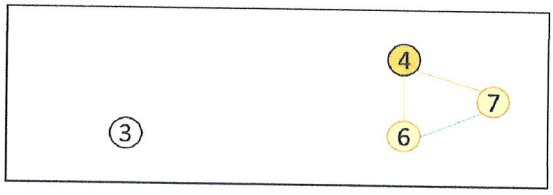
Début : C = {} Cand = {1,2,3,4,5,6,7}

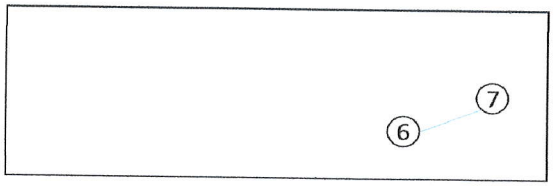
Étape 1 : Sj =n {5} C = C ∪ {Sj} = {5} Cand = {3,4,6,7}



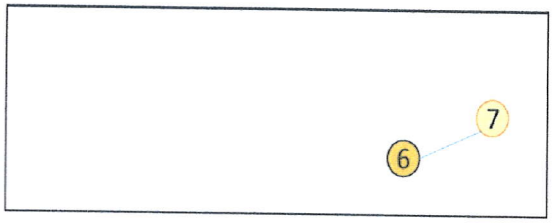


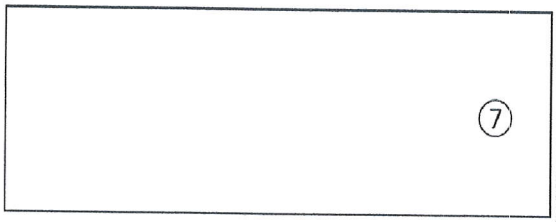
Étape 2 : Sj = {4} C = C ∪ {Sj} = {5,4} Cand = {6,7}



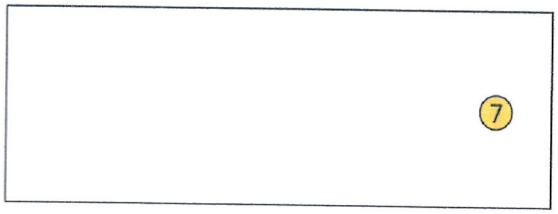


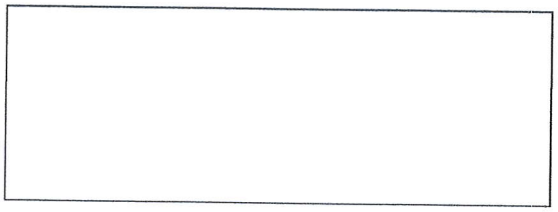
Etape 3 : Sj = {6} C = C ∪ {Sj} = {5,4,6} Cand = {7}





Etape 4 : Sj = {7} C = C ∪ {Sj} = {5,4,6,7} Cand = {}





Le clique maximum est : C = {5,4,6,7}

3.6.3 Les algorithmes Bron-Kerbosch pour énumération de la clique maximum

Les algorithmes de Bron-Kerbosch d'énumération des cliques maximales peuvent être étendus pour calculer la clique maximum. La version étendue de ces deux algorithmes illustrée par l'Algorithme 7 et l'Algorithme 8. Pour trouver la clique maximum, à chaque fois qu'on trouve une clique maximale on compare sa cardinalité par la cardinalité maximale trouvée et on garde la plus grande. A la fin de l'algorithme on ne garde que la clique dont sa cardinalité est la plus grande.

```
Tableau X, R; Graph P;
Maxclique integer;
BronKerboschsanspivot (Tableau: R, Graph P, Tableau X)
If (p. vertices = ∅  AND X = ∅ ) {
If (R.count > Maxclique) {
Ecrire (« la clique maximum est :» R)
Maxclique <- R.count }   }
If ((p. vertices ≠ ∅ )
{
  Pour (chaque sommet v dans p)  {
    R.add (v)
    Voisin <- p. collectNeighborIds (v)
    P. subgraph (P∩ Voisin)
    x.intersiction(Voisin)
    BronKerboschsanspivot (R, P, X)
    R. retirer (v)
    P<- P.subgraph (P/V)
    X.add(v)
  }
}
```

Algorithme 7 : Algorithme modifié de Bron-Kerbosch sans pivot sous Graphx

```
Array X, R;   Graph P;
BronKerboschavecpivot (Array: R, Graph P, Array X) {
Choisir le pivot X∪P
Voisin <- p. collectNeighborIds (pivot)
If (p.vertices = ∅  AND x =∅ ) {
If (R.count > Maxclique) {
Ecrire (« la clique maximum est :» R)
Maxclique <- R.count
}
}
If (P.vertices ≠ ∅ ) {
Pour (chaque sommet v dans p) {
R.add (v)
Voisin <- p. collectNeighborIds (v)
P. subgraph (P∩ Voisin)
x. intersection(Voisin)
BronKerboschavecpivot (R, P, X)
R. retirer (v)
P<- P.subgraph (P/V)
X.add(v) }
}
```

Algorithme 8 : Algorithme modifié de Bron-Kerbosch avec pivot sous Graphx

3.7 Conclusion

Le problème de clique maximale est un problème fondamental dans la théorie de graphes .Exemples d'applications peuvent être trouvés dans la conception et l'analyse du réseau, les sciences sociales, ou biologie mathématique.

Dans ce chapitre nous a permis d'approfondir nos connaissances sur les clique maximale (maximum), ça définition, ses domaine d'applications. Dans le suivant chapitre, nous allons réaliser l'implémentation des algorithmes proposés sous GraphX suivi par des évaluations expérimentales.

Chapitre 4 :

Mise en œuvre et évaluation des algorithmes

4.1 Introduction

Nous intéressons dans ce chapitre à l'installation et la configuration de l'environnement de développement à base de Spark capable d'héberger et de traiter de Big graphes. Nous allons commencer par la préparation d'un environnement virtualisé équipé d'un système d'exploitation (Linux) installer dans le Windows 7, sur lequel on va installer les différents outils nécessaires (JDK, Spark/Graphx, éclipse).puis en peut exécuter l'algorithme choisir dans un seul nœud et multi-nœuds Finalement, nous allons tester cet environnement utilisant un Benchmark de données.

4.2 Configuration de l'environnement de développement

Pour la réalisation d'un prototype nous avons choisi de travailler avec Ubuntu 14.04 qui est un système d'exploitation Linux, à cause de la stabilité de Spark sur cette plateforme.

L'installation de Spark nécessite l'installation et la configuration d'une version récente de Java (la version par défaut 8), ensuite, on va installer GraphX qui est à base de Spark, par la suite installer et configurer Eclipse avec Spark (voir la figure 4.1). Les différents logiciels nécessaires sont illustrées par le tableau 4.1, ainsi le tableau 4.2 contient plus des détails sur le matérielle et les machines virtuelles. Nous avons utilisé le langage de programmation Scala, ce dernier est un langage de programmation développé par l'équipe de Martin Odersky (le professeur de méthodes de programmation à l'EPFL en Suisse) à l'École polytechnique fédérale de Lausanne en Suisse. Scala a été développé pour offrir un langage multi-paradigme, extensible et à syntaxe concise. L'une des particularités les plus notables de Scala est qu'il est basé sur le langage Java, et hérite ainsi de ses bibliothèques et de sa machine virtuelle. Le langage Scala est disponible depuis 2003.

Le langage Scala commence à être utilisé dans l'industrie : plusieurs entreprises ont annoncées le passage de certaines de leurs applications au langage Scala, comme Twitter ou le journal The Guardian [54].

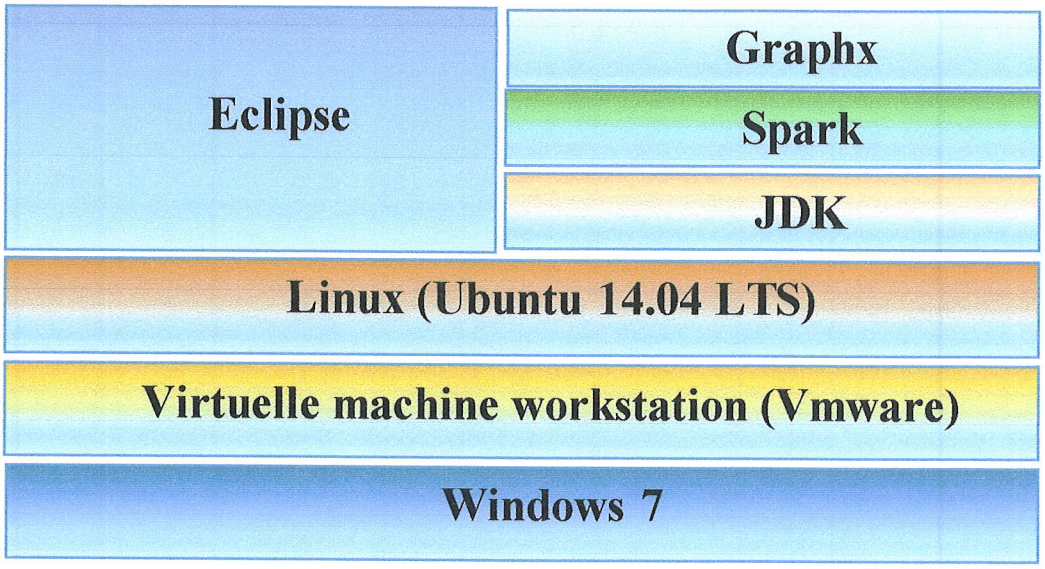


Figure 4.1 : La pile des logiciels.

Logiciel	Version	Logiciel prérequis	Remarque
System d'exploitation hôte	Windows 7	/	64 bits
Outils de virtualisation	VMware Workstation 12.1	Windows 7	http://www.vmware.com/
System d'exploitations invitées (Linux)	Ubuntu 14.04.4 LTS	VMware Workstation 12.1	slave1, slave2, slave3 : 64bits master : 64bits
Spark	Spark1.6.1	Ubuntu, Java Version8 (jdk8)	http://d3kbcqa49mib13.cloudfront.net/spark-1.6.1-bin-hadoop2.6.tgz
Eclipse	Eclipse v4.5.2 (Mars)	Ubuntu	http://download.eclipse.org/technology/epp/downloads/release/mars/2/eclipse-committers-mars-2-linux-gtk-x86_64.tar.gz

Tableau 4.1 Tableau descriptif de l'environnement.

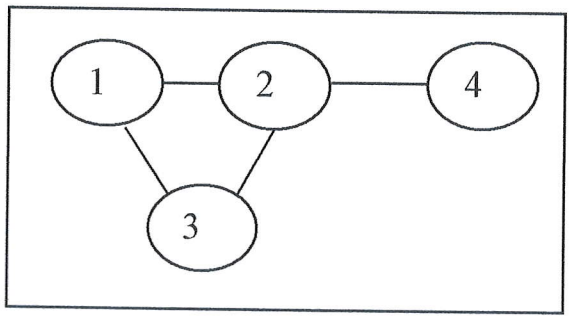
		PC
Machine physique	**Processeur**	Intel(R)core(TM)2Dou CPUE7500 @ 2.93 GHZ 2.94 GHZ2
	RAM	4 GO
	Espace disque	250 GO
Machine virtuelle Un seul Nœud	**Processeur**	2 GHZ
	RAM	2 GO
	Espace disque	20 GO

Tableau 4.2 : Tableau descriptif de matérielle et des machines virtuelles.

4.3 Exemple

Exécution de l'algorithme clique maximale dans les deux graphes suivant

► **Exécution de Graphe 1**



Graphe 1 avec 2 cliques (1, 2,3), (2,4)

1 - Fichier d'entrée

Graphe orienté

Graph1.txt ×

1 2
1 3
2 3
2 4

Graphe non orienté

Graph1non orient.txt ×

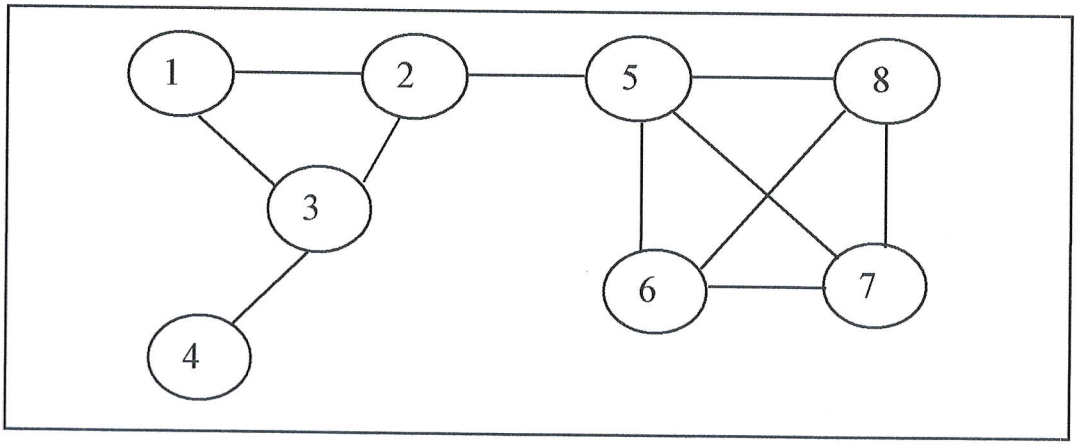
3 2
4 2
2 1
2 3
2 4
1 3
3 1
1 2

2 - Résultat d'exécution

```
16/06/06 00:15:10 INFO BlockMana
Le clique maximale est :(2,4)
Le clique maximale est :(2,1,3)
16/06/06 00:15:24 INFO RemoteActo
16/06/06 00:15:24 INFO RemoteActo
```

Figure 4.2 : Exemple d'exécution d'un graph

▶ **Exécution de Graphe 2**



Graphe 2 avec 4 cliques (1, 2,3), (3,4), (2,5), (5, 6,7, 8)

1 – Fichier d'entrée

Graphe orienté

Graph2.txt ×

```
1 2
1 3
2 3
3 4
2 5
5 7
5 6
5 8
6 7
6 8
7 8
```

Graphe non orienté

2 - Résultat d'exécution

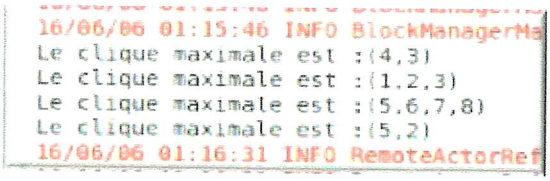


Figure 4.3 : Exemple d'exécution d'un graph

4.4 Benchmark de graphe données utilités

Pour tester notre environnement, nous allons télécharger une benchmark de données (Data set) (http://snap.stanford.edu/data) (Voir le tableau 4.3). Et nous allons générer des graphes de données en fixant de nombreux paramètre (Voir le tableau 4.4) (https://sites.google.com/site/santofortunato/inthepress), par exemple : nombre de sommets, ...etc.

• **ego-Facebook** Son jeu de données se compose de «cercles» (ou «listes d'amis») de Facebook. Les données Facebook ont été recueillies auprès des participants à l'enquête en utilisant cette application Facebook. L'ensemble de données comprend des fonctions de noeud (profils), des cercles et des réseaux de l'ego.

• **Wiki-Vote** Wikipedia est une encyclopédie libre écrite par des volontaires du monde entier. Une petite partie des contributeurs de Wikipedia sont les administrateurs, qui sont des utilisateurs ayant accès à des caractéristiques techniques supplémentaires qui facilitent la maintenance.

• **web-Google** Les noeuds représentent les pages Web et les bords dirigés représentent des hyperliens entre eux.

Non de fichier	Taille	Nombre des arcs	Nombre des sommets
facebook_combined.txt	942.6 Ko	88234	4039
twitter_combined.txt	42,4Mo	1768149	81306
web-Google.txt	71,8 Mo	5105039	875713

Tableau 4.3 Tableau descriptif des benchmark de données téléchargé.

Nom	Taille	Nombre des arcs	Nombre des sommets	Max dégrée	Moyenne dégrée
Type 1	petit-100	234	100	3	2
	petit-500	11154	500	3	2
	petit-1000	2298	1000	3	2
	petit-1500	3452	1500	3	2
	petit-2000	4572	2000	3	2
Type 2	moyen-100	472	100	10	5
	moyen-500	2372	500	10	5
	moyen-1000	4676	1000	10	5
	moyen-1500	7074	1500	10	5
	moyen-2000	9348	2000	10	5
Type 3	large-100	658	100	10	7
	large-500	3290	500	10	7
	large-1000	6674	1000	10	7
	large-1500	10118	1500	10	7
	large-2000	13288	2000	10	7

Tableau 4.4 Tableau descriptif des graphes générer.

4.5 Test de la scalabilité des algorithmes d'énumération des cliques proposé en termes de temps exécution

Les expérimentations suivantes sont réalisées dans une machine physique de 4 GO de RAM et Intel(R)core(TM)2Dou CPUE7500 @ 2.93 GHZ 2.94 GHZ2, avec une configuration single node. Nous avons exécuté les algorithmes des Cliques maximales et maximums dans Spark (pour plus de détaille sur l'installation de Spark voir l'annexe A.1.7) avec déférente benchmark de données pour ce faire nous avons utilisé deux jeux de données de tailles différentes.

4.5.1 Clique maximale

La figure 4.4 représente les résultats de l'exécution des algorithmes des cliques maximales avec pivot et sans pivot. A chaque fois on augmente le nombre de sommets et examinant le temps d'exécution.

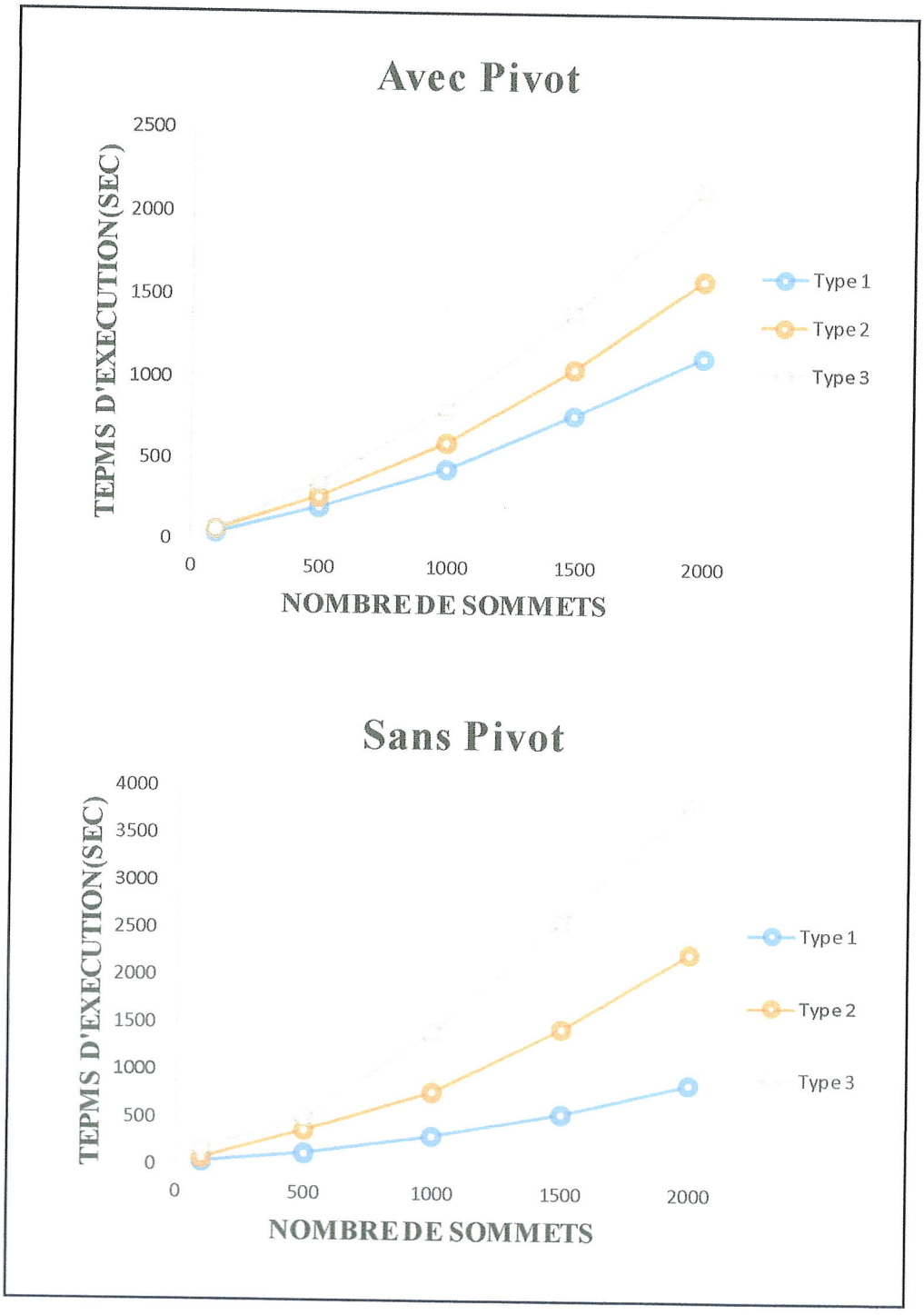
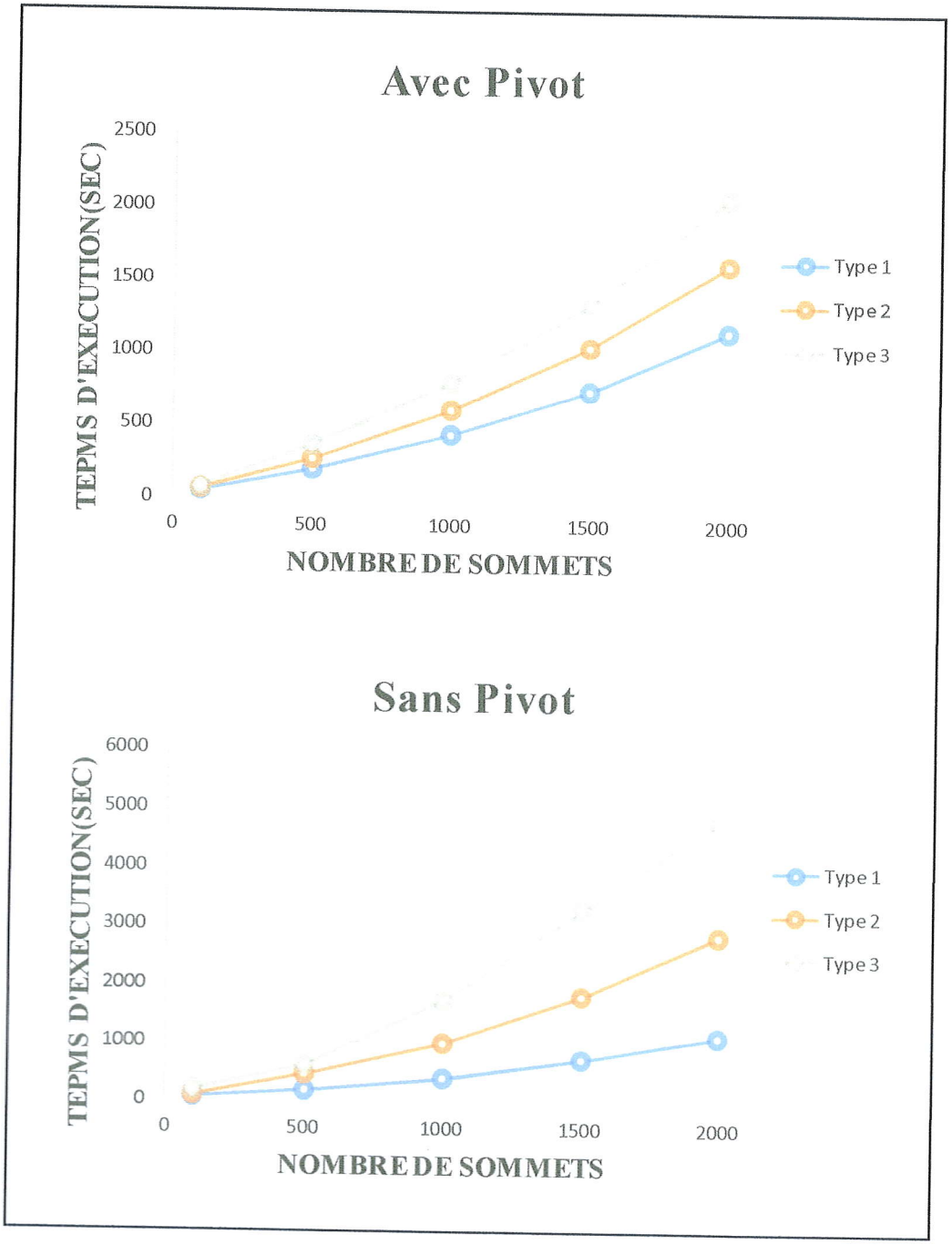


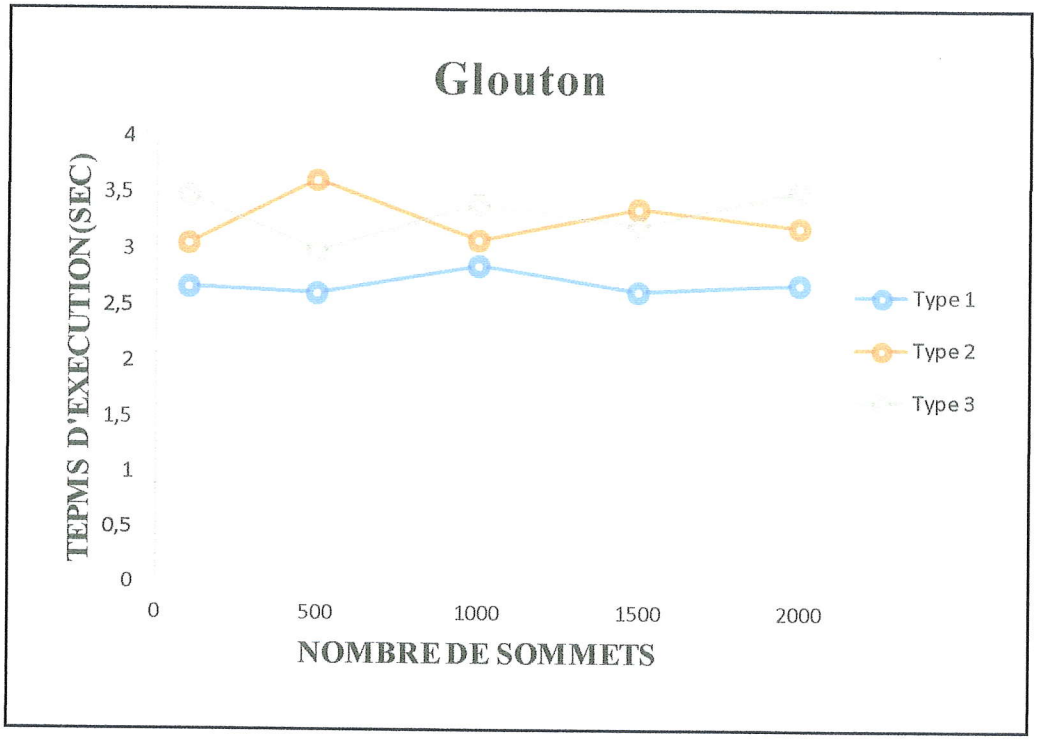
Figure 4.4 : Teste de sclabilité des algorithmes d'énumération des cliques Maximales : Nbr sommets vs temps exécution

Résultat : D'après les courbes ci-dessus, on constate que la scalabilité des algorithmes suit une fonction linéaire.

4.5.2 Clique maximum

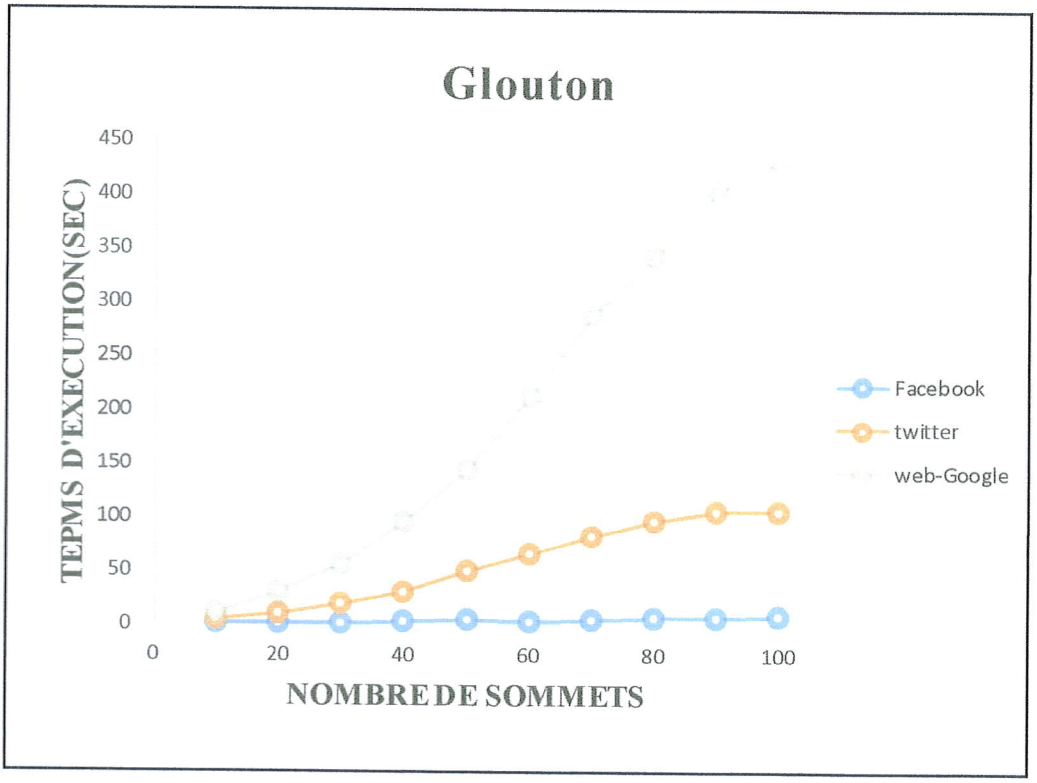
La figure 4.5 représente les résultats de l'exécution des algorithmes de la clique maximum. A chaque fois on augmente le nombre de sommets et on examine le temps d'exécution.





La figure 4.5 : Teste de sclabilité des algorithmes d'énumération des cliques Maximums : Nbr sommets vs temps exécution

Teste de scalabilité de l'algorithme Glouton en utilisant les benchmarks de données Facebook, Twitter et Google web (voir la figure 4.6).



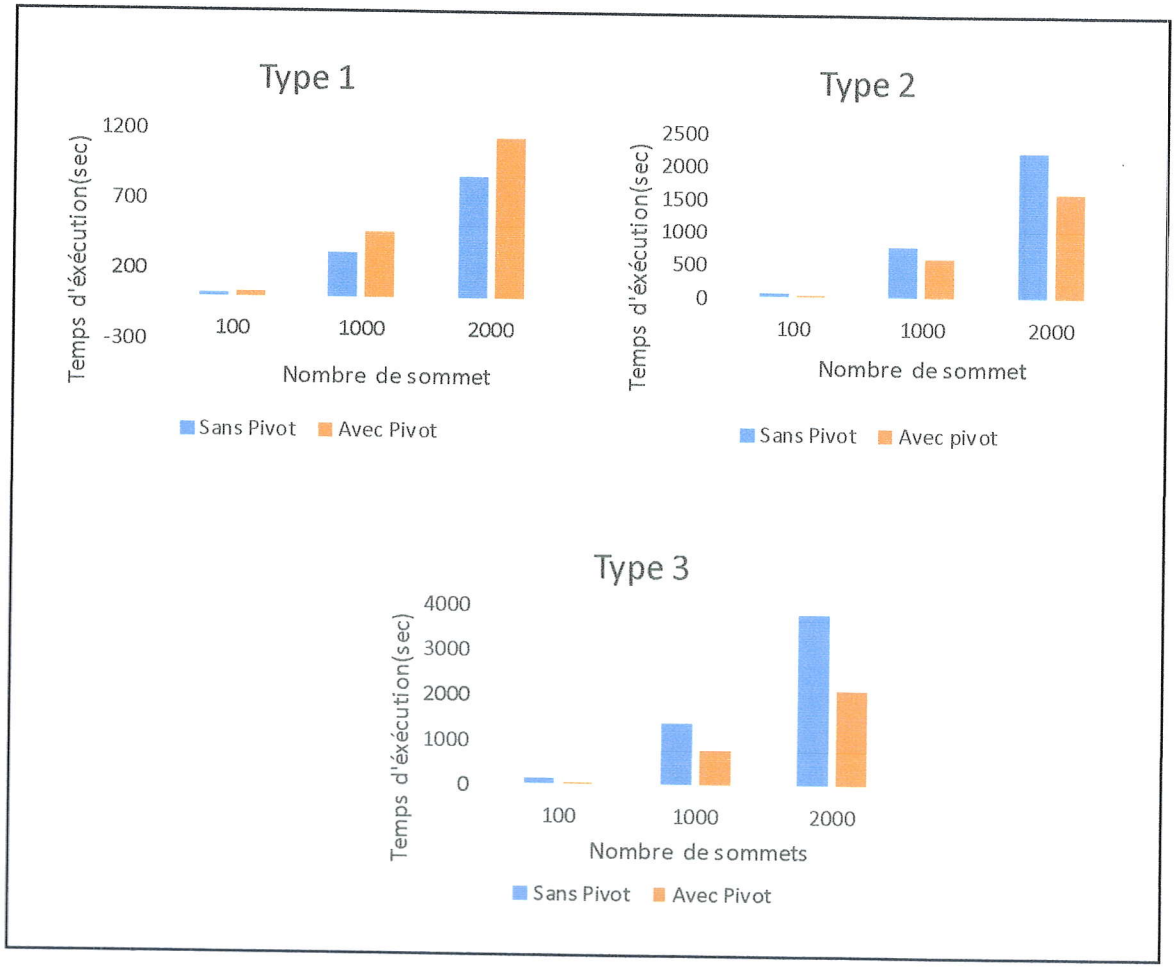
La figure 4.6 : Teste de sclabilité d'algorithme Glouton : Nbr sommets vs temps exécution

Résultat : D'après les courbes ci-dessus, on constate aussi que la scalabilité des algorithmes suit une fonction linéaire.

4.6 Comparaison single node

4.6.1 Clique maximale

La figure 4.7 représente une comparaison entre les algorithmes d'énumération des cliques maximales. A chaque fois on augmente le nombre de sommets, et on examine le temps d'exécution et on le compare entre les différents algorithmes.

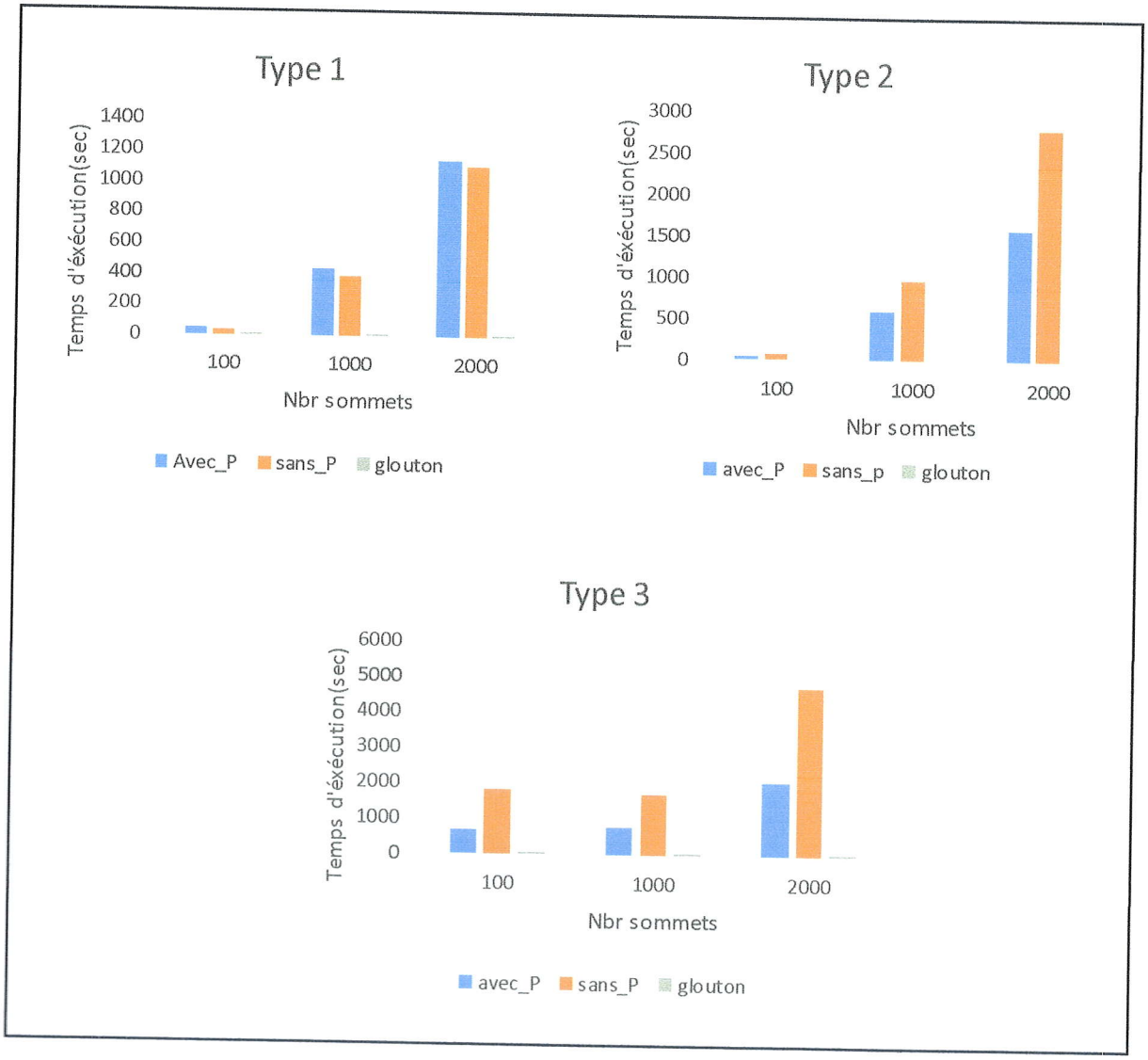


La figure 4.7 : comparaison entre les d'énumération des cliques maximales

Résultat : En observant les histogrammes décrits dans la figure 4.7, nous concluons que le meilleur algorithme dans le type 1 est l'algorithme sans pivot. Par contre, l'algorithme avec pivot donne des meilleurs résultats dans le type 2 et 3. Donc on constate que lorsqu'on augmente le nombre des arcs, le temps d'exécution de l'algorithme sans pivot est supérieur à celui de l'algorithme avec pivot car ce dernier parcourt seulement les sommets de plus fort dégrée.

4.6.2 Clique maximum

La figure 4.8 représente la comparaison entre les algorithmes de clique maximums. A chaque fois on augmenter le nombre de sommets, on examine le temps d'exécution et on le compare entre les différents algorithmes.



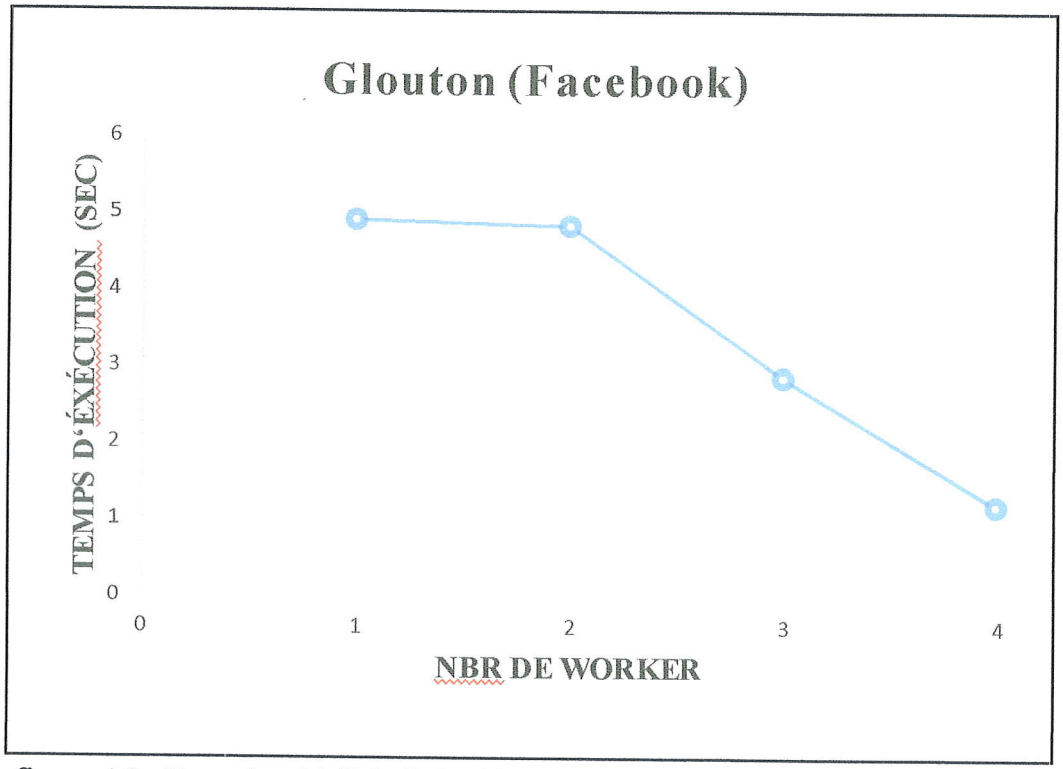
La figure 4.8 : Comparaison entre les algorithmes d'énumération des cliques maximums

Résultat : En observant les histogrammes décrits dans la figure 4.8, nous concluons que le meilleur algorithme en termes de la latence est l'algorithme glouton, car cet algorithme est une heuristique dont le temps d'exécution est très réduit par rapport à celui des autres algorithmes exacts, car il minimise l'espace de recherche, mais le seul bémol c'est que les résultats ne sont pas toujours optimaux.

4.7. Teste de scalabilité multi-nœud (cluster)

Ce teste a été effectué dans un cluster multi-nœud dont la description et la démarche de l'installation et configuration sont tous décrit dans l'annexe A.1. Ce cluster comprend un ensemble de 4 machines physiques sur lesquelles nous avons exécuté notre algorithme parallèle glouton en utilisant le benchmark de données de réseau social Facebook.

La figure 4.9 représente les résultats de l'exécution. A chaque fois on augmente le nombre de worker (machine) et on examine le temps d'exécution.



La figure 4.9 : Teste de sclabilité d'algorithme Glouton : Nbr Worker vs temps exécution

Résultat : En observant la courbe décrite dans la figure 4.9, nous observons qu'à chaque augmentation de nombre de worker, le temps d'exécution se réduit.

4.8. Conclusion

Dans ce chapitre, nous avons décrit les différents environnements de l'exécution et les benchmarks de données utilisés, puis on a effectué des tests de scalabilité des algorithmes d'énumération des cliques maximales et maximums, puis on a comparé les résultats sous GraphX. Les résultats obtenus montrent que les algorithmes parallèles proposés sont scalables.

Conclusion générale

Notre travail a fait l'objet d'une étude de nouvelles technologies de traitement et d'analyse de Big Graph. Nous avons pris connaissance de la puissance des graphes comme outil de modélisation dans divers domaines d'application, notamment en informatique. Nous nous sommes intéressés au framework Spark. La partie applicative de ce travail consistait, d'un part, à un déploiement d'un cluster GraphX dans un environnement parallèle à base de Spark, et d'autre part, à l'adaptation et l'implémentation de quelque algorithmes destiné à la résolution d'un problème d'analyse de graphes afin de les exécuter et les évaluer à travers ce cluster.

Pour cela, nous avons réalisé une description détaillée de cette partie, et avons été confrontées à pas mal de difficultés lors de l'implémentation de nos algorithmes en Scala (que nous avons choisi comme langage de programmation à cause des APIs de graphes existantes), ou à travers nos machines de cluster, à cause des ressources mémoire et processeurs limitées (néanmoins, la communication réseau inter-machines n'était pas un souci).

A la fin de ce travail, nous constatons que la mise en œuvre d'un cluster Spark GraphX, l'exécution des algorithmes parallélisés dans une architectures à plusieurs nœuds, et l'évaluation des différents statistiques des opérations effectuées, nous ont permis d'approuver l'impact de la distribution et le parallélisme sur l'efficacité des systèmes traitant de grands volumes de données de graphes. Ceci en distribuant les tâches de traitement pour garantir un meilleur temps d'exécution, et en répliquant les données de graphes sur les différents nœuds de réseau afin d'assurer la disponibilité et la tolérance aux pannes.

Perceptives

Le domaine de Big Graph reste très ouvert aux différents travaux et différents tests dans des environnements fortement distribuées. Néanmoins, la future étape qui se voit très intéressante, est la mise en œuvre d'un cluster Spark sur des machines physiques assez puissantes (i.e. des serveurs) pour pallier le manque de ressources matérielles que nous avons rencontré au cours de la réalisation de ce travail. De plus, la démonstration des points forts du framework Spark (y inclus GraphX), à savoir la performance, la tolérance aux pannes et l'extensibilité, en appliquant plusieurs tests sur des graphes très volumineux de plusieurs giga-octets sur des dizaines de nœuds de cluster, voire des centaines de nœuds, optimisation de l'algorithme proposer et la comparaison entre d'autre algorithmes d'énumérations des cliques maximales, peut faire l'objet d'autres prochains thèmes de recherche.

Bibliographie

[1] **MasterTheBoss**. Publié le 21 juin 2013 - Mis à jour le 29 juin 2013 .Dans le Cloud computing - le tutoriel pour débutant

[2] **SH. Lazare et F. Barthélemy** Introduction Big Data. Réf: AXIO_BD_V1

[3] **François PÊCHEUX**, « CLOUD COMPUTING ou INFORMATIQUE DANS LES NUAGES», Encyclopædia Universalis , consulté le 17 mai 2016.

[4] **Pascal Beramis / Ingenieur d'Affaires,** CLOUD COMPUTING : Enjeux de compétitivité et valeurs ajoutées en univers 2.0

[5] Comment travailler sur une grappe de serveurs (cluster) , jeu 12 Juil 2012

[6] **Renaud Vayssad.** Le clustering sous Linux - Quest ce qu'un cluster, 13 janvier 2002, consulté le 13 avril 2015

[7] https://www.vmware.com/fr/virtualization/overview.html(Consulter le 20 fevrier 2016)

[8] https://www.vmware.com/fr/virtualization/getting-started.html(Consulter le 20 février 2016)

[9] Avantages et risques liés aux stockages définis par logiciel (Software Defined Storages, SDS) 25 août 2015
http://blog.ontrack.fr/avantages-et-risques-lies-aux-stockages-definis-par-logiciel-software-defined-storages-sds/ (Consulter le 25 février 2016)

[10] **Dominique Filippone** La virtualisation du poste de travail au cœur des enjeux des DSI 18/01/11

[11] **Rajkumar Buyya, James Broberg, Andrzej M. Goscinski**, Cloud Computing : Principles and Paradigms, John Wiley & Sons, 2010 (ISBN 9781118002209) March 2011.

[12] Club Des Responsables d'Infrastructure Et De Production, L'OBSERVATOIRE Des Directeurs d'Infrastructures Et De Production, CLOUD COMPUTING Définitions & Concepts, Enquête Et Analyse Des Tendances, Juin 2010

[13] http : //www.petite-entreprise.net/P-3714-83-G1-le-cloud-computing-les-avantages-et-les-inconvenients.html, Le cloud computing : les avantages et les inconvénients

Publié le vendredi 12 avril 2013 (Consulter le 25 février 2016)

[14] https://missarte.wordpress.com/les-inconvenients-du-Cloud-computing/ (Consulter le 25 février 2016)

[15]https://www.google.dz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=26&cad=rja&uact=8&ved=0ahUKEwjGprTtYrNAhUHuBQKHbmrDvM4FBAWCEMwBQ&url=https%3A%2F%2Fwww.ida.gov.sg%2F~%2Fmedia%2FFiles%2FInfocomm%2520Landscape%2FTechnology%2FTechnologyRoadmap%2FBigData.pdf&usg=AFQjCNGvJqALZv-IP6O141SlLwP3Az2Eyg&sig2=_TZxpHzJQrDXNKyztVFTxA&bvm=bv.123664746,d.bGg

[16] Big data l'accélérateur d'innovation, Livre blanc de l'institut G9+, en partenariat avec renaissance numérique 16 déc. 2014

[17] **Eric Alain FROIDEFOND**, les travaux de l'enass, école nationale d'assurances, Le big data dans l'assurance, le site www.enass.fr

[19] **Alain Fernandez** , La technologie mise en oeuvre "sous" le Big Data : http://www.piloter.org/business-intelligence/technologie-big-data.htm (Consulter 17/05/16)

[20] http://www.orange.com/fr/actualites/2015/novembre/Les-6-defis-du-Big-Data (Consulter le 25 février 2016)

[21] **Paul Burckhardt**, Big Graphe, Date de publication : 28 novembre 2014 | Dernière mise à jour : 18 décembre 2014 | Dernière révision : 18 décembre 2014

[22] http://www.irisa.fr/celtique/genet/GEN/q1D.pdf

[23] **Yue Zhao,Kenji Yoshigoe,1 Mengjun Xie,1 Suijian Zhou, Remzi Seker and Jiang Bian3.** Evaluation and Analysis of Distributed Graph-Parallel Processing Frameworks. Reçu 15 Juin 2014; Accepté 20 Août 2014 Publication 7 Octobre 2014

[24] Étude d'une architecture MapReduce tolérant les fautes byzantines, RenPar'20 / SympA'14, CFSE 8, Saint-Malo, France, du 10 au 13 mai 2011

[25] **Maria Malek**Implémentation parallèle de certains algorithmes defouille de donnees avec le framework MapReduce Algorithmes : K-means et Apriori

[26] Panorama des solutions de big data Principe de fonctionnement de MapReduce, /04/07/13.

[27] **Khaled TANNIR** Doctorant CIFRE LARIS/ESTI, MapReduce Algorithme de parallélisassions des traitements, / 2e SéRI 2010-2011 Jeudi 17 mars 2011

[28] http://fr.hortonworks.com/hadoop/ . (Consulter le 2 février 2016)

[29] **Julien Delhomme**, Traitements Big Data avec Apache Spark - 1ère partie : Introduction, Écrit par Srini Penchikala , le 03 mars 2015.

[30] **Claudio Martella**, **Roman Shaposhnik**, and **Dionysios Logothetis**, Practical Graph Analytics with Apache Giraph, 2015

[31] https://fr.wikipedia.org/wiki/Giraph (Consulter le 26 février 2016)

[32] **Prashant Raghav** Using Pregel To Calculate Page Rank Of A Webpage A Bulk Synchronous Parallel Programming Approach.

[33] **Joseph E. Gonzalez** ,**Yucheng Low**, **Haijie Gu** , **Danny Bickson**, **Carlos Guestrin** ,PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs 2012

[34] **Maurice Audin**,Etat de l'art du Cloud Computing et adaptation au Logiciel Libre 2009

[35] **Xin, R. S., Gonzalez, J. E., Crankshaw, D., Franklin, M. J., Dave, A., & Stoica, I.** GraphX: Unifying Data-Parallel and Graph-Parallel Analytics. UC Berkeley AMPLab (2014, Février).

[36] https://mcherif.wordpress.com/2015/05/21/big-data-spark-concepts-generaux (Consulter le 27 février 2016)

[37] Site officiel de projet GraphX : http://amplab.github.io/graphx/ (Consulter le 27 février 2016)

[38] **Michael S. Malak and Robin East** Spark GraphX in Action (Book) Spark GraphX in Action June 2016

[39] **Michel Crucianu, Pierre Cubaud, Raphaël Fournier-S'niehotta, Marin Ferecatu - Cnam.** Mis à jour le Jun 08, 2016.

[40] http://www.technologies-methodes-it.com/big-data-apache-spark-prochaine-big-plateforme / (consulter 1 avril 2016)

[41] http://blog.jetoile.fr/2014/05/rdd-quest-ce-que-cest.html (consulter 1 avril 2016)

[42] Documentation officiel de Spark:http://spark.apache.org/docs/latest/graphx-programming-guide.html#graph-algorithms (consulter 5 avril 2016)

[43]https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/lib/PageRank.scala (consulter 2 mars 2016)

[44] **Joseph E. Gonzalez, Reynold S. Xiny, Ankur Dave, Daniel Crankshaw Michael J. Franklin, Ion Stoicay** UC Berkeley AMPLab yDatabricks GraphX: Graph Processing in a Distributed Dataflow Framework

[45] **Matei Z., Mosharaf Ch., Tathagata D., Ankur D., Justin M., Murphy Mc., Michael J., Scott S., Ion S,** Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

[46] **Michel Crampes - Michel Plantié - Julien Bidault**. Cliques maximales d'un graphe et treillis de Galois. 2011

[47] **Laurent Beauguitte**. Cliques, communautés et dérivées. Janvier 2011.

[48] **J. Jeffry Howbert (jhowbert), Jacki Roberts (jackirob)**, The Maximum Clique Problem, CSEP 521 Applied Algorithms Winter 2007.

[49] Algorithmique et Programmation .Projet : Algorithme de Bron-Kerbosch pour Maximum-Clique .Ecole normale supérieure Département d'informatique algoL3@di.ens.fr 2013-2014

[50] **Alessio Conte.** Review of the Bron-Kerbosch algorithm and variations School of Computing Science .Sir Alwyn Williams Building.University of Glasgow G12 8QQ .Level 4 Project— May 05, 2013

[51] **David Eppstein, Maarten L'offler, and Darren Strash** .Listing All Maximal Cliques in Sparse Graphs in Near-optimal Time. Department of Computer Science, University of California, Irvine, USA

[52] **Alessio Conte, Roberto De Virgilio, Antonio Maccioni2, Maurizio Patrignani2, Riccardo Torlone**. Finding All Maximal Cliques in Very Large Social Networks Université di Pisa, Pisa, Italy conte@di.unipi.it .Université Roma Tre, Rome, Italy {fdvr, maccioni, patrigna, torloneg@dia.uniroma3.it}

[53] **J. Jeffry Howbert (jhowbert), Jacki Roberts (jackirob)** CSEP 521 Applied Algorithms Winter 2007

[54] http://www-igm.univ-mlv.fr/~dr/XPOSE2011/le_langage_scala/presentation.html (consulter 25 avril 2016)

Annexes

A.1. Installation et configuration d'un cluster Spark en mode standalone

Apache Spark est un framework de calcul distribué à base mémoire qui offre une grande flexibilité en terme de développement des applications et de leurs déploiements sur un cluster de machines ou un Cloud. L'un des avantages de développer une application avec Spark est de pouvoir passer à l'échelle en ajoutant plus de machines au cluster d'exécution, c.-à-d. développer l'application localement sur une seule machine utilisant des APIs de haut-niveau, puis la déployer plus tard sur un cluster d'un grand nombre de machines sans avoir à changer de code. Un autre avantage, c'est que Spark peut être s'exécuté sur plusieurs gestionnaires de cluster, tel que Hadoop YARN et Apache Mesos, ou même sur une infrastructure Cloud, tel que Amazon EC2. De plus, Spark possède son propre gestionnaire de cluster, dite « Spark en mode standalone ». Dans le cadre de ce travail, nous avons adopté le mode standalone de Spark comme un cluster d'exécution pour nos algorithmes.

A.1.1. Architecture de cluster Spark utilisée

Nous avons conçu l'architecture de cluster suivante (voir la Figure A.1) qui comprend un réseau local de 4 machines interconnectées à l'aide d'un switch.

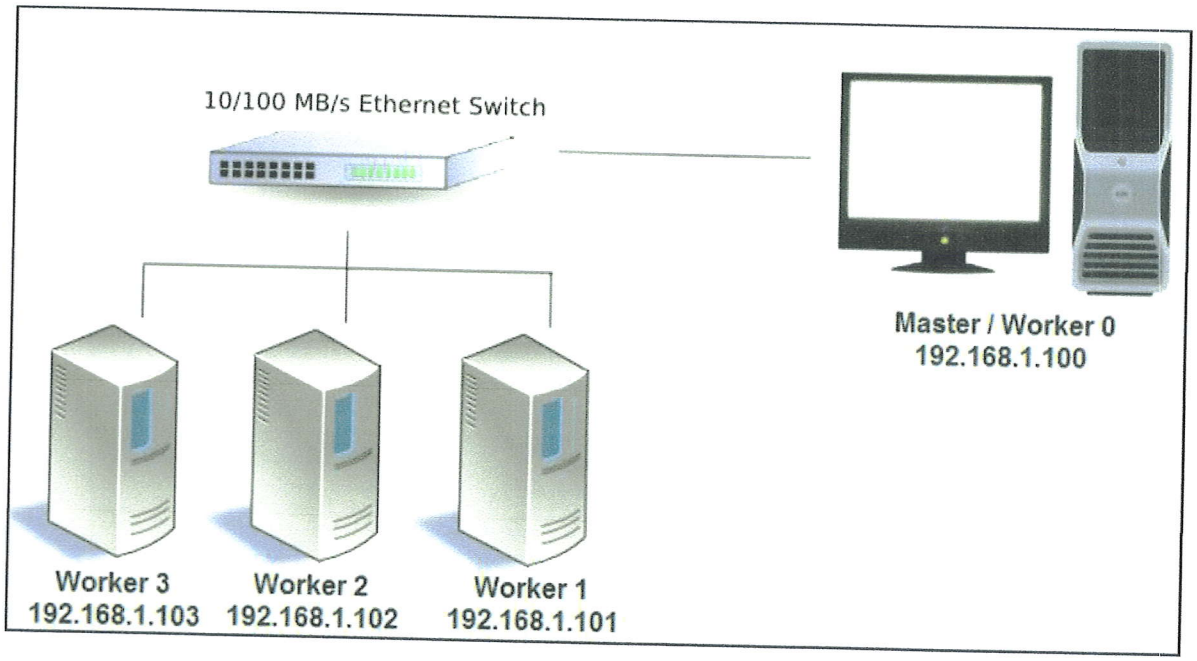


Figure A.1. Schéma de l'architecture de cluster Spark.

La table suivante (Table A.1) décrit le rôle et la configuration de chaque machine de ce cluster :

	Nœuds			
Rôle	**Master/Worker 0***	**Worker 1**	**Worker 2**	**Worker 3**
Catégorie	PC Portable	PC Bureautique	PC Bureautique	PC Bureautique
Processeur	Intel Core i7-2630QM 2.9GHz	Intel Core 2 Duo E7500 2.93GHz	Intel Core 2 Duo E7500 2.93GHz	Intel Core 2 Duo E7500 2.93GHz
Mémoire vive	8192MB	4096MB	4096MB	4096MB
Stockage (Disque dur)	SSD 256GB	HDD 500GB (5400tr/min)	HDD 500GB (5400tr/min)	HDD 500GB (5400tr/min)
Carte Réseau	Ethernet 10/100Mbps	Ethernet 10/100Mbps	Ethernet 10/100Mbps	Ethernet 10/100Mbps
Système d'exploitation	Ubuntu 14.04.4 LTS	Ubuntu 14.04.4 LTS	Ubuntu 14.04.4 LTS	Ubuntu 14.04.4 LTS
Architecture	64-bit	64-bit	64-bit	64-bit
Nom d'hôte	ubuntu-master	ubuntu-worker1	ubuntu-worker2	ubuntu-worker3
Compte d'utilisateur	master2rs	master2rs	master2rs	master2rs
Mot de passe	master2rs	master2rs	master2rs	master2rs
Adresse IP	192.168.1.100	192.168.1.101	192.168.1.102	192.168.1.103
Version Spark	v1.6.1	v1.6.1	v1.6.1	v1.6.1
Version Java	Oracle JDK 8u91	Oracle JDK 8u91	Oracle JDK 8u91	Oracle JDK 8u91
Version IDE	Eclipse v4.5.2 (Mars)	n/a	n/a	n/a

Table A.1. Configuration logicielle et matérielle de l'environemet.

Le nœud Master joue lui-même le rôle d'un Worker, au même temps il est responsable de la gestion de cluster et l'ordonnancement des tâches entre les nœuds.

A.1.2. Conditions préalables

Certains paramètres (voir la Table A.1) doivent absolument être identiques sur tous les nœuds :

- L'architecture et la version du système d'exploitation (i.e. Ubuntu 14.04.4 LTS 64-bit)
- Le nom et le mot de passe de compte d'utilisateur (i.e. master2rs/master2rs), qui doivent être précisés pendant la phase d'installation du système d'exploitation.
- La version d'Oracle Java JDK installé (i.e. JDK 8u91).
- La version de Spark installé (i.e. v1.6.1).
- L'emplacement d'installation de Spark (i.e. /home/master2rs/spark).

- Les adresses IP doivent appartenir au même réseau local et être fixées d'une manière statique sur chaque nœud (cependant, chacun doit avoir un nom d'hôte unique).

A.1.3. Installation d'Oracle Java JDK 8

Spark et Eclipse s'exécutent sur des machines virtuelles Java (JVM). L'installation préalable d'Oracle JDK 8 est nécessaire. Les commandes suivantes sont à exécuter sur chaque nœud :

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
sudo apt-get install oracle-java8-set-default
```

Une fois bien installé, on peut vérifier la version de Java via la commande "java –version" :

```
master2rs@ubuntu-master:~$ java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) Server VM (build 25.91-b14, mixed mode)
```

A.1.4. Installation d'Eclipse

Eclipse est l'environnent de développent intégré (IDE) que nous avons utilisé pour développer nos algorithmes en langage Scala. Eclipse s'appuie principalement sur Java, mais nous pouvons lui ajouter le support de langage Scala via les extensions « **Scala IDE** » et « **m2eclipse-scala** ». On commence par l'installation de l'Eclipse sur le nœud Master :

```
# Télécharger le fichier d'installation d'Eclipse
wget http://download.eclipse.org/technology/epp/downloads/release/mars/2/eclipse-committers-mars-2-linux-gtk-x86_64.tar.gz
# Extraire le contenu du fichier téléchargé
tar -zxf eclipse-committers-mars-2-linux-gtk-x86_64.tar.gz
# Créer un nouveau fichier texte 'eclipse.desktop'
gedit eclipse.desktop
```

L'éditeur de texte se lance avec un fichier "eclipse.desktop" initialement vide. Les informations contenues dans ce fichier permettront de configurer l'installation de l'Eclipse (voir la Figure A.2).

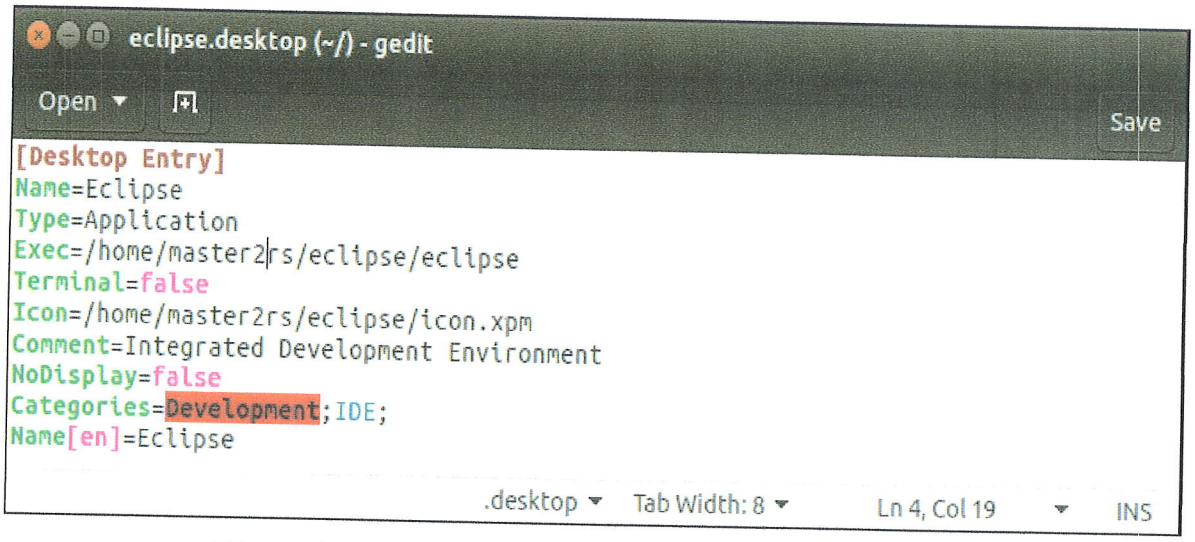


Figure A.2. Texte à remplir dans le fichier "eclipse.desktop".

Une fois terminé, nous enregistrons et fermons le fichier, puis on termine l'installation par les commandes suivantes :

```
# Finaliser l'installation d'Eclipse
sudo desktop-file-install eclipse.desktop
sudo ln -s /home/master2rs/eclipse/eclipse /usr/bin/eclipse
sudo cp /home/master2rs/eclipse/icon.xpm /usr/share/pixmaps/eclipse.xpm
# Lancer Eclipse depuis le terminal, ou bien chercher 'Eclipse'
# dans le menu démarrer d'Ubuntu
eclipse
```

Le premier lancement d'Eclipse affiche la fenêtre du choix de l'emplacement de l'espace de travail (voir Figure la A.3) où nos projets seront sauvegardés. On coche la case et on clique OK.

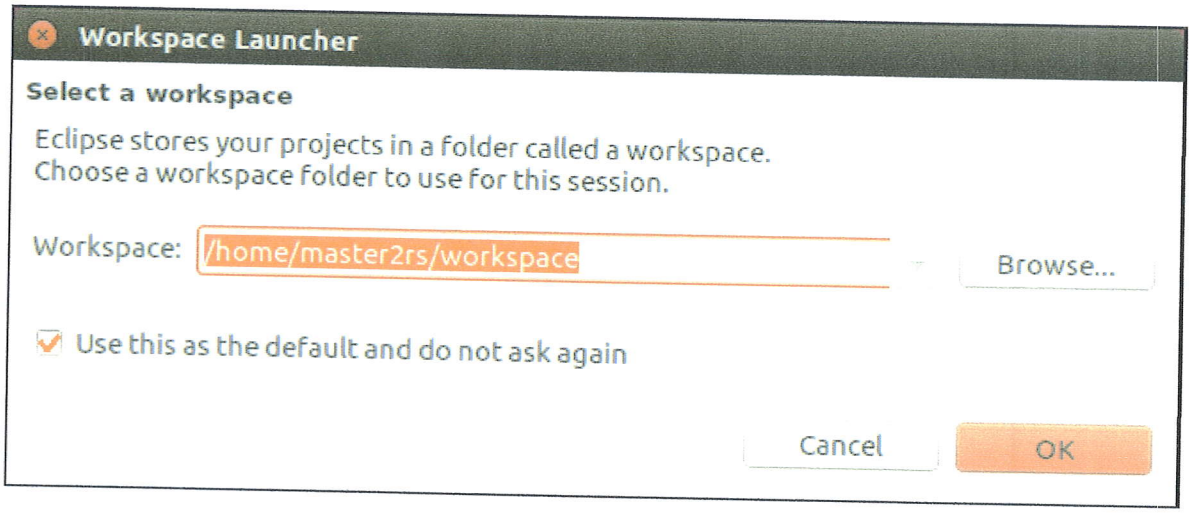


Figure A.3. Fenêtre de choix du l'emplacement du workspace Eclipse.

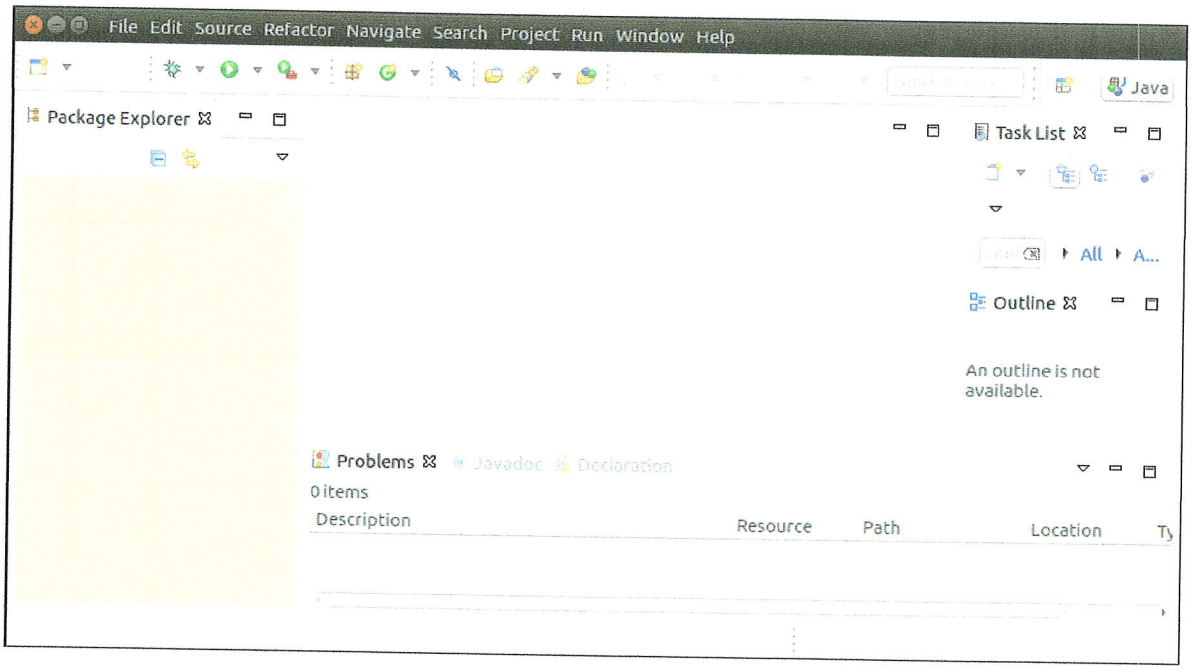


Figure A.4. Premier lancement de l'Eclipse.

A.1.5. Installation de l'extension « Scala IDE »

Eclipse ne supporte pas nativement le langage Scala, la raison pour laquelle se trouve l'extension « Scala IDE » qui s'ajoute comme étant un plugin à Eclipse et qui permet l'utilisation des APIs Scala au sein de l'Eclipse pour le développement et l'exécution des applications écrits en Scala, ou même en deux langages Java et Scala.

Nous pouvons installer le plugin directement depuis Eclipse en parcourant le menu Help -> Install New Software -> Add, et on saisit les champs suivants (voir la Figure la A.5) :

Name :	Scala IDE
Location :	http://download.scala-ide.org/sdk/lithium/e44/scala211/stable/site

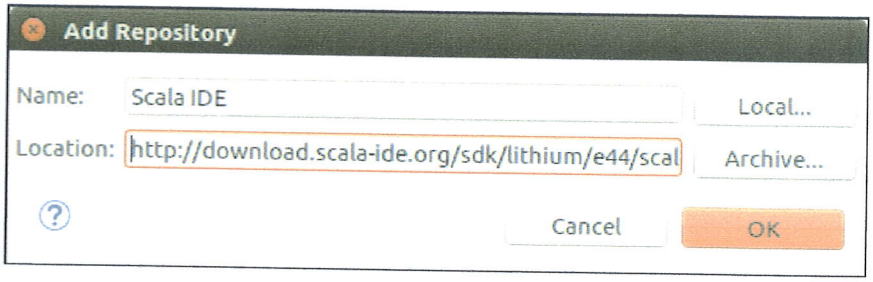


Figure A.5. Lien d'installation de l'extension Scala IDE.

Dans la fenêtre qui suit, on séléctionne le composant « **Scala IDE for Eclipse** » et on suit les étapes de l'installation jusqu'à la fin (une connexion Internet est requise).

A.1.6. Installation de l'extension « m2eclipse-scala »

A cause de certaines spécificités du langage Scala, une extension additionnelle doit être installée, qui ajoute un meilleur support de ce langage au sein de l'Eclipse. Les étapes d'installation sont similaires à celles de l'extension « Scala IDE » : menu Help -> Install New Software -> Add, et on saisit les champs suivants (voir la Figure A.6) :

Name :	m2eclipse-scala
Location :	http://alchim31.free.fr/m2e-scala/update-site/

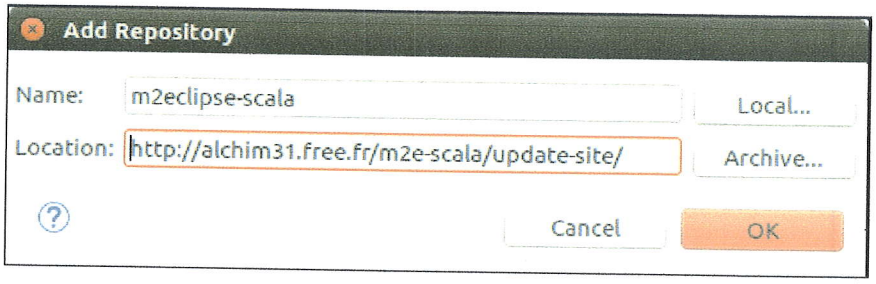


Figure A.6. Lien d'installation de l'extension m2eclipse-scala.

Dans la fenêtre qui suit, on séléctionne le composant « **Maven Integration for Scala IDE** » et on suit les étapes de l'installation jusqu'à la fin (une connexion Internet est requise).

Dès que les deux extensions sont bien insallées, un redémarage de l'Eclispe est nécessaire, et une nouvelle barre d'outils pour le developpement Scala s'affiche au prochaine lancement de l'IDE (voir la Figure la A.7). Eclipse est maintenant prêt pour développer en Scala.

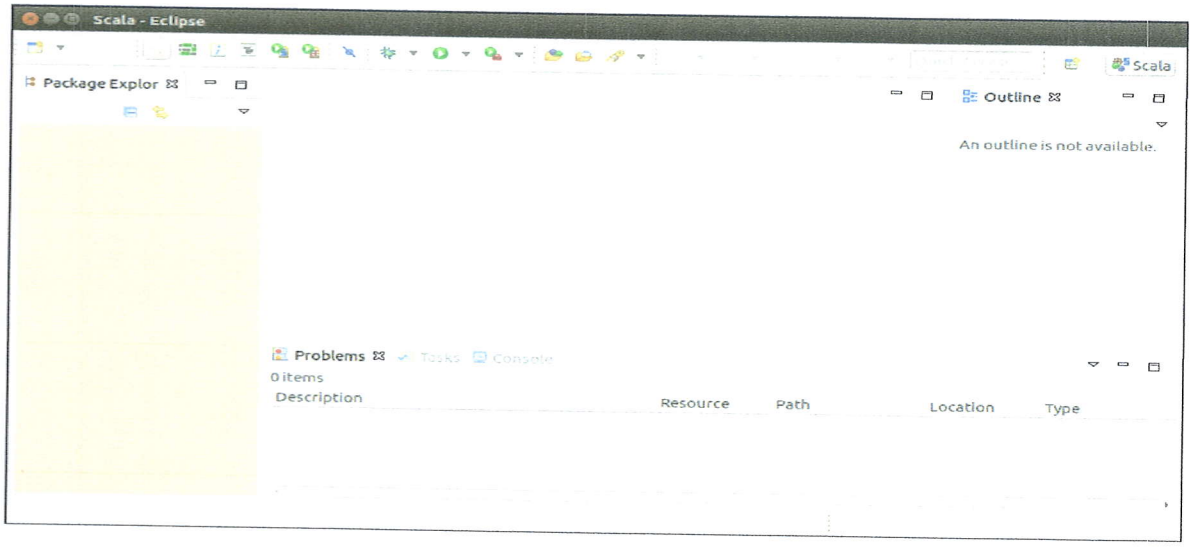


Figure A.7. La nouvelle interface Eclipse pour le développement Scala.

A.1.7. Installation de Spark

La même version de Spark doit être installé et configuré sur tous les machines. Téléchargez le package « Spark v1.6.1 Pre-built for Hadoop 2.6 » depuis le site officiel d'Apache Spark (voir Figure la A.8) : http://spark.apache.org/downloads

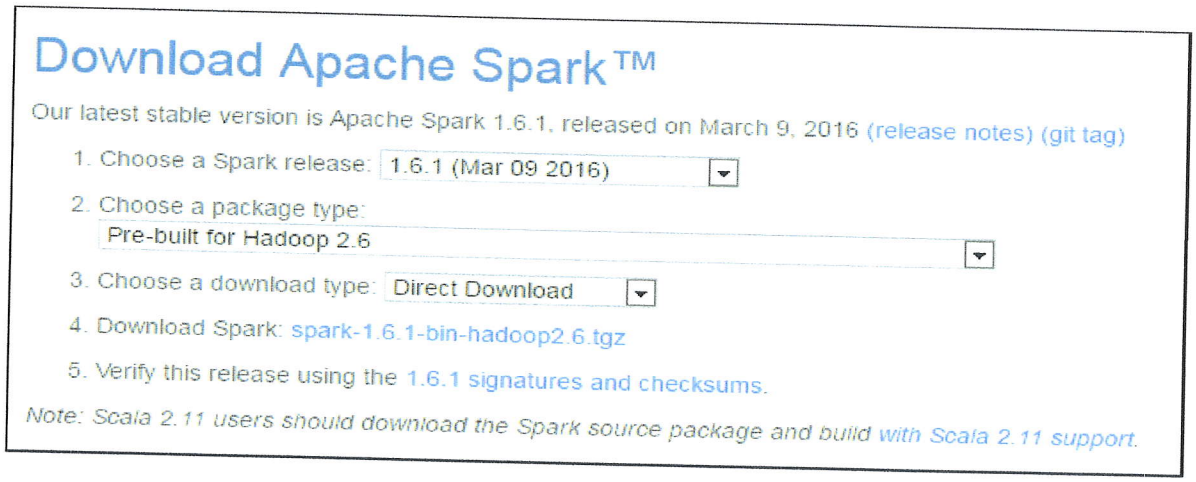


Figure A.8. Site officiel d'Apache Spark.

Lancez une fenêtre de terminal et exécutez les commandes suivantes sur chaque machine :

```
# Télécharger le package d'installation
wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.1-bin-hadoop2.6.tgz
# Extraire le package
tar -zxf spark-1.6.1-bin-hadoop2.6.tgz
# Renommer le dossier 'spark-1.6.1-bin-hadoop2.6' vers 'spark'
mv spark-1.6.1-bin-hadoop2.6 spark
# Configurer la variable d'environnement SPARK_HOME
gedit ~/.profile
```

Utilisez l'éditeur de texte et ajoutez la ligne suivante à la fin du fichier ".profile" :

```
export SPARK_HOME="$HOME/spark"
```

L'installation de Spark est terminée, redémarrez les machines pour que les changements prennent effet. Par la suite, nous procédons à l'étape de configuration de Spark.

A.1.8. Configuration de l'accès SSH (SSH password-less access)

Pour simplifier la tâche de communication, Spark exige que le Master doive avoir un droit d'accès sans mot de passe à tous les Workers via le protocole SSH. Pour garantir ce droit d'accès, une clé privé SSH doit être générée sur le Master et puis transférée vers chaque Worker.

Premièrement, un serveur SSH doit être installé sur chaque Worker :

```
# Installer 'openssh-server' sur chaque Wokrer :
sudo apt-get update
sudo install openssh-server
```

Ensuite, sur le Master, lancez un terminal et exécutez les commandes suivantes :

```
# Générer une clé privé SSH. Tapez la touche Entrée quand
# le nom de ficher et le mot de passe sont demandés.
ssh-keygen
# Parcourir le dossier '.ssh' où la clé est stockée :
cd .ssh
# Copier la clé vers chaque Worker :
ssh-copy-id -i id_rsa.pub master2rs@192.168.1.101
ssh-copy-id -i id_rsa.pub master2rs@192.168.1.102
ssh-copy-id -i id_rsa.pub master2rs@192.168.1.103
# Copier la clé vers le Master lui-même (Worker 0):
cat id_rsa.pub >> authorized_keys
```

Le résultat de ces commandes doit être similaire à celui de la Figure A.9 :

```
master2rs@ubuntu-master:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/master2rs/.ssh/id_rsa):
Created directory '/home/master2rs/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/master2rs/.ssh/id_rsa.
Your public key has been saved in /home/master2rs/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:eQQ89SgC0wj0vL+a8hLGfnAWaF+/TlWDFlMamQ+nyeQ master2rs@ubuntu-master
The key's randomart image is:
+---[RSA 2048]----+
|  ..oo ....      |
| .o .o. o. o     |
|.*.o o. ..o .    |
|+=O...+. +       |
|oE+.oo.oS..      |
|.+ =.  .+.       |
|o.=    ..        |
|o. +  ..         |
| +=.o...         |
+----[SHA256]-----+
master2rs@ubuntu-master:~$ cd .ssh
master2rs@ubuntu-master:~/.ssh$ ssh-copy-id -i id_rsa.pub master2rs@192.168.1.101
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "id_rsa.pub"
The authenticity of host '192.168.1.101 (192.168.1.101)' can't be established.
ECDSA key fingerprint is SHA256:e31aMTROUZrDuRoj1MZOM1ssCO3lhaqe5iY7k3xo7hE.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
master2rs@192.168.1.101's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'master2rs@192.168.1.101'"
and check to make sure that only the key(s) you wanted were added.
```

Figure A.9. Exemple de génération et transfert de la clé SSH vers le Worker 1.

Une fois le cluster est en cours d'exécution, nous pouvons aussi nous y connecter à travers Eclipse ou le Shell Spark (voir Figure A.10) en passant comme paramètre l'URL de cluster spark://192.168.100:7077.

```
./bin/spark-shell --master spark://192.168.100:7077
```

La Figure A.10 représente l'interface principale du Shell Spark en cours d'exécution :

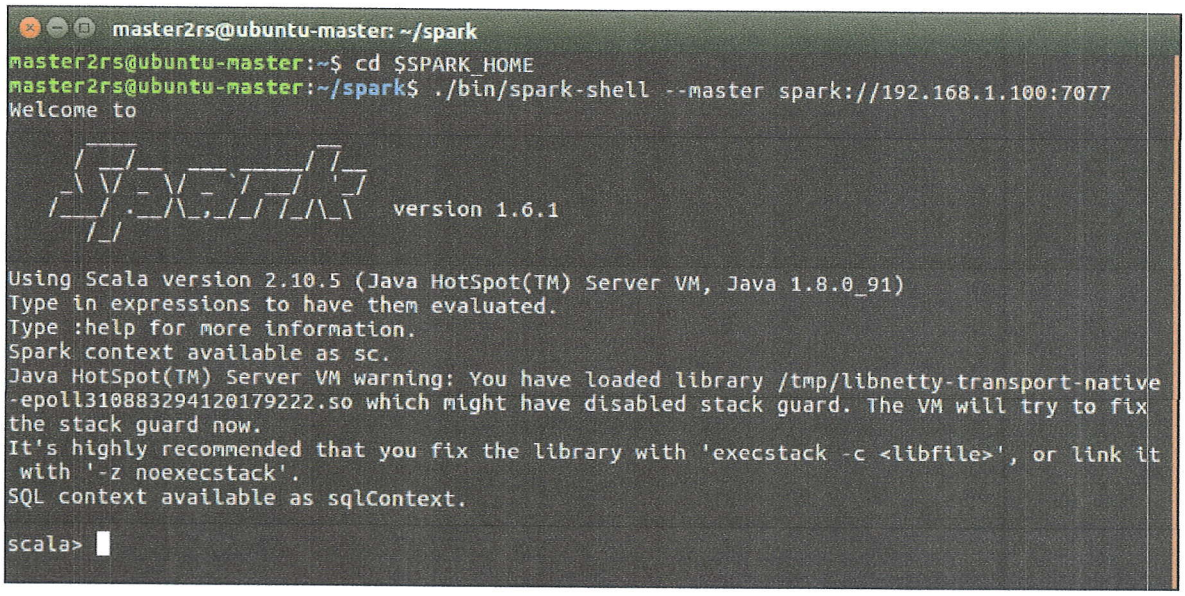


Figure A.10. Le Shell Spark en cours d'exécution sur le cluster.

A.1.11. La console Web de Spark

Quand le Shell Spark est en cours d'exécution, il est possible de consulter les statistiques de l'exécution en accédant à la console Web de Spark, via l'URL http://localhost:4040. La Figure A.11 montre la console Spark, avec ses onglets pour les étapes (stages), le stockage, l'environnement et les exécuteurs.

De plus, il est possible de vérifier l'état des Workers en accédant à la console Web de Cluster, à travers le lien http://localhost:8080 (voir Figure A.12). Si l'interface Web ne s'affiche pas, veuillez vérifier votre configuration de cluster (configuration SSH en particulier).

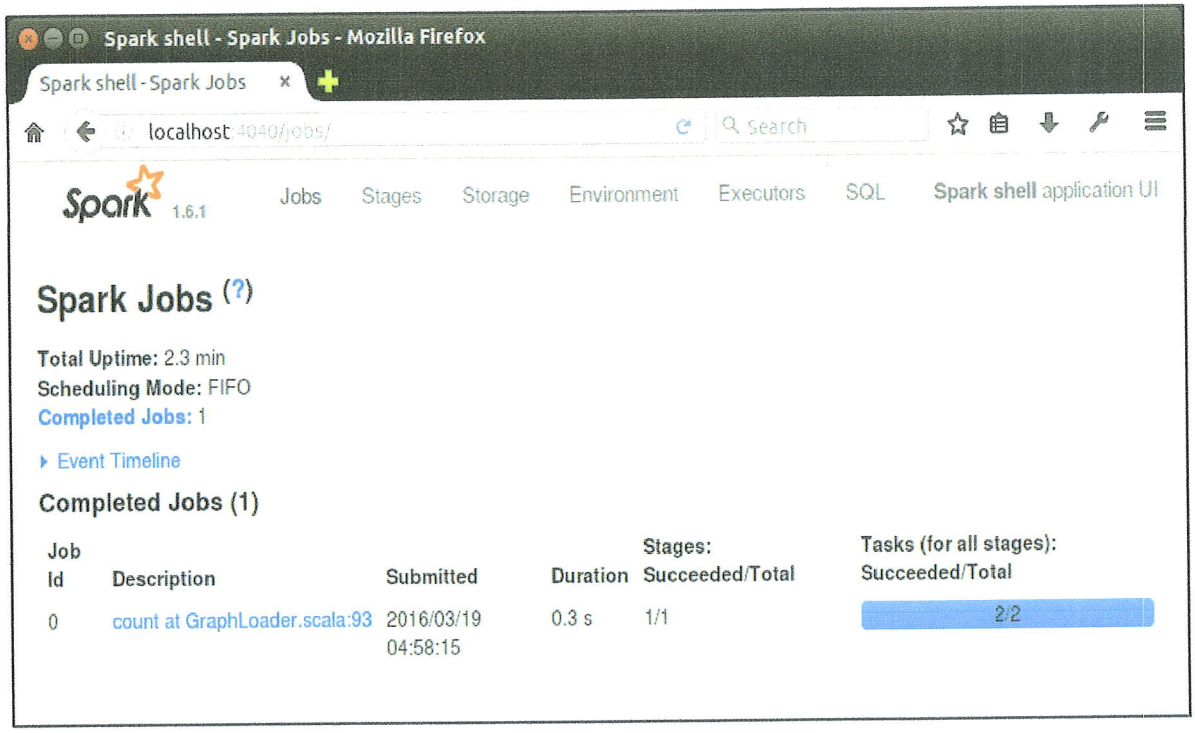


Figure A.11. La console Web de Spark Shell.

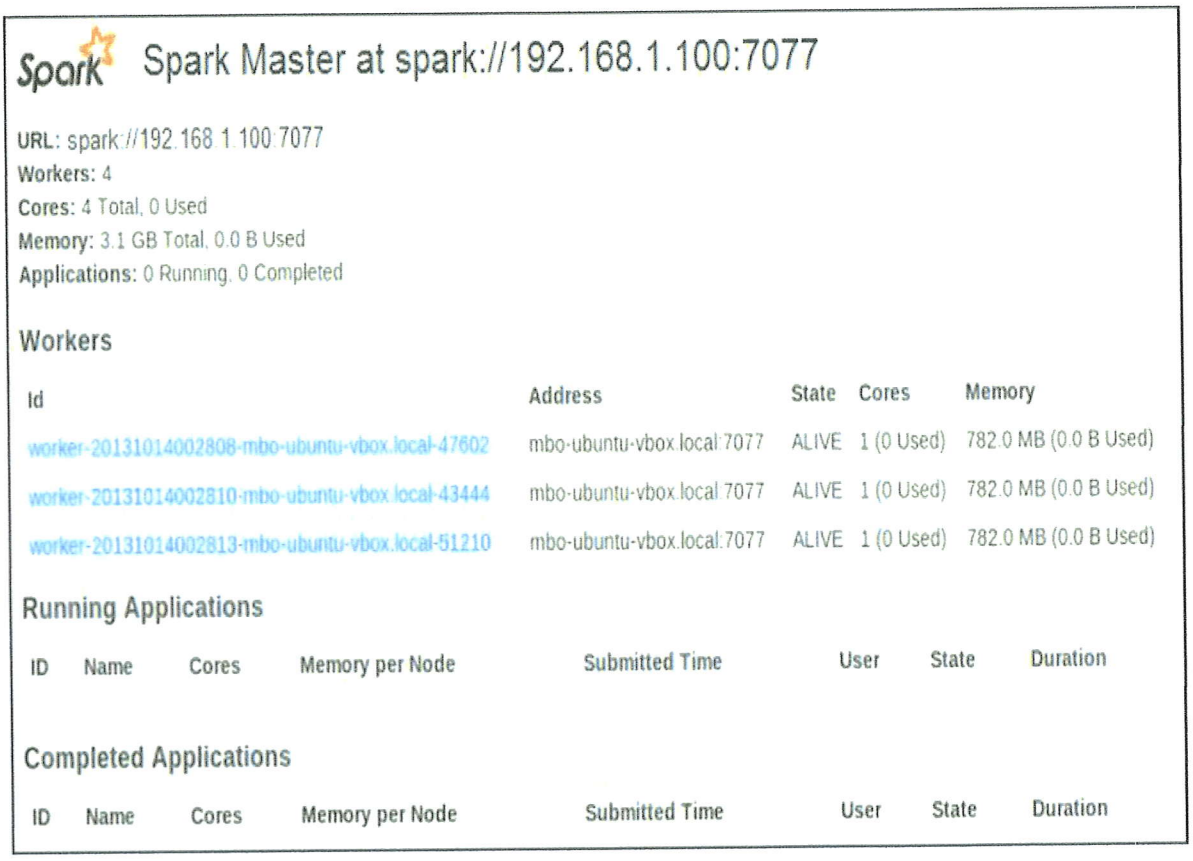


Figure A.12. La console Web de Spark Cluster.

A.2. Liste complète des actions et des transformations sur les RDDs

A.2.1 Liste des actions

Action	Signification
reduce(func)	Agréger les éléments du RDD en utilisant la fonction func (qui prend 2 arguments et retourne 1 résultat). La fonction devrait être associative et commutative afin de pouvoir être correctement calculée en parallèle.
collect()	Retourner tous les items du RDD comme un tableau au programme *driver*. A utiliser seulement si le RDD a un volume faible (par ex., après des opérations de type filter très sélectives).
count()	Retourner le nombre de items du le RDD.
take(n)	Retourner un tableau avec les n premiers items du RDD.
first()	Retourner le premier item du RDD (similaire à take(1)).
countByKey()	Pour RDD de type (clé, valeur), retourne l'ensemble de paires (clé, Int) avec le nombre de valeurs pour chaque clé.
takeSample(withReplacement, num, [seed])	Retourne un tableau avec un échantillon aléatoire de num items du RDD, avec ou sans remplacement, avec une possible spécification de seed pour le générateur aléatoire.
saveAsTextFile(path)	Ecrit les items du RDD dans un fichier texte dans un répertoire du système de fichiers local, HDFS ou autre fichier supporté par Hadoop. Spark appelle toString pour convertir chaque item en une ligne de texte dans le fichier.
saveAsSequenceFile(path)	(Java et Scala seulement !) Ecrit les items du RDD sous forme de Hadoop SequenceFile dans un répertoire du système de fichiers local, HDFS ou autre fichier supporté par Hadoop. Disponible pour RDD de paires (clé,valeur) qui implémentent l'interface Writable de Hadoop. En Scala, disponible aussi pour des types implicitement convertibles à Writable (Int, Double, String,etc.).

saveAsObjectFile(path)	(Java et Scala seulement !) Ecrit les éléments du RDD en un format simple utilisant la sérialisation Java. Cette sérialisation qui peut être lu ensuite avec la méthode SparkContext.objectFile().

A.2.2 Liste des transformations

Transformation	**Signification**
map(func)	Retourne un nouveau RDD obtenu en appliquant la fonction func à chaque item du RDD de départ.
filter(func)	Retourne un nouveau RDD obtenu en sélectionnant les items de la source pour lesquels la fonction func retourne « vrai ».
flatMap(func)	Similaire à map mais chaque item du RDD source peut être transformé en 0 ou plusieurs items ; retourner une séquence (Seq) plutôt qu'un seul item.
sample(withReplacement, fraction, seed)	Retourne un RDD contenant une fraction aléatoire fraction du RDD auquel la transformation s'applique, avec ou sans remplacement, avec une spécification de seed pour le générateur aléatoire.
union(otherDataset)	Retourne un RDD qui est l'union (ensembliste) des items du RDD source et du RDD argument (otherDataset).
intersection(otherDataset)	Retourne un RDD qui est l'intersection des items du RDD source et du RDD argument (otherDataset).
distinct([numTasks]))	Retourne un RDD qui est l'union des items du RDD source et du RDD argument (otherDataset).
groupByKey([numTasks])	Pour RDD de type (clé, valeur), retourne un RDD de paires (clé, Iterable<valeur>). Si le regroupement est réalisé en vue d'opérations d'agrégation (somme, moyenne), de meilleures performances

	peuvent être obtenues avecreduceByKey ou combineByKey.
reduceByKey(func, [numTasks])	Pour RDD de type (clé, valeur), retourne un RDD de paires (clé, valeur) où les valeurs pour chaque clé sont agrégées grâce à la fonction func de type (valeur,valeur) => valeur.
aggregateByKey(zeroValue) (seqOp,combOp,[numTasks])	Pour RDD de type (clé, valeur), retourne un RDD de paires (clé, U) où les valeurs pour chaque clé sont agrégées grâce aux fonctions de combinaison seqOp, combOp et à une valeur neutre zeroValue.
sortByKey([ascending], [numTasks])	Pour RDD de type (clé, valeur) où la clé implémente Ordered, retourne un RDD de paires (clé, valeur) trié par ordre ascendant ou descendant des clés (suivant la valeur de vérité de [ascending].
join(otherDataset, [numTasks])	Pour RDD de type (K, V) (ou (clé, valeur)) et otherDataset de type (K,W), retourne un RDD de paires (K, (V, W)) avec toutes les paires d'items pour chaque clé. Jointures externes avec leftOuterJoin, rightOuterJoin, and fullOuterJoin.
cogroup(otherDataset, [numTasks])	Pour RDD de type (K, V) (ou (clé, valeur)) et otherDataset de type (K,W), retourne un RDD de paires (K, (Iterable<V>, Iterable<W>)). Synonyme de groupWith.
cartesian(otherDataset)	Pour RDD de type T et otherDataset de type U, retourne un RDD de type (T, U) (toutes les paires d'items).
pipe(command, [envVars])	*Pipe* de chaque partition à travers une commande shell (par ex. script Perl ou bash). Les items de RDD sont écrits sur stdin du process et les sorties du process (stdout) sont retournées comme un RDD de strings.

subgraph	L'opérateur prend un prédicat des sommets et d'arêtes et renvoie le graphe ne contenant que les sommets qui satisfont le prédicat.
vertices	renvoie tous les sommets de graphes
collectNeighborIds	renvoie les voisins d'un sommet

A.3. Conversation d'un graphe orienté a un graphe non orienté

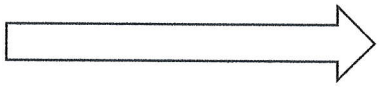
L'algorithme d'énumération de cliques maximales exige que le graphe d'entrée doive être un graphe non orienté. Pour cela, nous avons implémenté un algorithme supplémentaire (voir la Figure A.13) qui convertit un graphe orienté donnée à un graphe non orienté.

```
import org.apache.spark.{SparkConf, SparkContext, graphx}
import org.apache.log4j.{Logger, Level}
import org.apache.spark.graphx._
import java.io.{File, FileWriter, PrintWriter}

object OrientedGraph_To_NonOrientedGraph
{
  def main(args : Array[String])
  {
      Logger.getLogger("org").setLevel(Level.ERROR)
      val conf = new SparkConf().setAppName("Graph Conversion").setMaster("local[*]")
      val sc = new SparkContext(conf)

      val newFile = new File("/home/master2rs/spark/data/", "NonOrientedGraph.txt")
      val printWriter = new PrintWriter(new FileWriter(newFile))

      val graph = GraphLoader.edgeListFile(sc, "/home/master2rs/spark/data/OrientedGraph.txt")
      val bidirectedGraph = Graph(graph.vertices, graph.edges union graph.edges.reverse).cache()
      bidirectedGraph.edges.distinct().collect().sortBy { e => (e.srcId, e.dstId) }.foreach
      {
        v => printWriter.write(v.srcId.toString() + " " + v.dstId.toString() + "\n")
      }
      printWriter.close()
  }
}
```

Figure A.13. Code source de l'algorithme de conversion.

Exemple d'application de l'algorithme

Graphe orienté
1 2 1 3 2 3 3 4 2 5 5 7 5 6 5 8 6 7 6 8 7 8



Graphe non orienté
2 1 5 7 4 3 3 4 3 1 5 6 6 7 7 6 5 8 3 2 7 5 8 5 6 5 8 6 6 8 1 3 7 8 8 7 2 5 2 3 5 2 1 2

Résumé

De nos jours, l'utilisation de graphes a fait l'objet de nombreux travaux. Les graphes sont largement utilisés pour la modélisation de plusieurs types de réseaux, comme les réseaux sociaux, les réseaux pair-à-pair, le Web...etc. Particulièrement, l'analyse de graphes constitue un défi majeur dans le contexte de très grands volumes de données.

Avec la prédominance de l'informatique et des réseaux, et l'explosion quantitative des données numériques, le domaine d'analyse de graphes reçoit une attention significative de la communauté industrielle et académique. En raison de la croissance de la taille des graphes d'entrée, les algorithmes de graphe traditionnels ne peuvent être utilisés efficacement. Cela signifie qu'il y a un besoin réel pour des algorithmes qui peuvent être facilement parallélisés. De ce fait, plusieurs approches efficaces de traitement parallèle et de passage à l'échelle pour l'analyse de graphes ont été développées, surtout avec la haute disponibilité des environnements de Cloud Computing.

Durant ces dernières années, des nouveaux paradigmes de traitement parallèle des larges graphes ont été proposé, tels que Pregel de Google et son implémentation open source Giraph, Graphlab et GraphX. Ce dernier, est l'un des paradigmes de traitement parallèle de graphes qui ont prouvés leurs performances et leur efficacité, en présentant une nouvelle abstraction de traitement de Big Graph. L'objectif de ce travail est de répondre à un problème d'analyse de graphes en adaptant et implémentant un algorithme de graphe séquentiel dans un environnement parallèle Spark GraphX. Pour ce faire, nous avons choisi l'algorithme de Bron-Kerbosh, l'un des algorithmes les plus utilisés dans la littérature pour l'énumération des cliques maximales, et un algorithme glouton d'énumération de la clique maximum.

Les algorithmes choisis sont implémentés, testés et comparés sous GraphX. Les résultats obtenus montrent que les algorithmes parallèles proposés sont scalables et permettent l'énumération parallèle des cliques maximales et maximums de manière efficace.

Mots-clés : Cloud Computing, Big Data, Big Graph, Spark GraphX, Scala, Clique Maximale, Clique Maximum.

Abstract

Nowadays, the use of graphs has been the subject of numerous researches. Graphs are widely used for modeling various types of networks, such as social networks, peer-to-peer networks, the Web...etc. Particularly, graph analysis is a major challenge in the context of very large volumes of data.

With the prevalence of IT and networks, and the quantitative explosion of digital data, the graph analysis domain receives significant attention of both industrial and academic community. Due to the large size of the input graphs, traditional graph algorithms can no more be used efficiently. This relieves the need for easily parallelizable algorithms. Therefore, several effective approaches of parallel processing and scaling for graph analysis were developed, especially with the high availability of Cloud environments.

During the last years, new paradigms for parallel processing of large graphs have been proposed, such as Google's Pregel and its open source implementation Giraph, Graphlab and GraphX. The latter is one of the graph parallel processing paradigms that have proven their performance and effectiveness, by presenting a new processing abstraction of Big Graph. This work aims to answer a graph analysis problem by adapting and implementing a sequential graph algorithm in a Spark GraphX parallel environment. For this purpose, we chose the Bron-Kerbosh algorithm, one of the most popular maximal clique enumeration algorithms, and a greedy algorithm for maximum clique enumeration.

The chosen algorithms were implemented, tested and compared under GraphX. The obtained results show that the proposed parallel algorithms are both scalable and efficient in term of parallel enumeration of maximal and maximum cliques.

Keywords: Cloud Computing, Big Data, Big Graph, Spark GraphX, Scala, Maximal Clique, Maximum Clique.