

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

Université de - Jijel- Mohamed Sadik BENYAHIA



*Faculté des Sciences Exactes et Informatique
Département d'Informatique*

*Mémoire
de fin d'études pour l'obtention du diplôme
Master de Recherche en Informatique*

Option : Réseaux et Sécurité

Thème

Les Graphes sous Hadoop/Giraph: Algorithmes et implémentations

Réalisé par :

- Loudini Meryem
- Djemaïoune Massika

Encadré par :

- Melle : Brighen Assia



Promotion : 2016

Remerciements

Nous tenons tout d'abord à remercier Dieu le tout puissant qui nous a donné la force et la patience d'accomplir ce modeste travail.

Nous remercions notre encadreur : M^{lle} Brighen Assia pour ces précieux conseils et son aide durant la période de notre travail.

Nous remercions également les membres de jury qui ont accepté d'évaluer ce travail.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Merci à tous et à toutes.

*01
02*

Dédicace

L'aide de DIEU tout puissant, qui trace le chemin de ma vie, j'ai pu arriver à réaliser ce modeste travail que je dédie :

A ma plus belle étoile qui puisse exister dans l'univers, ma très chère mère celle à qui je souhaite une longue vie et bonne santé.

A mon père qui n'a pas cessé de m'encourager, qui je souhaite une longue vie.

A mon cher mari pour le soutien et le respect qu'il me porte pour son aide, son encouragement et sa disponibilité.

A tous mes frères, sœurs et toute ma famille

A mon binôme et sa famille

A toutes mes amies et collègues

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

MERYEM

Dédicace

L'aide de DIEU tout puissant, qui trace le chemin de ma vie, j'ai pu arriver à réaliser ce modeste travail que je dédie :

A ma plus belle étoile qui puisse exister dans l'univers, ma très chère mère celle à qui je souhaite une longue vie et bonne santé.

A mon père qui n'a pas cessé de m'encourager, qui je souhaite une longue vie.

A tous mes frères, sœurs et toute ma famille

A mon binôme et sa famille

A toutes mes amies et collègues

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

MASSIKA

Table des matières

1.1 Table des matières.....	I
1.2 Listes des figures.....	V
1.3 Listes des tableaux.....	VIII
1.4 Listes des abréviations.....	IX
Introduction Générale.....	I

Chapitre 1 : Notions introductifs et Concepts de base

1.1 Introduction.....	3
1.2 Cloud Computing.....	3
1.3 Concepts liés au Cloud Computing.....	4
1.3.1 Nœud.....	4
1.3.2 Cluster.....	4
1.3.3 Data Center.....	4
1.3.4 Virtualisation.....	5
1.4 Virtualisation et Cloud Computing.....	5
1.5 Les services de Cloud Computing.....	5
1.5.1 SaaS.....	6
1.5.2 PaaS.....	6
1.5.3 IaaS.....	6
1.6 Mode de déploiement.....	6
1.6.1 Cloud privé.....	7
1.6.2 Cloud public.....	7
1.6.3 Cloud communautaire.....	7
1.6.4 Cloud hybride.....	7
1.7 Les acteurs de Cloud.....	7
1.7.1 Amazon.....	7
1.7.2 Google.....	7

1.7.3 Salesforce	8
1.8 Big Data.....	8
1.8.1 Définition	8
1.8.2 Origine des données Big Data.....	8
1.8.3 Caractéristiques des Big Data	9
1.8.4 Les défis spécifiques du Big Data	10
1.8.5 Les solutions de Big Data.....	10
1.8.5.1 NoSQL	11
1.8.5.2 MapReduce	11
1.8.5.3 Hadoop	15
1.9 Big Data et Cloud Computing.....	17
1.10 Les graphes.....	17
1.10.1 Définition	17
1.10.2 Représentation des graphes.....	18
1.10.3 Domaine d'application des graphes	19
1.11 Graph et Big Data.....	19
1.12 Frameworks de traitement de big graphe	19
1.13 Conclusion.....	20
Chapitre 2 : Giraph et les algorithmes de graphe	
2.1 Introduction.....	21
2.2 Pregel.....	21
2.3 Apache Giraph.....	23
2.3.1 Définition	23
2.3.2 Architecture	23
2.4 Modèle de données.....	25
2.5 Modèle de traitement.....	26
2.6 Les étapes d'exécution d'un programme Giraph	27

2.7 Tolérance aux pannes	28
2.8 Compréhension des applications en Giraph	28
2.8.1 Exemple de SSSP (Single Source ShortestPath) dans un graphe orienté.....	30
2.8.2 PageRank.....	31
2.8.3 Composante connexe.....	33
2.9 Avantages et inconvénients de Giraph	36
2.9.1 Avantages	36
2.9.2 Inconvénients	36
2.10 Conclusion.....	36

Chapitre3 : Proposition d'un algorithme de graphe sous Giraph

3.1 Introduction	37
3.2 Le problème d'énumération des cliques.....	37
3.2.1 Définition de clique maximale	37
3.2.2 Définition de clique maximum.....	38
3.2.3 Domaines d'application des cliques	38
3.2.4 Algorithmes d'énumération des cliques	39
3.2.5 Proposition des nouveaux algorithmes d'énumération des cliques sous Giraph	43
3.3 Algorithme glouton pour le problème de clique maximum	48
3.3.1 Algorithme Glouton d'énumération de cliques maximum sous Giraph	49
3.4 Conclusion	52

Chapitre 4: Mise en œuvre et évaluation de l'algorithme

4.1 Introduction	53
4.2 Environnement du travail	53
4.3 Benchmarks de données choisis.....	54
4.4 Implémentation et évaluation expérimentale	57
4.4.1 Format d'entrée/sortie	57
4.4.2 Expérimentation et analyse des résultats	59
4.5 Conclusion	76
Conclusion Générale	77
Bibliographie	78

Annexe A	82
Annexe B	88
Annexe C	94

Listes des figures

Figure 1.1: Les couches du Cloud Computing.....	6
Figure 1.2: Exemple d'un programme MapReduce.....	12
Figure 1.3: Flux d'un opération MapReduce.....	14
Figure 1.4: Représentation des graphes sous matrice d'adjacence.....	18
Figure 1.5: Représentation d'un graphe par liste d'adjacence.....	18
Figure 2.1: Structure d'un nœud dans le modèle Pregel.....	22
Figure 2.2: Fonctionnement de BSP.....	22
Figure 2.3: Giraph intégré avec Hadoop.....	23
Figure 2.4: Architecture de Giraph.....	24
Figure 2.5: Le flux de données de Giraph dans une implémentation MapReduce classique... 25	25
Figure 2.6: Plan décrit la transition entre les étas actifs et inactifs.....	26
Figure 2.7: Exemple de calcul de la valeur maximale de sommet dans BSP.....	27
Figure 2.8: Exemple de valeur maximale (les lignes pointillées sont des messages, les sommets ombragés ont voté pour stoper.....	28
Figure 2.9: Format d'entré Json sous Giraph.....	29
Figure 2.10: Exemple de SSSP sous Giraph.....	30
Figure 2.11: La methode Compute () de ShortestPath sous Giraph.....	31
Figure 2.12: Exemple de PageRank sous Giraph.....	32
Figure 2.13: La methode Compute () de PageRank	32
Figure 2.14: La methode Compute () de Composante connexe.....	34
Figure 2.15: Exemple d'algorithme Composante connexe sous Giraph.....	35
Figure 3.1: Un graphe avec 5 cliques maximales.....	38
Figure 3.2: Un graphe avec une clique maximum {4, 5,6,7}.....	38
Figure 3.3: Algorithme Bron-Kerbosh sans Pivot [16].....	40
Figure 3.4: Algorithme Bron-Kerbosh avec Pivot [16].....	41
Figure 3.5: Exemple de graphe avec 4 cliques maximales.....	41
Figure 3.6: Pseudo code de l'algorithme EPCM.....	43
Figure 3.7: Exemple avec 5 cliques maximales.....	44
Figure 3.8: Pseudo code de BKP-AP.....	46
Figure 3.9: Pseudo code d'algorithme Glouton.....	48
Figure 3.10: Déroulement de l'algorithme Glouton sur le graphe de la figure 3.2.....	49

Figure 3.11: Pseudo code de l'algorithme HPECM.....	50
Figure 3.12: Pseudo code de la fonction qui calcule la clique maximum.....	51
Figure 4.1: Exemple de Format d'entré des données de graphe de figure 3.7.....	57
Figure 4.2: Résultats d'exécution d'algorithme HPECM sous Giraph.....	58
Figure 4.3: Graphe avec 5 cliques maximales.....	58
Figure 4.4: Format d'entrée de données de graphe de figure 4.3.....	59
Figure 4.5: Résultats d'exécution d'algorithme Op-EPCM sous Giraph sur le graphe de figure 4.3.....	59
Figure 4.6: Résultats d'exécution d'algorithme HPECM sous Giraph en termes de temps de calcul.....	60
Figure 4.7: Résultats d'exécution d'algorithme HPECM sous Giraph en termes de messages.....	60
Figure 4.8: Résultats d'exécution d'algorithme HPECM sous Giraph en termes de Supersteps.....	60
Figure 4.9: Test de scalabilité des algorithmes d'énumération des cliques maximales en termes de tems CPU sous Giraph.....	62
Figure 4.10: Test de scalabilité des algorithmes d'énumération des cliques maximales en termes de message sous Giraph.....	63
Figure 4.11: Test de convergence des algorithmes d'énumération des cliques maximales en termes de supersteps sous Giraph.....	64
Figure 4.12: Test de scalabilité des algorithmes d'énumération des cliques maximum sous Giraph en termes de temps CPU sous Giraph.....	65
Figure 4.13: Test de scalabilité des algorithmes d'énumération des cliques maximum sous Giraph en termes de nombre des messages sous Giraph.....	66
Figure 4.14: Test de convergence des algorithmes d'énumération des cliques maximum sous Giraph en termes de nombre des supersteps sous Giraph.....	67
Figure 4.15: Résultats de Comparaison entre les algorithmes de cliques maximales en termes de temps CPU sous Giraph.....	68
Figure 4.16: Résultats de Comparaison entre les algorithmes de cliques maximales en termes de nombre de messages sous Giraph.....	69
Figure 4.17: Résultats de comparaison entre les algorithmes d'énumération des cliques maximales en termes de nombre de Supersteps sous Giraph.....	70
Figure 4.18: Résultats de comparaison des algorithmes de cliques maximums en termes de temps CPU sous Giraph.....	72

Figure 4.19: Résultats de comparaison des algorithmes de clique maximum en termes de nombre de message sous Giraph.	73
Figure 4.20: Résultats de comparaison des algorithmes de clique maximum en termes de nombre de superstep sous Giraph.	74
Figure 4.21: Comparaison entre les algorithmes d'énumération des cliques maximums dans un environnement multi-node	75
Figure 4.22: Comparaison entre les algorithmes d'énumération des cliques maximales dans un environnement multi-node	75

Liste des tableaux

Tableau 4.1 : Tableau descriptif de l'environnement.....	53
Tableau 4.2 : Tableau descriptif des machines virtuelles et physique	54
Tableau 4.3 : Benchmarks de données utilisés (non orienté).....	54
Tableau 4.4 : Résultats d'exécution de HPECM sur quelques graphes de DIMACS	55
Tableau 4.5 : Les données de Sp-graphe 1.....	55
Tableau 4.6 : Les données de Sp-graphe 2.....	56
Tableau 4.7 : Les données de Sp-graphe 3.....	56
Tableau 4.8 : Les données de Sp-graphe 4.....	57

Liste des Abréviations

- Amazon EC2:** Amazon Elastic Compute Cloud
- API:** Interface de Programmation Applicative
- AWS:** Amazon Web Services
- BSP:** Bulk Synchronous Parallèle
- BKP-AP:** Bron Kerbosh Parallèle Avec Pivot
- BKP-SP:** Bron Kerbosh Parallèle Sans Pivot
- CRM:** Customer Relationship Management
- E/S:** Entré/Sortie
- EPCM :** Énumération Parallèle de Clique Maximale
- GFS:** Google File System
- GRH:** Gestion des Ressources Humaines
- Hadoop:** High Availability Distributed Object Oriented Platform
- HDFS:** Hadoop Distributed File System
- HPECM :** Heuristique Parallèle d'Énumération des Clique Maximum
- IAAS:** Infrastructure as a Service
- IBM:** *International Business Machines*
- IPP:** Interactions Protéine-Protéine
- MCE:** Maximal Clique Exploration
- NIST:** National Institute of Standards and Technology
- NOSQL:** Not Only SQL
- Op-EPCM :** Optimisation d'Enumération Parallèle de Clique Maximale
- PaaS:** Platform as a Service
- PCM:** Problème de Clique Maximum

SaaS: Software as a Service

SQL: Structured Query Language

SSSP: Single Source Shortest Path

Introduction Générale

Au fur et à mesure que les systèmes informatiques évoluent, la demande en quantité d'espace de stockage, de convivialité et de simplicité dans le travail augmente. Il y a quelques années, les espaces de stockage réduits, et les systèmes complexes étaient le quotidien des employés d'entreprises.

Les entreprises modernes traitent de grandes quantités d'informations aussi nombreuses que variées. Ainsi, elles ont besoin de grandes capacités de stockage et une puissance de calcul élevée. Les ressources matérielles et logicielles nécessaires n'étant pas à la portée de toutes les entreprises, le Cloud Computing est une solution pour résoudre ce problème.

Le Cloud Computing ou l'informatique en nuage consiste à pourvoir des ressources informatiques à distance et à la demande, qu'il s'agisse d'infrastructure, de plates-formes ou de logiciels d'application. Le Cloud Computing s'appuie sur la virtualisation pour mutualiser les ressources et utiliser les applications, les environnements de développement et de l'infrastructure matérielle en tant que services. Depuis l'apparitions de cette technologies, et les bénéfices qu'elle apporte, des nombreuses applications migrent a grand pas vers le Cloud, parmi ces applications on trouve le Big Data.

Big Data correspond à la limite où il devient impossible d'utiliser un petit ensemble d'outils génériques, prêts à l'emploi, pour stocker et analyser un volume important de données produites (structurées, semi-structurées ou non structurées). Il existe des différentes technologies pour la gestion et l'analyse des Big Data. Parmi ces technologies, dont les plus célèbres sont MapReduce de Google et son implémentation open source Hadoop qui permet le traitement et le stockage de large quantité de données.

Généralement, les Big Data sont modélisées par les graphes. Ces graphes peuvent être d'une grande taille (Big graph), donc il devient difficile de les traités efficacement sur une seule machine. Cela signifie qu'il ya un besoin pour les algorithmes qui peuvent être facilement parallélisés. Par conséquent, pour simplifier la programmation des algorithmes de graphe dans un environnement distribué un certain nombre des outils open-source ont été développé. Actuellement, Giraph qui s'appui sur le modèle BSP, popularisé par le projet Pregel de Google, est le plus utilisé. Il est conçu pour exécuter les algorithmes de graphes itératifs à travers des clusters de machines.

Problématique et objectif du travail

Giraph est un outil puissant de traitement parallèle de graphes. Il est libre, open source et délivré avec une bibliothèque qui contient un ensemble d'algorithmes de graphes prédéfinis bien testés et prêts à s'exécuter sur des larges graphes. La bibliothèque Giraph ne couvre pas tous les algorithmes de graphes qui se trouvent dans la littérature. Dans ce travail nous allons proposer et implémenter des nouveaux algorithmes de graphes sous Giraph. Pour ce faire, nous allons examiner l'énumération de cliques.

Organisation du mémoire

Nous avons organisé notre mémoire en quatre Chapitre :

- ✓ Le premier chapitre est consacré au concept Cloud Computing. A travers ce chapitre, nous essayons de bien clarifier le terme Cloud Computing et de présenter les différents concepts liés à ce nouveau modèle, ses principales caractéristiques, ses acteurs ainsi que les outils de traitement parallèle des graphes sur le Cloud.
- ✓ Le deuxième chapitre est destiné à la présentation du framework Giraph. Au cours de ce chapitre nous allons parler sur le framework Giraph, son architecture, son modèle de données et modèle de traitement, ainsi nous allons présenter, comme exemple, trois algorithmes de graphe sous Giraph.
- ✓ Dans le troisième chapitre nous allons définir le problème d'énumération des cliques, ainsi nous allons proposer des algorithmes de graphes parallèles sous Giraph pour l'énumération des cliques maximales (maximum).
- ✓ Le quatrième chapitre nous a permis de tester la scalabilité et la convergence de nos algorithmes proposés et de faire une comparaison pratique entre eux en utilisant des benchmarks de données.
- ✓ En fin, notre mémoire s'achève par une conclusion générale où nous dégagons quelques perspectives de recherche.

Chapitre 1 :
Notions introductifs et concepts
de base

1.1 Introduction

La technologie de l'Internet se développe de manière exponentielle depuis sa création. Actuellement, une nouvelle tendance dans le monde des technologies de l'information et de la communication a fait son apparition, il s'agit du Cloud Computing, ou « informatique dans les nuages ». Ce nouveau paradigme consiste à proposer les services informatiques sous forme de services à la demande, accessibles de n'importe où, n'importe quand et par n'importe qui.

Dans ce chapitre nous allons présenter les différents concepts liés à ce nouveau paradigme de consommation de l'informatique, ensuite nous intéressons de l'évolution massive des données «Big Data» avec une solution de stockage «NoSQL, Hadoop », aussi on va présenter des notions liées au big graph et les différents outils de traitement parallèle de large graph sur le cloud.

1.2 Cloud Computing

Même si les experts ne sont pas d'accords sur la définition exacte du Cloud Computing, la plupart s'accordent à dire qu'elle inclue la notion de services disponibles à la demande et extensibles à volonté. En contradiction avec les systèmes actuels, les services sont virtuels et illimités et les détails des infrastructures physiques sur lesquels les applications reposent ne sont plus du ressort de l'utilisateur. Dans cette section, nous allons citer comme exemple quelques définitions qui ont circulés :

- **Selon NIST (National Institute of Standards and Technology):** Le Cloud Computing est un modèle d'accès à des ressources informatiques partagées et configurables (par exemple : réseaux, serveurs, stockage, applications et services) depuis un accès réseau de manière simple, à partir de n'importe quel type d'appareil et de n'importe quel endroit. Ces ressources sont disponibles rapidement et opérationnelles avec un minimum d'efforts et d'interactions avec le fournisseur de services [32].
- **Selon Buyya et al:** Un nuage est un type de système parallèle et distribué composé d'un ensemble d'ordinateurs interconnectés et virtualisés et qui sont provisionnés dynamiquement et présentés comme une ou plusieurs ressources informatiques unifiées, basées sur des accords de service établis par la négociation entre le prestataire et les consommateurs [41].

- **Selon l'université de Berkely:** Le Cloud Computing désigne à la fois les applications qui sont livrées comme des services sur Internet, ainsi que le matériel et les logiciels dans les Datacenter qui fournissent ces services. Les services eux-mêmes ont été longtemps appelés Software as a service (SaaS), donc nous utilisons ce terme. Le matériel, Datacenter et le logiciel sont ce que nous appellerons un nuage [33].

Selon les définitions précédentes le Cloud Computing, littéralement informatique dans les nuages, désigne l'utilisation de serveurs distants(en général accessible par internet) pour traiter ou stocker l'information. L'accès se fait le plus souvent à l'aide d'un navigateur web. Enregistrer des fichiers via internet sur un serveur. Le logiciel lui-même peut être déporté lui aussi sur l'ordinateur distant.

1.3 Concepts liés aux Cloud Computing

1.3.1 Nœud :

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

1.3.2 Cluster :

Le Cluster est un ensemble d'ordinateurs interconnectés entre eux et travaillent en collaboration pour réaliser une fonction unique. Le Cloud Computing permet de construire un nuage de clusters ainsi de gérer l'interface entre le nœud et l'utilisateur, ainsi que la connexion entre un ensemble de machines sur un réseau défini. Les utilisateurs peuvent ensuite déployer des machines virtuelles dans ce nuage, ce qui leur permet d'utiliser un certain nombre de ressources tels que l'espace disque, de la mémoire vive, ou encore du CPU (processeur).

1.3.3 Data Center

Un centre de données (Datacenter) est un endroit physique où sont rassemblées de nombreuses machines (bien souvent des serveurs) contenant des données informatiques. Les fournisseurs de services Cloud utilisent les centres de données pour abriter les services de Cloud et des ressources basées sur le Cloud. Les vendeurs aussi souvent propriétaires de plusieurs centres de données dans plusieurs endroits géographiques pour préserver la disponibilité des données pendant les pannes et autres défaillances des centres de données.

1.3.4 Virtualisation

La virtualisation est un mécanisme informatique qui consiste à faire fonctionner plusieurs systèmes, serveurs ou applications, sur un même serveur physique. La virtualisation est un composant technique clé dans le Cloud Computing. La virtualisation repose sur le mécanisme suivant :

- Un système d'exploitation principal (appelé « système hôte ») est installé sur un serveur physique unique. Ce système sert d'accueil à d'autres systèmes d'exploitation.
- Un logiciel de virtualisation (appelé « hyperviseur¹ ») est installé sur le système d'exploitation principal. Il permet la création d'environnements clos et indépendants sur lesquels seront installés d'autres systèmes d'exploitation (« systèmes invités »). Ces environnements sont des « machines virtuelles ».
- Un système invité est installé dans une machine virtuelle qui fonctionne indépendamment des autres systèmes invités dans d'autres machines virtuelles. Chaque machine virtuelle dispose d'un accès aux ressources du serveur physique (mémoire, espace disque...) [20].

1.4 La virtualisation et Cloud Computing

Le Cloud Computing et la virtualisation sont étroitement liés, on pouvait voir chaque concept comme un sous-ensemble de l'autre. La virtualisation est tout simplement le processus dans lequel le logiciel est utilisé pour simuler le matériel. Le Cloud Computing ajoute une couche d'abstraction, en utilisant la virtualisation comme son infrastructure physique tout en permettant un accès largement diffus. La principale différence entre ces deux concepts est que, dans la virtualisation, on s'intéresse à la recherche dans la gestion interne du matériel, tandis que les services de Cloud Computing sont déjà pris en charge par le fournisseur du réseau étendu [34].

1.5 Les services de Cloud Computing

Le Cloud Computing peut être subdivisé en trois couches (voir la figure 1.1) :

¹Aussi appelé VMM (Virtual Machine Monitor) est une plateforme de virtualisation qui permet à plusieurs systèmes d'exploitation de travailler sur une machine physique en même temps.

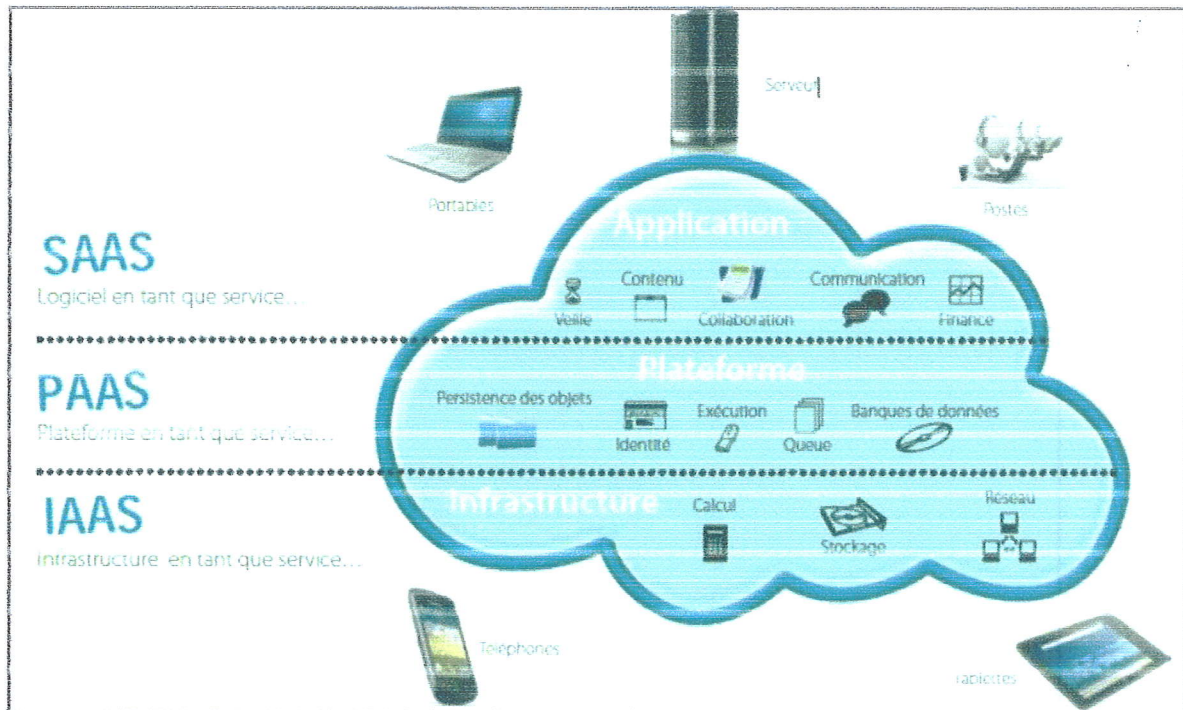


Figure 1.1 : Les couches du Cloud Computing [17].

1.5.1 SaaS (Software as a Service):

SaaS est accessible à toutes les entreprises, il est facturé au nombre d'utilisateurs. L'entreprise loue les applications du fournisseur de services et n'a plus besoin d'acheter un logiciel. Ces applications sont accessibles via différentes interfaces, navigateurs Web, clients légers... [17]

1.5.2 PaaS (Platform as a Service):

PaaS, facturée à la consommation, est un environnement qui permet à l'entreprise de déployer ses propres applications en dehors de sa salle informatique. L'entreprise loue un environnement middleware à l'infrastructure masquée [17].

1.5.3 IAAS (Infrastructure as a Service) :

IaaS c'est la mise à disposition par Internet de machines virtuelles aux ressources facilement modifiables (à la hausse ou baisse) et hautement disponible. L'entreprise loue ainsi des capacités de traitement, de stockage et autres ressources qu'elle peut structurer et gérer de façon autonome côté logiciel dès le système d'exploitation [17].

1.6 Mode de déploiement

On peut distinguer quatre modèles de déploiement : Cloud Privé, Cloud public, Cloud communautaire et Cloud hybrid.

- **Cloud privé (ou interne)** : Réseau informatique propriétaire ou un centre de données qui fournit des services hébergés pour un nombre limité d'utilisateurs.
- **Cloud public (ou externe)** : Prestataire de services qui propose des services de stockage et d'applications Web pour le grand public. Ces services peuvent être gratuits ou payants, comme exemple on trouve: Amazon Elastic Compute Cloud (EC2), Sun Cloud, IBM's Blue Cloud, Google AppEngine et Windows Azure Services Platform.
- **Cloud communautaire**: L'infrastructure Cloud est partagée par plusieurs organisations pour les besoins d'une communauté qui souhaite mettre en commun des moyens (sécurité, conformité, etc.). Elle peut être gérée par les organisations ou par une tierce partie et peut être placée dans les locaux ou à l'extérieur.
- **Cloud hybride (interne et externe)** : Un environnement composé de multiples prestataires internes et externes. Un exemple, IBM avait conclu un partenariat avec Juniper Networks. Cette association a permis à Big Blue de déployer son offre de cloud hybride. Ainsi les entreprises qui utilisent ce service peuvent faire basculer, par un simple glissé-déposé, des applications hébergées dans un nuage privé interne vers un nuage public sécurisé.

1.7 Les acteurs de Cloud

Les fournisseurs de services de Cloud Computing sont des hébergeurs qui mettent à disposition des infrastructures physiques proposant une plate-forme de Cloud. Dans cette section nous citons quelques principaux acteurs :

1. Amazon (l'acteur venu d'ailleurs) :

« Amazon Web Services » (AWS) met à disposition un cloud public depuis 2006. Au départ le but d'Amazon était de rentabiliser ses énormes infrastructures requises pour assumer les montées en charge pendant la période de Noël sur leur boutique en ligne.

Aujourd'hui Amazon propose de nombreux services en ligne, à commencer par l'IaaS probablement le plus connu : Elastic Compute Cloud (EC2). Ce dernier présente un environnement informatique vraiment virtuel, permet d'utiliser des interfaces de service Web pour lancer des instances avec une variété de systèmes d'exploitation, de les charger avec un environnement d'applications personnalisées, de gérer les autorisations d'accès au réseau, et d'exécuter une image en utilisant autant ou aussi peu de systèmes désirés.

2. Google : Google AppEngine lancé en 2008, disponible uniquement en Cloud Public, sous la forme d'une offre gratuite. Sa grande force réside dans le fait qu'il soit justement gratuit, et

propose des applications de qualité, malgré le très grand nombre de requête effectués sur les serveurs.

3. Salesforce : Salesforce (Salesforce.com) est le premier hébergeur de Cloud en 1999, la principale application proposée par Salesforce est la CRM (Customer Relationship Management). Il propose une plate- forme pour le développement et les outils proposés sont tournés vers le travail collaboratif, la gestion des ventes et le déploiement des applications de gestion à la demande comme la gestion des ressources humains (GRH), etc.

1.8 Big data

1.8.1 Définition

Big data est un terme anglais, signifiant littéralement les « grosses données », parfois appelées données massives. Ce terme a été évoqué la première fois par le cabinet d'études Gartner² en 2008[43]. Big Data désignent énormes volumes de données structurées et non structurées, difficilement gérables avec des solutions classiques de stockage et de traitement. Ces données proviennent de sources diverses et sont (pour la plupart) produites en temps réel [35]. La rapidité de leurs échanges ainsi que leur accessibilité et disponibilité sont également des points structurant du Big Data [4].

1.8.2 Origine des données Big Data

Les données traitées par le Big Data proviennent notamment [44]:

- du **Web**: journaux d'accès, réseaux sociaux, e-commerce, indexation, stockage de documents, de photos, de vidéos, linked data, etc. (ex: Google traitait 24 petaoctets de données par jour avec MapReduce en 2009).
- plus généralement, de l'**internet** et des **objets communicants**: réseaux de capteurs, journaux des appels en téléphonie.
- des **sciences**: génomique, astronomie, physique subatomique (ex: le CERN annonce produire 15 petaoctets de données par an avec le LHC), climatologie (ex: le centre de recherche allemand sur le climat gère une base de données de 60 petaoctets), etc.
- données **commerciales** (ex: historique des transactions dans une chaîne d'hypermarchés).

² est une entreprise américaine de conseil et de recherche dans le domaine des techniques avancées dont le siège social est situé à Stamford, Fondée en 1979.

- données **personnelles** (ex: dossiers médicaux).
- données **publiques** (open data).

1.8.3 Caractéristiques des Big Data

Le Big Data est une force de rupture, qui offre aux services informatiques des opportunités aussi bien que des défis à relever. Le concept de Big Data est défini par la règle “4V” qui permette de distinguer les principaux problèmes informatiques à résoudre :

- **Volume** : les données dépassent les limites de la scalabilité verticale des outils classiques, nécessitant des solutions de stockage distribués et des outils de traitement parallèles [44] (se situant actuellement entre quelques dizaines de téraoctets et plusieurs péta-octets). À titre d'exemple, Twitter générait en janvier 2013, 7 téraoctets de données chaque jour et Facebook 10 téraoctets. Selon IBM [29] : «Chaque jour, nous générons 2,5 trillions d’octets de données. A tel point que 90% des données dans le monde ont été créés au cours des deux dernières années seulement. Ces données proviennent de partout: de capteurs utilisés pour collecter les informations climatiques, de messages sur les sites de médias sociaux, d’images numériques et de vidéos publiées en ligne, d’enregistrements transactionnels d’achats en ligne et de signaux téléphones mobiles, pour ne citer que quelques sources».
- **Variété** : Les procédés traditionnels de gestion des données ne peuvent faire face à l’hétérogénéité du Big Data, tels que des traces d’accès et des historiques de recherche Web. Les données proviennent de sources internes (88% provient de transactions, 57% d’emails, etc) ou externes (34% de photos et vidéos, 43% depuis les réseaux sociaux, etc.) [21] dont les supports et les formats ne sont pas toujours contrôlés par l’entreprise. Ces données structurées ou non structurées doivent être exploitées dans leur format natif.
- **Vitesse** : La vitesse non seulement en termes de création de données, mais aussi sur le plan de leur traitement, de leur analyse et de leur restitution à l’utilisateur en respectant les exigences des applications en temps réel [13]. Par exemple plusieurs fois par jour, une banque d’affaire doit pouvoir envoyer les centaines de milliers de rapports financiers à ses clients en moins d’une heure. En revanche, si un phénomène évolue rapidement dans le temps, un laboratoire de génie génétique ou de microbiologie doit disposer des résultats d’une analyse de millions d’échantillons en moins de 2 minutes ou les suivre en temps réel [21].

- **Vélocité** : La vélocité représente à la fois la fréquence à laquelle les données sont générées, capturées et partagées et mises à jour. Plus récemment, des nouvelles dimensions pertinentes ont été ajoutées: la valeur, la visibilité et la datavisualisation.

Le modèle Big Data est aujourd'hui défini par 7 attributs:

- Les **Valeurs** que représentent ces données apportent de nouvelles connaissances.
- **Visibilité** : Il est nécessaire que les informations soient disponibles et visuellement présentables, c'est-à-dire accessibles rapidement afin de les surveiller, de les comprendre quel que soit le support utilisé.
- **Datavisualisation** : Concerne la représentation des données sous formes intelligentes pratiques et interactives. Si l'exploitation des données permet de leur donner un sens, ce sont les images, les diagrammes, les graphiques qui leur donne un sens global.

1.8.4 Les défis spécifiques du Big Data

Le Big Data représente certes une opportunité significative, mais pose aussi des défis spécifiques. Parmi lesquelles on peut citer les éléments suivants :

- **Les défis des compétences** : A chaque apparition d'une nouvelle technologie, les entreprises cherchent à trouver des compétences. Les plateformes Big Data comme Hadoop et NosQL sont des nouveaux frameworks avec peu de compétence dans ces domaines. Peu de développeurs sont formés à la programmation MapReduce et peu d'entreprises ont les ressources nécessaires à la formation.
- **Les défis de l'infrastructure** : Il faut mettre en place cette infrastructure ou aller la chercher sur le Cloud. Le défi ici sera à la fois la capacité de montée en charge et l'intégration avec les systèmes d'informations existants.
- **Les défis des sources de données** : La multiplicité des sources de données créent des défis techniques pour les Big Data, particulièrement pour l'intégration de données.
- **Les défis de la gouvernance des données** : Un des principaux défis en matière de gouvernance des données consiste à éviter les fuites et à protéger ces données.
- **Les défis économiques** : Une entreprise peut décider de traiter toutes ses données mais il faut aussi maîtriser ses dépenses [49].

1.8.5 Les solutions de Big Data

Les outils et les infrastructures traditionnels ne sont pas efficaces pour travailler avec des données plus importantes, plus variées et arrivant à grande vitesse. Les

Cloud Computing offrent une moyenne économique face aux défis de Big Data. En plus, des nouvelles technologies font leur apparition pour rendre l'analyse de Big Data évolutive et rentable. Une nouvelle approche utilise la puissance d'un réseau distribué de ressources informatiques, des plateformes de traitement distribuées et des bases de données non relationnelles, afin de redéfinir la façon dont les données sont gérées et analysées [24]. Hadoop et NoSQL sont les technologies phares de l'univers Big Data.

1.8.5.1 NoSQL

NoSQL désigne une vaste catégorie de systèmes de stockage de bases de données très différents de l'architecture classique des bases relationnelles [14].

Les limitations des systèmes de gestion de base de données de type relationnel ont obligé les acteurs du monde informatique à trouver de nouveaux moyens pour traiter des données dont le volume croissait de manière exponentielle tout en conservant des coûts de maintenance raisonnables. Le NoSQL est apparu comme une solution à ce problème. Apparu en 1998, ce terme acronyme de Not Only SQL désigne un nouveau paradigme dans lequel tout est orienté vers la performance et la simplicité. Le NoSQL abandonne donc le concept de modèle relationnel qui avait pourtant été considéré lors de sa création comme une évolution notoire dans le monde des systèmes de gestion de bases de données [1].

1.8.5.2 MapReduce

a) Définition :

MapReduce est un modèle de programmation popularisé par Google. Il est principalement utilisé pour la manipulation et le traitement d'un nombre important de données typiquement supérieures en taille à 1 téraoctet au sein d'un cluster de nœuds [42].

b) Caractéristiques :

- Permet de répartir la charge sur un grand nombre de serveurs (cluster).
- Fournit une tolérance aux fautes efficaces à laquelle il peut redémarrer les nœuds ayant rencontré une erreur ou affecter la tâche à un autre nœud.
- La librairie MapReduce existe dans plusieurs langages (C++, C#, Java, Python, ...).
- Utilise une architecture de type Maître-esclave où un nœud maître dirige tous les nœuds esclaves.

- La parallélisation est invisible à l'utilisateur afin de lui permettre de se concentrer sur le traitement des données.

c) Phases d'exécution d'un job MapReduce:

L'algorithme MapReduce s'exécute en 5 phases :

1. Lecture d'une grande quantité de données.
2. La phase **Map** (distribution du calcul).
3. La phase **Shuffle** (transfert des données de Map vers Reduce).
4. La phase de **Sort** (permet de trier les valeurs selon la clé après l'agrégation de L'ensemble des partitions et des valeurs pour une clé donnée).
5. La phase **Reduce** (réduction des résultats).

Dans le modèle de programmation MapReduce, le développeur implémente deux fonctions : La fonction **Map** et la fonction **Reduce**.

-Map [30] :

- Prend en entrée un ensemble de « **Clé, Valeurs** »
- Retourne une liste intermédiaire de « **Clé1, Valeur1** »

Map (clé1, valeur1) → List (clé2, valeur2)

-Reduce [30] :

- Prend en entrée une liste intermédiaire de « **Clé1, Valeur1** »
- Fournit en sortie un ensemble de « **Clé1, Valeur2** »

Reduce (key2, List (valeur2)) → valeur3

Exemple: Parmi les exemples les plus connus, on trouve le comptage de nombre d'occurrences de chaque mot dans une grande collection de documents. L'utilisateur écrirait le code semblable au pseudo-code suivant [36] :

MapFunc (String key, String value):

```
// Key: id du document;
// value: contenu du document
for each word w in value
EmitIntermediate (w, 1)
```

ReduceFunc (Stringkey,Iterator values)

```
// key: un mot;
// values: une liste de valeurs
result = 0
for each count v in values
result += v
EmitResult (result)
```

Figure1. 2: Exemple d'un programme MapReduce.

- Lorsque La fonction Map reçoit comme entrée un document dont la clé est l'identifiant de document et la valeur c'est ses éléments.
- La fonction Map parcourt le document qu'elle reçoit comme paramètres et renvoi chaque mot trouvé et le nombre d'occurrence associé (juste 1 dans cet exemple).
- Les résultats intermédiaires sont la liste des couples clé-valeur ou les clés sont les différents mots trouvés dans les documents d'entrée et les valeurs sont la liste des valeurs envoyées pour chaque mot par Map. Les résultats intermédiaires des différentes clés sont groupés par la bibliothèque MapReduce avant qu'ils soient envoyés à la fonction Reduce.
- La fonction Reduce prend une clé (un mot) et la liste des valeurs associées à cette clé et faire la somme de ces valeurs. Le résultat obtenu est renvoyé comme sortie.

d) Flux d'exécution d'une opération MapReduce :

Quand le programme d'utilisateur fait appelle à la fonction MapReduce, les séquences suivantes qui sont illustré par la figure 1.3 se produit [10] :

1. La librairie MapReduce dans le programme, commence par diviser le flux d'entée en M morceaux. La taille de ces morceaux varie de 16 à 64 MB. Des copies du programme sont exécutées en parallèle sur plusieurs noeuds.
2. Une de ces copies est particulière : le maître (master). Les autres sont des simples esclaves qui se voient assigné des tâches par le maître. "M" Map et "R" Reduce doivent être assignées.
3. Un esclave (Worker) qui se voit attribué une tâche lit le « split » correspondant en entrée. Il en déduit un ensemble de clé/valeur selon la fonction Map définie par l'utilisateur. Les résultats sont stockés dans des buffers.
4. Périodiquement, ces buffers sont écrits sur le disque local, préalablement partitionné en R régions. Les adresses de ces régions sont indiqués au maitre, qui les transferts aux Reduce esclaves.
5. Le Reduce esclave lit les buffers de donnés contenu sur le disque local des Map esclaves. Il trie ces valeurs intermédiaires par clés : elles sont regroupées par clé.
6. Une fois que les valeurs sont regroupées par clés unique, ces jeux sont passés à la fonction Reduce définie par l'utilisateur. La sortie de la fonction Reduce est stockée dans un fichier final.
7. Quand toutes les tâches Map/Reduce sont effectuées, le maitre réveille le programme utilisateur.

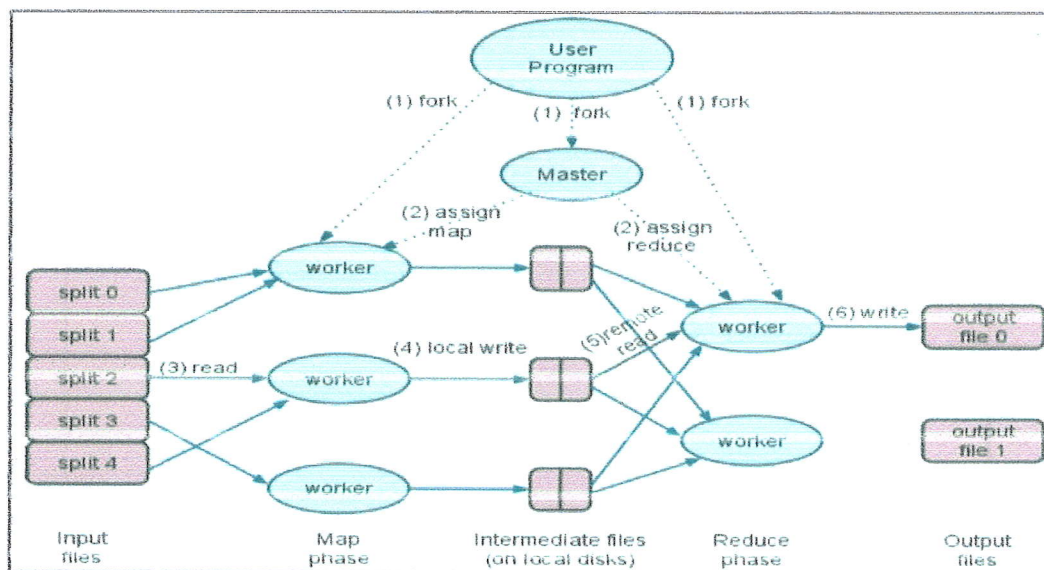


Figure 1.3 : Flux d'une opération MapReduce.

e) Avantages de MapReduce :

- Largement utilisé dans les environnements de Cloud Computing.
- peu de tests sont nécessaires. Les bibliothèques MapReduce ont déjà été testées et fonctionnent correctement.
- Cache les détails de parallélisme aux utilisateurs.

f) Inconvénients de MapReduce :

- Elle ne supporte pas les langages hauts niveau comme le SQL.
- Une seule entrée pour les données
- Les opérations de MapReduce ne sont pas toujours optimisées pour les entrées/sorties. De plus, les méthodes Map () et Reduce () sont bloquantes. Cela signifie que pour passer à l'étape suivante, il faut attendre que toutes les tâches de l'étape courante soient terminées.
- MapReduce n'a pas de plan spécifique d'exécution et n'optimise pas le transfert de données entre ces nœuds.
- MapReduce n'est pas open source et il est utilisé seulement chez Google.

1.8.5.3 Hadoop

a) Définition :

Hadoop (High Availability Distributed Object Oriented Platform) est un framework Java libre géré par la Fondation Apache destiné pour faciliter la création des applications distribuées et scalable [2]. Il a été mis en avant par des grands acteurs du web tels que Yahoo et Facebook. Les deux composants principaux de Hadoop sont le framework MapReduce et le system de fichier distribué HDFS (Hadoop Distributed File System). Dans Hadoop, le nœud maître est appelé le JobTracker et les nœuds esclaves sont appelés TaskTracker. Chaque nœud esclave va contenir les blocs de données en les répliquant. Le nœud maître connaît les emplacements des différentes répliques. Le nœud maître secondaire sert à effectuer des sauvegardes régulières du nœud maître afin de pouvoir le réutiliser en cas de problème [42].

b) HDFS :

HDFS est un système de fichier distribué, extensible et portable inspiré par le Google File System (GFS). Il a été conçu pour stocker de très gros volumes de données sur un grand nombre de machine équipées de disques durs banalisés. Il permet l'abstraction de l'architecture physique de stockage afin de manipuler un système de fichier distribué comme s'il s'agissait d'un disque dur unique [47]. Un cluster HDFS repose sur deux types de composants majeurs [2] :

-NameNode(le maître HDFS) : Ce composant gère l'espace de noms, l'arborescence du système de fichiers et les métadonnées des fichiers et des répertoires. Il centralise la localisation des blocs de données répartis dans le cluster. Il est unique mais dispose d'une instance secondaire qui gère l'historique des modifications dans le système de fichiers.

-DataNode (l'esclave HDFS): Ce composant stocke et restitue les blocs de données. Lors du processus de lecture d'un fichier, le NameNode est interrogé pour localiser l'ensemble des blocs de données. Pour chacun d'entre-eux, le NameNode renvoie l'adresse du DataNode le plus accessible, c'est-à-dire le DataNode qui dispose de la plus grande bande passante. Les DataNodes communiquent de manière périodique au NameNode. Si certains blocs ne sont pas assez répliqués dans le cluster, l'écriture de ces blocs s'effectue en cascade par copie sur d'autres.

c) Hadoop MapReduce :

MapReduce permet de manipuler de grandes quantités de données en les distribuant dans un cluster de machines. Ce modèle connaît un vif succès auprès de sociétés possédant d'importants centres de traitement de données telles Amazon ou Facebook. Hadoop MapReduce est une implémentation open source de Google. Il est basé sur une architecture maître-esclave :

- **Hadoop job** : Est une unité de traitement mettant en oeuvre un jeu de données en entrée, un programme MapReduce et des éléments de configuration.
- **Hadoop JobTracker (maître)** : Coordonne l'exécution des jobs sur l'ensemble du cluster. Il communique avec les TaskTrackers en leur attribuant des tâches d'exécution (Map ou Reduce). Il assure leur suivi et ré-exécute les tâches échouées. Le JobTracker est un démon cohabitant avec le NameNode. Il n'y a donc qu'une instance maître par cluster.
- **Hadoop TaskTracker (esclave)** : Exécutent les tâches (Map ou Reduce). Par ailleurs, ils notifient périodiquement le JobTracker du niveau de progression d'une tâche ou bien le notifient en cas d'erreur afin que celui-ci puissent reprogrammer et assigner une nouvelle tâche. Un TaskTracker est un démon cohabitant avec un DataNode.

-Les avantages de Hadoop :

- **Robuste** : Si un nœud de calcul tombe en panne, ses tâches sont automatiquement réparties sur d'autres nœuds. Les blocs de données sont également répliqués.
- **Crée spécialement pour les gros volumes de données** : Largement utilisé sur le Cloud, par exemple : MapReduce pour l'analyse des requêtes, etc.
- **Rentable** : Car il optimise les coûts via une meilleure utilisation des ressources existants.
- **Souple** : Capable de traiter différents types de données (structurées, non structurées).
- **La performance** : Contrairement aux bases de données traditionnelles, Hadoop permet aux applications d'analyser des téraoctets de données très rapidement tout en garantissant les performances.
- **La rapidité** : La structure distribuée HDFS permet de localiser et de récupérer la donnée où qu'elle se trouve. Ainsi, Hadoop permet de traiter des requêtes en quelques minutes sur des téraoctets de données et sur des pétaoctets en quelques heures.
- **Open source** : Il est faisable pour l'évaluer.

-Les inconvénients de Hadoop :

Hadoop reste un sujet de discussion populaire lorsque l'on parle d'environnements Big Data, la technologie est aussi la cible d'une quantité de critiques. Non seulement la technologie est assez complexe et requiert des compétences précises, il présente l'absence de sécurité [5]. Dans sa version initiale, Hadoop ne disposait pas de fonctionnalités de sécurité garantissant un déploiement sans risque [6].

1.9 Big Data et Cloud Computing

Aujourd'hui le Cloud Computing est devenue une solution fiable pour le traitement, le stockage et la distribution des données. Avec sa réputation progressive, des multiples applications se déplacent vers le Cloud. Le Big Data est l'un des plus célèbres applications du Cloud.

Il devient plus difficile de les traitées on utilisant les outils de gestion de BD traditionnelles. Donc le Cloud Computing se diffère d'un environnement traditionnel par ses nouvelles technologies qui sont développées pour faire face à la volumétrie de données et la charge de travail. Ces nouvelles technologies, dont les plus populaires sont MapReduce/Hadoop et BD NoSQL, sont jugés les meilleurs candidats pour le traitement de Big Data.

Généralement, les données de Big Data sont fortement liées. Ces relations complexe et indépendante sont plus naturellement modélisées et analysées on utilisant les graphes. De ce fait, les graphes sont des techniques importantes pour la découverte et l'exploration de Big Data sur le Cloud.

1.10 Les Graph**1.10.1 Définition**

Dans sa définition la plus simple, un graphe est un ensemble de nœuds reliés entre eux par des liens. Formellement, nous définissons un graphe $G = (V, E)$ par son ensemble de nœuds V et son ensemble de liens E qui connectent les nœuds entre eux.

1.10.2 Représentation des graphes

Il existe deux façons classiques de représenter un graphe en machine : par une matrice d'adjacence ou par un ensemble de listes d'adjacence [45].

- **Matrice d'adjacence:**

Soit le graphe $G = (S, A)$. On suppose que les sommets S sont numérotés de 1 à n , avec $n = |S|$. La représentation par matrice d'adjacence de G consiste en une matrice booléenne M de taille $n \times n$ telle que: $M_{ij} = 1$ si un lien de nœud i à j et $M_{ij} = 0$ sinon (figure 1.5). Cette représentation permet d'accélérer la recherche et de simplifier les algorithmes de calcul. Cependant, elle ne convient qu'aux des graphes simples et permet un stockage inutile des cas inintéressants (les zéros de la matrice) et résulte une redondance des informations pour les graphes non orientés [48].

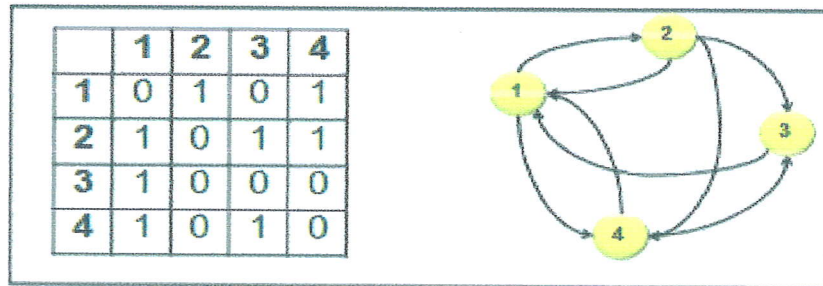


Figure 1.4: Représentation des graphes sous matrice d'adjacence.

- **Listes d'adjacence:**

Soit $G = (V, E)$ un graphe. La représentation par liste d'adjacence de G (figure 1.6) consiste à associer à chaque sommet du graphe la liste de ses voisins [7]. Cette représentation est beaucoup plus compacte et facilite le calcul sur les liens de sortie, mais il reste beaucoup plus difficile de calculer sur les liens d'entrée.

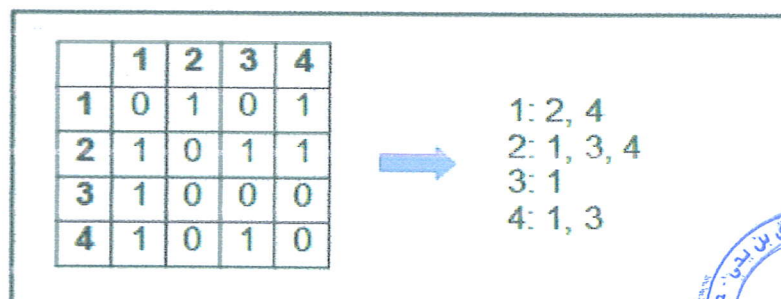


Figure 1.5: Représentation d'un graphe par liste d'adjacence.

1.10.3 Domaine d'application des graphes

- Le web : chaque page est un sommet du graphe, chaque lien hypertexte est une arête entre deux sommets.
- Un réseau ferroviaire : chaque gare est un sommet, les voies entre deux gares sont les arêtes.
- Un réseau social : les sommets sont les personnes, deux personnes sont adjacentes dans ce graphe lorsqu'elles sont "amies". Si la notion d'amitié n'est pas réciproque, le graphe est orienté [25].

1.11 Graph et Big Data

Les graphes sont devenus ces dernières années, notamment grâce à l'étude des réseaux sociaux, le principal outil de représentation de relations forte ou faible entre variables dans tous les champs d'application. On retrouve les graphes à divers niveaux de description des mécanismes en biologie systémique, les réseaux métaboliques (graphes entre réactions et produits de réaction) ; les réseaux protéines/protéines (graphes de proximité des séquences protéiques); les réseaux de régulation (graphes des interactions du produit d'un gène sur la transcription d'autres gènes) ; ou les réseaux de co-expression (graphes des corrélations entre niveau d'expressions de gènes). Si l'analyse des réseaux biologiques a beaucoup emprunté aux télécommunications et aux sciences sociales [11].

Le succès des réseaux sociaux (par exemple Facebook) et les applications Web, et la haute disponibilité de sources de données, les données du graphe sont produites à un taux incomparable. Ils sont maintenant mesurés dans les téraoctets et conduire vers petabyte, avec plus de billions de noeuds et arc. Par exemple, Facebook charge 60 téraoctets de nouvelles données tous les jours, Yahoo avait un 1.4 milliard noeuds Web graphiques à 2002, Microsoft avait 1.15 milliard question URL paires à 2009, et Google traite 20 petabyte par jour [31]. Cependant, les graphes obtenus de ces grandes masses de données (Big Data) sont de grands graphes (Big Graphes). Leur exploration nécessite des nouveaux outils efficaces.

1.12 Frameworks de traitement de Big graphs

Il existe des framework se basant sur des modèles connus comme le MapReduce ou des dérivés qui ont de nombreux avantages comme celui de pouvoir traiter de gros volumes de données dans un environnement distribué de manière relativement simple. Néanmoins,

MapReduce ne supporte pas les applications d'analyse des données itératives directement. Parmi les défis engendrés par le traitement itératif des larges graphes avec MapReduce que les données doivent être échangées d'itération à itération à travers le réseau et doivent être rechargées et retraitées à chaque itération, en gaspillant les E/S, bande passante du réseau, et les ressources de CPU. Ainsi, le programme peut prendre un longtemps si beaucoup d'itérations sont exigées. Afin de simplifier la programmation d'algorithme de graphe dans un environnement distribué, un nouveau modèle a été créé : Pregel.

Pregel a été introduit par Malewicz et al [18] en 2010. Il utilise le modèle de programmation de BSP, il est tolérant aux pannes, et construit une plate-forme évolutive pour le traitement des larges graphes. Son API flexible a été prouvée pour être puissante afin d'exprimer des algorithmes de graphe et pour fonctionner efficacement sur plusieurs clusters. Pregel concentre le calcul sur les sommets du graphe et ses flux d'exécution sont exprimés par des primitives de BSP. Il est bon de noter qu'il ya un autre framework Giraph qui est le plus utilisé car il est une alternative open-source à Google Pregel et implémente le même modèle aussi utilise plusieurs sous-projets Apache dont Hadoop (infrastructure basée sur le modèle MapReduce) [26].

Apache Giraph est un framework de traitement de graphe conçu pour être hautement scalable et donc pouvoir traiter de gros volumes de données. Il existe des alternatives à Giraph comme Neo4j et GraphLab ne se basant pas sur les mêmes framework ni sur les mêmes modèles, mais Giraph propose des fonctionnalités avancées et semble avoir de bonnes performances en pratique.

1.13 Conclusion

Dans ce chapitre nous avons présenté le Cloud Computing comme concept de base avec quelques notions liées à ce terme, ensuite nous avons abordé l'évolution massive des données (Big Data) qui se caractérisent par le volume, la variété, la vélocité, la vitesse et qui posent des défis significatifs avec les nouvelles solutions qui remplacent les BD relationnelles. Nous avons aussi abordé les big graphes comme principal outil de représentation de relations forte ou faible entre variables dans tous les champs d'application avec les outils de traitement parallèle de ces graphes sur le cloud comme MapReduce, Hadoop, Pregel et Giraph. Ces deux derniers seront plus détaillés dans le chapitre suivant.

Chapitre 2 :
Giraph et les algorithmes de
graphes

2.1 Introduction

Depuis quelques années il y a une évolution rapide des données connu sous le nom de (Big Data) soulève de nombreuses problématiques. En particulier, être capable de stocker, partager et analyser de telles quantités de données constitue un enjeu d'étude essentiel.

Dans de nombreux cas, ces données peuvent être représentées en utilisant la théorie des graphes. Cette dernière est particulièrement appropriée pour étudier les réseaux sociaux, où les connexions entre utilisateurs peuvent facilement être représentées et analysées en utilisant des graphes, le plus souvent orientés. Cependant, le nombre d'utilisateurs des réseaux sociaux sur Internet a littéralement explosé récemment, ce qui aboutit à la création de graphes de très grande taille, dont les propriétés structurelles sont difficiles à étudier.

Afin de pouvoir analyser efficacement des réseaux d'une telle taille, le parallélisme et la distribution des algorithmes au sein d'un parc de serveurs ou d'un Cloud devient nécessaire. Ainsi, de nombreux framework spécifiques au développement d'algorithmes de graphes distribués inspirés de l'approche Pregel sont nés : Apache Giraph, Apache Hama. Ceux-ci forcent à implémenter les algorithmes selon un modèle précis et donc à les repenser partiellement.

Dans ce chapitre nous allons détailler le framework Giraph avec une description détaillée de quelques algorithmes nécessaires pour parcourir des graphes sous Giraph. Mais avant de parler de ce framework, il est nécessaire de détailler le modèle Pregel sur lequel il repose.

2.2 Pregel

Pregel est un outil de traitement parallèle des graphes sur le Cloud. En 2010, Google a introduit le système Pregel [37] comme une plate-forme évolutive pour la mise en œuvre des algorithmes de graphes. Ce framework repose sur une approche de sommet centré et est inspiré par le modèle Bulk Synchronous Parallèle (BSP). Dans Pregel, le plus petit élément manipulable est un nœud. Ce nœud possède un identifiant, des propriétés, des arcs sortants (avec des propriétés) qui pointe sur des nœuds (via leurs identifiants) et un état (actif ou non). Chaque nœud détient en plus une boîte-aux-lettres de messages reçus, lui permettant de lire des messages provenant d'autres nœuds qui peuvent être connecté. Un nœud peut aussi envoyer des messages à d'autres nœuds. Seuls les nœuds actifs travaillent. Un nœud actif peut s'endormir, il devient alors inactif et sera réveillé lorsqu'il recevra un message. Le traitement

du graphe prend fin lorsque tous les nœuds sont inactifs. Fondamentalement, chaque nœud actif va donc lire des messages dans sa boîte-aux-lettres puis les traiter et finalement retransmettre d'autres messages à ces voisins si nécessaire [26].

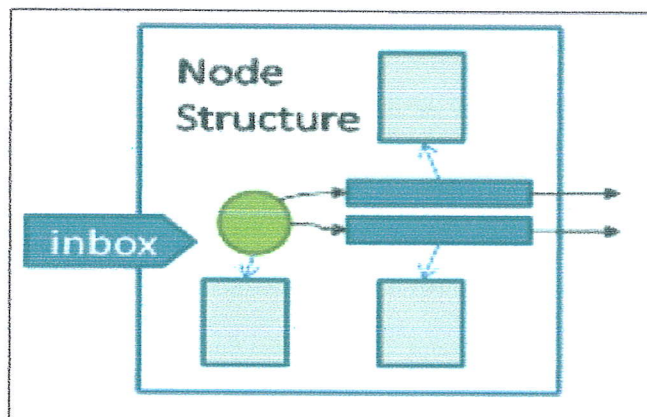


Figure 2.1: Structure d'un nœud dans le modèle Pregel.

Le graphe est décomposé en multiple groupe appelés partitions et chacune contient un grand nombre de nœuds et chaque partition est traitée de manière séquentielle.

Le modèle d'exécution utilisé est le BSP (Bulk Synchronous Processing). Dans ce modèle, une multitude de tâche s'exécute en parallèle sous forme de séquences appelées Supersteps. Dans chaque Superstep, les tâches commencent par recevoir tous les messages provenant des tâches du Superstep précédent, les traitent et peuvent envoyer d'autres messages. La transmission des messages se fait de manière asynchrone et simultanée sur toutes les tâches. Les messages envoyés lors d'un Superstep ne seront pas accessibles avant le Superstep suivant. Lorsque toutes les tâches se sont terminées, un nouveau Superstep peut être démarré et ainsi de suite jusqu'à ce qu'une condition d'arrêt soit vraie (voir la figure 2.2).

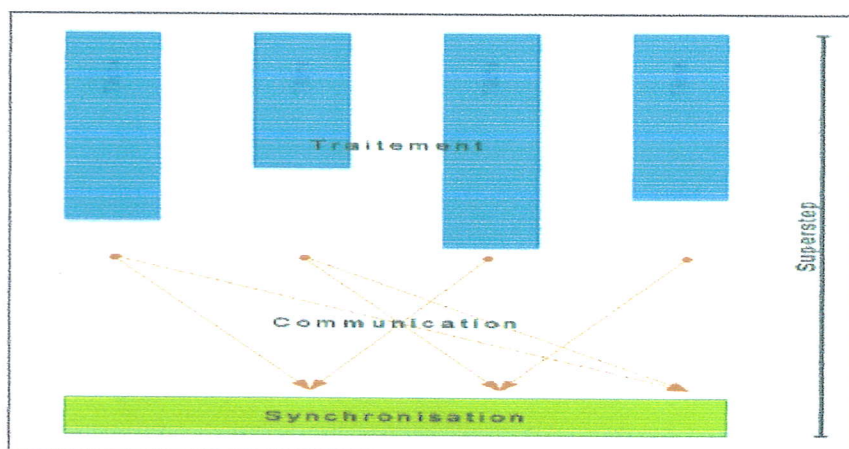


Figure 2.2: Fonctionnement de BSP.

Pregel est un framework fiable et efficace pour le traitement des larges graphes, mais malheureusement il est utilisé seulement chez Google. Par conséquent, en 2012 la fondation Apache a développé un nouveau framework libre et open source inspiré de Pregel et basé sur un cluster Hadoop. Ce nouveau framework nommé Giraph.

2.3 Apache Giraph

En 2012, Apache Giraph lancé comme un projet open source de Pregel. Giraph peut fonctionner comme un job Hadoop typique qui utilise l'infrastructure de cluster Hadoop [10].

2.3.1 Définition

Giraph est un projet Apache pour le traitement des graphes sur des volumes importants de données. Giraph s'appuie sur l'implémentation de MapReduce réalisée par Apache Hadoop afin de traiter les graphes (voir figure 2.3).

Facebook a utilisé Giraph avec quelques améliorations de performances pour analyser 1000 milliard d'arêtes (relations) en 4 minutes et en utilisant 200 machines.

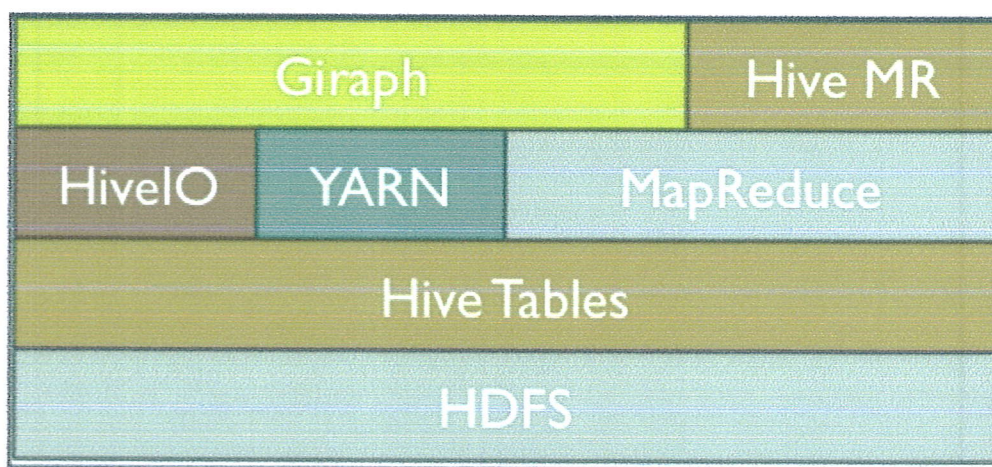


Figure 2.3: Giraph intégré avec Hadoop [10].

2.3.2 Architecture

L'architecture de Giraph se compose d'un maître, de nombreux travailleurs (worker), et un Apache ZooKeeper (Voir la figure2.4) :

- **Maitres :** Il y a toujours un service maître actif et quelques maîtres candidats à devenir actifs si le maître courant échoue. Les maîtres candidats sont en veille et ne jouent pas un rôle actif dans le cycle de vie de fonctionnement de Giraph. Une fois un

maître devient actif, son travail consiste à coordonner le calcul. Cette tâche consiste à procéder comme suit [10]:

- La transition des workers d'un Superstep à l'autre de manière coordonnée.
 - Avant chaque Superstep, l'attribution des partitions pour les workers actifs.
 - L'exécution du code maître de calcul.
 - Le suivi de l'état et des statistiques de tous les workers.
- **Worker** : Les workers représentent la majorité des services de Giraph. Leur fonction principale est de gérer l'état de partitions de graphe qui leur sont assignées. Chaque worker expose un ensemble d'API de réseau pour permettre aux workers distants de manipuler les données dans les partitions qui lui sont assignées.

Au cours de chaque Superstep, les workers itère sur les partitions de graphes qu'ils possèdent et exécute la méthode *Compute()* pour chaque sommet appartenant à ces partitions. Les workers sont également responsables de sauvegarde de points de contrôle de leur état de temps en temps comme un moyen de récupération d'une défaillance du worker [10].

- **Coordonneurs** : Sont des nœuds exécutant le service de coordination. Ils fournissent une configuration distribuée, la synchronisation, et de nommer des services de registre pour le reste de Giraph. Il existe plusieurs implémentations de services de coordination, mais le choix par défaut dans Hadoop écosystème a toujours été Apache ZooKeeper. Une collection de nœuds exécutant un service de coordination est collectivement connu sous le nom d'un ensemble de ZooKeeper [10].

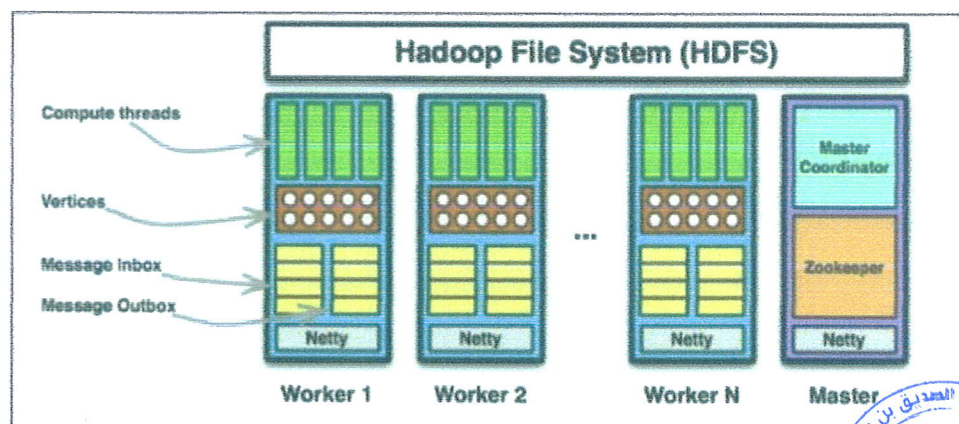


Figure 2.4: Architecture de Giraph [10].



2.4 Modèle de donnée

Un cluster Giraph se compose d'un maître, de plusieurs workers et d'un ensemble d'apache Zookeeper qui maintient l'état global de l'application ou le job de Giraph. Le maître actif fonctionne comme un coordonnateur de l'application. Il synchronise les Supersteps. Si le maître échoue, un autre maître de la file d'attente principale de ZooKeeper peut être activé, ainsi la tolérance aux pannes est intégrée dans ce dernier.

Les workers reçoivent une partition du graphe assignée du maître avant que le Superstep commence. Chaque worker gère les opérations d'entrée/sortie, la partie de calcul de l'algorithme et les messages entre les nœuds.

La figure 2.5 illustre les trois phases de flux de données dans une application Giraph. Dans la **phase1** il commence à charger le graphe. Au cours de cette phase le maître attribue l'entrée partitionnée aux workers, qui sont responsables du chargement de données. Ce processus est similaire au traitement classique de MapReduce. La **phase2** est la partie itérative de l'algorithme, qui se compose par des Supersteps concaténés. Enfin, chaque worker responsable du format de sortie et apporte les données de toutes les partitions dans le but d'écrire les résultats dans HDFS, HBase ou même Accumulo.

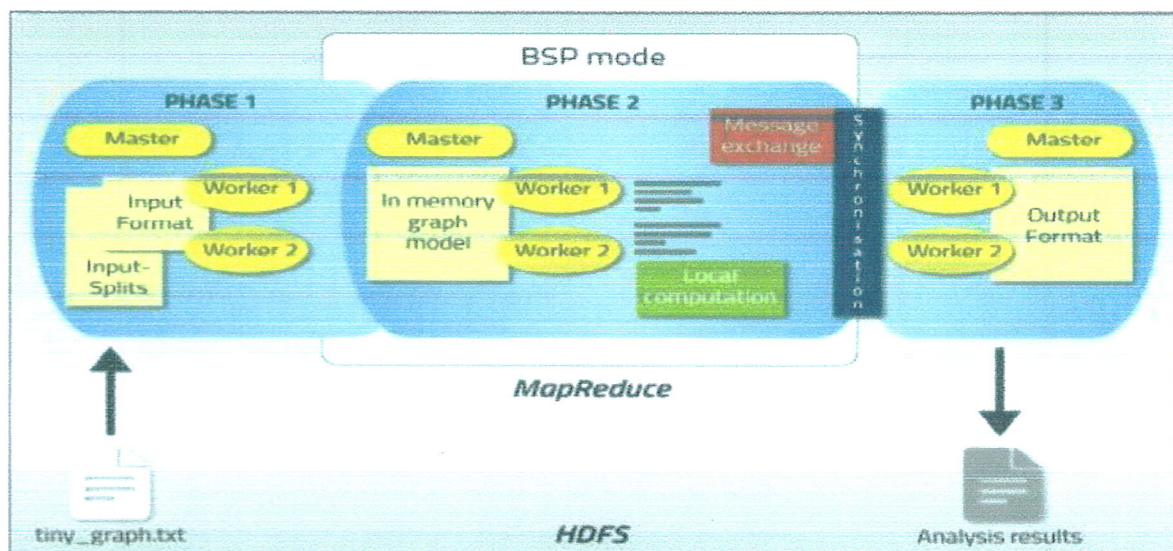


Figure 2.5: Le flux de données de Giraph dans une implémentation MapReduce classique [22].

A chaque Superstep, la méthode `Compute ()` est appelée pour chaque sommet. Un sommet a les données, qui décrivent l'état du nœud, et sa méthode de calcul fait partie de l'implémentation d'un seul algorithme [22].

2.5 Modèle de traitement

Dans Giraph, les programmes de traitement des graphes sont exprimés sous forme d'une séquence d'itérations appelés Superstep. Au cours d'une Superstep, le framework commence une fonction définie par l'utilisateur pour chaque sommet, conceptuellement en parallèle. La fonction définie par l'utilisateur spécifie le comportement à un seul sommet V et un seul Superstep S . La fonction peut lire les messages qui sont envoyés à V dans le Superstep $S-1$, envoi des messages à d'autres sommets qui seront reçus au Superstep $S+1$, et de modifier l'état de V et ses arcs sortants. Les messages sont généralement envoyés le long des arcs sortants, ou à un identificateur connu. Chaque Superstep représente des unités atomiques de calcul parallèle [45].

Dans ce modèle de programmation, tous les sommets sont actifs au Superstep 1. Tous les sommets actifs exécutent la méthode `Compute()` à chaque Superstep. Chaque sommet peut se désactiver en votant pour arrêter (`vote-to-halt`) et se transforme en état inactif si elle ne reçoit pas un message. Un sommet peut revenir à l'état actif si elle reçoit un message dans l'exécution de toute Superstep ultérieure. Ce processus, illustré sur la figure 2.6, se poursuit jusqu'à ce que tous les sommets deviennent inactifs et aucun message n'en transit. Par conséquent, l'exécution du programme se termine lorsque, à un moment où tous les sommets sont inactifs. Pour bien illustrer le modèle de traitement du Giraph, la figure 2.7 présente les différents Supersteps de calcul d'un exemple sous Giraph. Cet exemple cherche la valeur maximale de sommet dans un graphe.

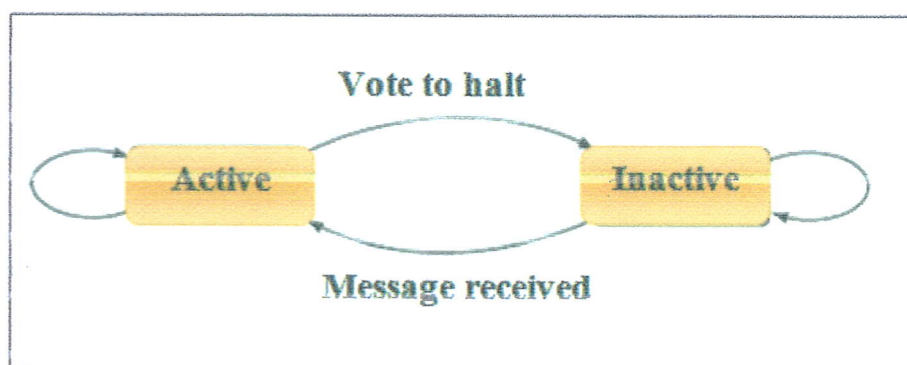


Figure 2.6: Plan décrit la transition entre les états actifs et inactifs.

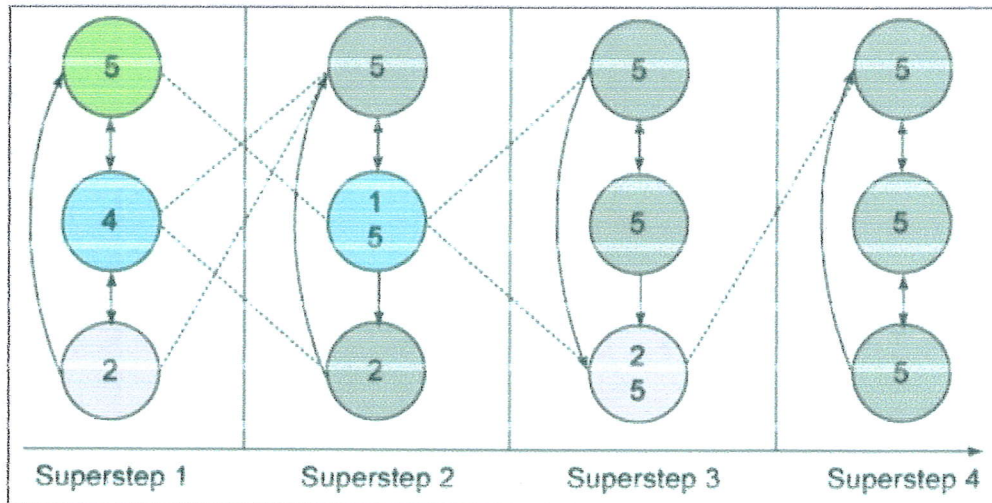


Figure 2.7: Exemple de calcul de la valeur maximale de sommet dans BSP.

Dans Superstep1, chaque sommet envoie sa valeur à son sommet voisin. Dans Superstep 2, chaque sommet compare sa valeur avec la valeur reçue de son sommet voisin. Si la valeur reçue est supérieure à la valeur de sommet, il met à jour sa valeur à la valeur la plus élevée et envoie la nouvelle valeur de son sommet voisin. Si la valeur reçue est inférieure à la valeur de sommet, le sommet garde sa valeur et vote pour s'arrêter (vote-to-halt). Ainsi, dans Superstep 2, seul le sommet avec la valeur 1 met à jour sa valeur à plus forte valeur reçue (5) et envoie sa nouvelle valeur. Ce processus se produit de nouveau dans Superstep 3 pour le sommet avec la valeur 2, alors que dans Superstep 4 tous les sommets votent pour arrêter et le programme se termine.

2.6 Les étapes d'exécution d'un programme Giraph

Le nœud maître attribue les partitions aux workers, coordonne la synchronisation. Comme Hadoop, Giraph utilise Apache ZooKeeper pour la synchronisation. Les workers sont responsables de sommets. Un worker commence la fonction Compute () pour les sommets actifs. Il envoie également (reçoit) des messages aux autres sommets. Pendant l'exécution, si un worker reçoit une entrée qui n'est pas pour ses sommets, il passe le long (voir la figure 2.8).

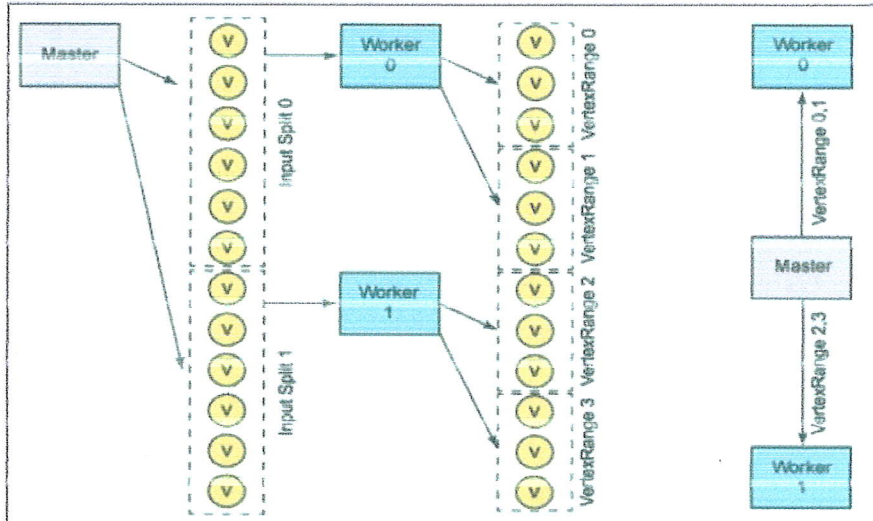


Figure 2.8: Exemple de valeur maximale (Les lignes pointillées sont des messages, les sommets ombragés ont voté pour stopper).

2.7 Tolérance aux pannes

La tolérance aux pannes est accomplie à travers des checkpoint (points de contrôle). Au début d'un Superstep, le maître informe les esclaves de sauvegarder leur état, y compris les valeurs du sommet, les valeurs d'arc, et les valeurs d'entrée [19]. Le maître détecte les esclaves en panne par l'envoi de manière régulière des messages de "Ping". Si un esclave ne reçoit pas un message du "Ping" après une période spécifiée, le processus esclave se termine. Si le maître n'a pas reçu une réponse d'un esclave, le maître marque que le processus esclave est mort. Quand un esclave ou plus mort, l'état courant des partitions assignées à ces esclaves est perdue. Le maître réaffecte ces partitions du graphe à l'ensemble des esclaves actuellement disponibles et ils rechargent leur états à partir de checkpoint disponible au début d'un Superstep S .

2.8 Compréhension des applications en Giraph

Une fois que le graphe est configuré, les étapes de traitement réelles pour l'algorithme souhaité peuvent être mises en œuvre. Giraph utilise une conception centrée sur le sommet. Afin d'illustrer ce que cela signifie, trois algorithmes seront abordés: plus courts chemins, Page Rank et Composante connexe, mais avant de commencer l'explication de ces trois algorithmes il faut tout d'abord définir les flux d'entrée et de sortie de ces algorithmes. Pour l'exécution réelle de ces algorithmes sous Giraph voir l'annex A.

-Déterminer le format d'entrée de données : La première étape d'une application Giraph consiste à déterminer le format de données d'entrée d'un graphe. Il ya plusieurs structures de donnée d'entrée (par exemple : le format Json) (voir la figure 2.9).

- ID Vertex : Vertex ID est l'identifiant d'un sommet dans le graphe qui peut être une étiquette simple ou complexe représenté sous une forme que Giraph peut analyser à partir du fichier d'entrée.

-Vertex Value : Contient des informations supplémentaires associées à un sommet. Typiquement, ce champ contient des valeurs ou des objets qui doivent être mis à jour pendant le traitement de graphe.

-Tuples d'arcs : Contient des informations nécessaires pour définir l'ensemble des arcs-sortant associés à l'identifiant d'un sommet source. Cette information est composée de tuples à deux éléments par arc: l'identifiant d'un sommet de destination et le poids d'arc, où le poids d'arc est facultatif.

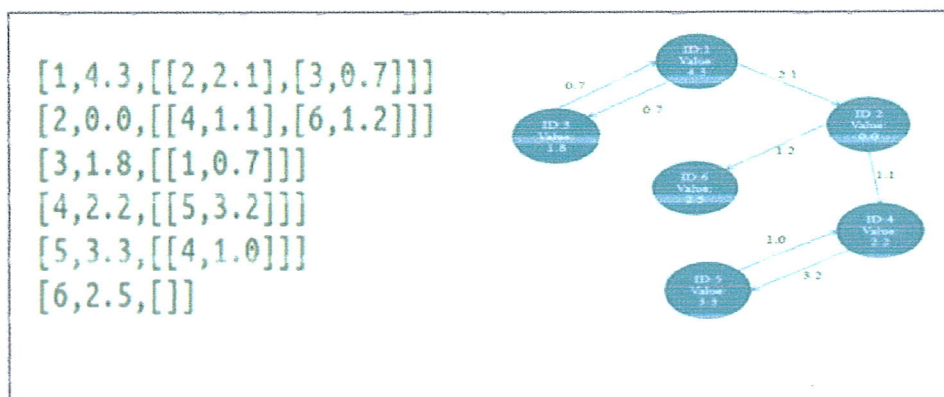


Figure 2.9: Format d'entré Json sous Giraph.

-Déterminer le format des données de sortie :

La deuxième étape permet aux applications Giraph de déterminer le format de données pour les résultats obtenus. Giraph a des formats de sortie intégrés pour cela aussi, où l'un les plus courantes est le "IdWithValueTextOutputFormat". Cela va écrire la sortie du texte de façon que tous les ID_Vertex est imprimé avec sa valeur de sommet.

2.8.1 Exemple de SSSP (Single Source ShortestPath) dans un graphe orienté

Pour mieux comprendre l'algorithme de Shortestpath nous allons expliquer un exemple de SSSP dans un graphe orienté. Cet exemple illustré par la figure 2.10 calcule le plus court chemin d'un nœud source (le nœud numéro1) à tous les autres nœuds. Au début, tous les nœuds initialisent leur valeurs à l'infinie, sauf le nœud source à 0. Dans le Superstep 0, le nœud source va se propager sa distance à ses voisins. Dans les Superstep suivantes, chaque sommet reçoit les coûts de ses voisins, si la valeur est inferieur de la valeur de sommet actuel, le sommet met à jour sa valeur et envoi ces mises à jour à ses voisins. Ces voisins à leur tour vont mettre à jour leurs valeurs et les envoient à leurs voisins. L'algorithme se termine lorsqu'aucune mise à jour ne produit. Si un nœud n'a aucun voisin, sa valeur sera 0. Sinon, si le nœud est isolé, sa valeur reste infinie. Par la suite la valeur associée à chaque sommet désigne la distance minimale entre le sommet de source et ce sommet. Le pseudo code de cet algorithme en Giraph est présenté par la figure 2.11.

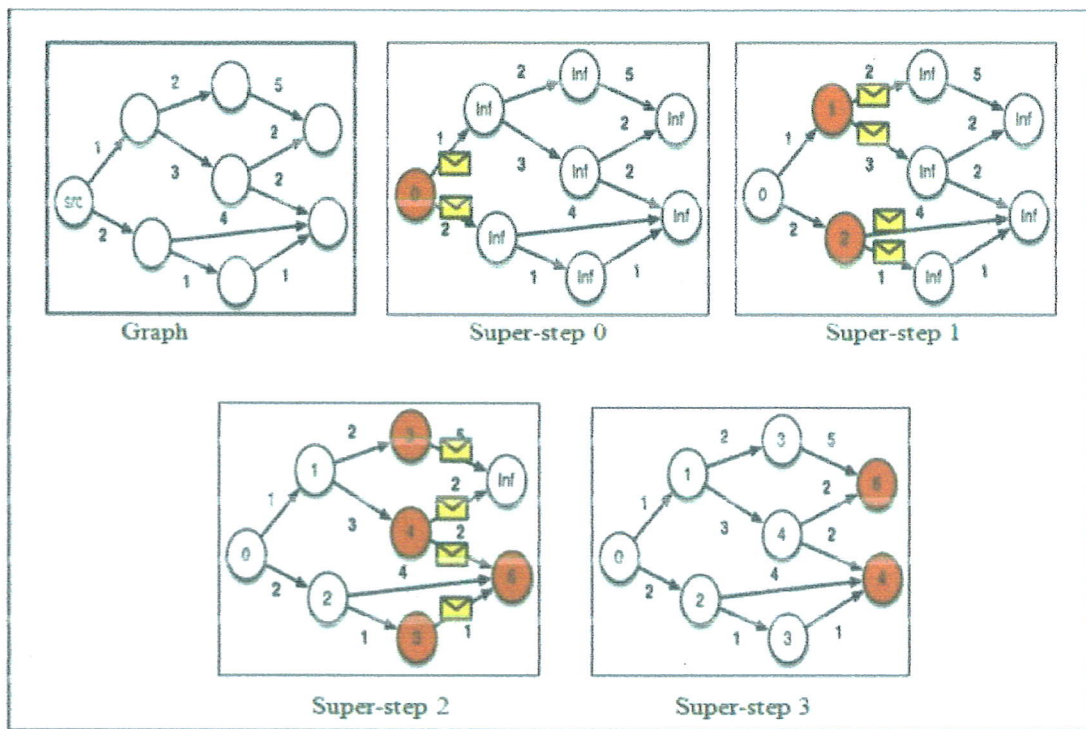


Figure 2.10: Exemple de SSSP sous Giraph.

```

public static class ShortestPathVertex extends Vertex<Text, IntWritable, IntWritable> {
    public void compute(Iterator<IntWritable> messages) throws IOException {
        int minDist = isStartVertex() ? 0 : Integer.MAX_VALUE;
        while (messages.hasNext()) {
            IntWritable msg = messages.next();
            if (msg.get() < minDist) {
                minDist = msg.get();
            }
        }
        if (minDist < this.getValue().get()) {
            this.setValue(new IntWritable(minDist));
            for (Edge<Text, IntWritable> e : this.getEdges()) {
                sendMessage(e, new IntWritable(minDist + e.getValue().get()));
            }
        } else {
            voteToHalt();
        }
    }
}

```

Figure 2.11: La méthode Compute() de Shortestpath sous Giraph.

2.8.2 Page Rank

PageRank est une méthode utilisée par Google pour calculer l'importance d'une page sur le web, alors que les balises, les titres et la fréquence de mots clés sont pris en considération, Google utilise cet indice pour ajuster ces résultats et permettre aux sites les plus importants de gagner des places dans les résultats de recherche pour des requêtes données [8].

Dans cet algorithme le type de valeur du sommet est un nombre réel pour stocker un PageRank provisoire, et le type de valeur d'arc est nul parce que les arcs ne stockent pas d'informations. On suppose que le graphe est initialisé dans le Superstep 0 et la valeur de chaque sommet est $1/\text{NumVertices}()$. Dans chacune des 30 premiers Supersteps, chaque sommet envoie le long de chaque arc sortant son PageRank provisoire divisé par le nombre d'arcs sortants. A partir du Superstep 1, chaque sommet résume les valeurs qui arrivent sur les messages en sum, et définit son propre PageRank provisoire à $0.15/\text{NumVertices}() + 0.85 \times \text{sum}$. Après avoir atteint le Superstep 30, aucun autre message ne sera envoyé et chaque sommet vote pour s'arrêter [40]. La figure 2.12 montre un exemple de déroulement de cet algorithme sous Giraph et la figure 2.13 présente le pseudo code de la méthode compute du calcul de PageRank dans un graphe.

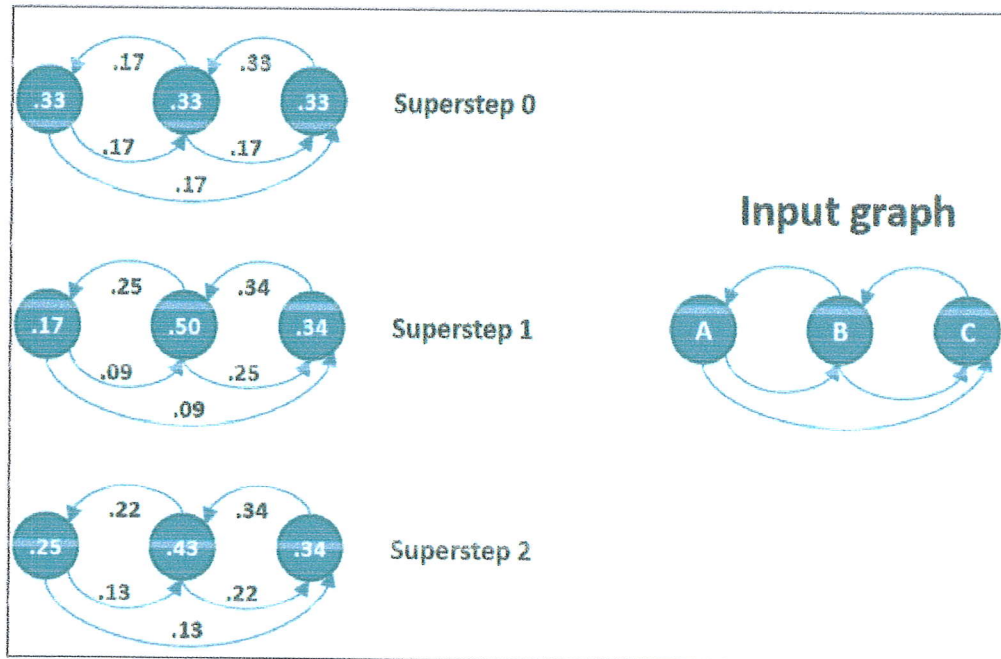


Figure 2.12: Exemple de PageRank sous Giraph.

Dans la pratique, un algorithme PageRank fonctionnerait jusqu'à ce que la convergence soit atteinte, et les agrégateurs seraient utiles pour la détection de la condition de convergence.

```
function compute(vertex, messages){
  if (getSuperstep() == 0){
    vertex.setValue(1 / getTotalNumVertices());
  }
  else{
    double prSum = 0;
    for (DoubleWritable message : messages) {
      prSum += message.get();
    }
    pr = 0.15 / getTotalNumVertices() + 0.85 * prSum;
    vertex.setValue(pr);
    if (getSuperstep() == NUMBER_OF_ITERATIONS){
      vertex.voteToHalt();
    }
    else{
      msg = vertex.getValue() / vertex.getNumEdges();
      sendMessageToAllEdges(vertex, msg);
    }
  }
}
```

Figure 2.13: La méthode Compute() de PageRank.

Le processus principal de cette méthode Compute () est de mettre à jour la valeur de sommet avec un calcul d'importance basée sur le nombre d'arcs. Elle est déterminée par itération sur tous les messages reçus, Il est important de noter que sur la première itération, l'ensemble de ce processus sera ignorée.

La dernière étape consiste à envoyer un message à travers tous les contrôles arcs-sortant afin que ces sommets puissent mettre à jour leurs valeurs d'importance. Ce procédé consiste à diviser la valeur actuelle attribuée à ce sommet par le nombre d'arcs-sortant. Ce processus se répète une fois pour chaque Superstep, jusqu'à ce que le nombre maximum de Superstep a été atteint - auquel les sommets voteront pour arrêter.

2.8.3 Composante connexe

Une composante connexe, dans un graphe non orienté, est un sous-graphe connexe du graphe. Un graphe peut être composé de plusieurs sous-graphes. Par exemple un réseau social composé de deux groupes d'amis, où aucun des individus du premier groupe ne connaît aucun individu du deuxième. Chacun d'entre eux est un composant différent, parce que les membres de chaque groupe sont déconnectés des membres de l'autre [10].

L'implémentation de cet algorithme se fonde sur l'idée que, dans une composante connexe, l'information peut propager à tous les sommets. En ce qui concerne Giraph, cela signifie que si un sommet envoie un message à ses voisins, et chaque voisin envoie ce message à ses voisins, le message a finalement atteint tous les sommets. Si les sommets ont des ID comparables tels que on peut trouver le maximum ou le minimum (des entiers ou des chaînes), chaque composante connexe peut être caractérisée par un minimum (ou maximum) ID d'un sommet appartenant à ce composant. Par la suite, chaque sommet propage la valeur min à ses voisins jusqu'à ce que tous les sommets reçoivent le plus petit ID dans le composant. Le pseudo code qui implémente cet algorithme est illustré par (la figure 2.14) et un exemple de déroulement de cet algorithme est illustré par la figure 2.15.

```
function compute(){
    if(getSuperstep()==0){
        vertex.setValue(vertex.getId());
    }elif{
        minID=min(message);
        if(minID<vertex.getValue()){
            vertex.setValue(minID);
            sendMessageToAllEdges(vertex,minID);
        }
        vertex.voteToHalt();
    }
}
```

Figure 2.14: La méthode Compute () de Composante connexe.

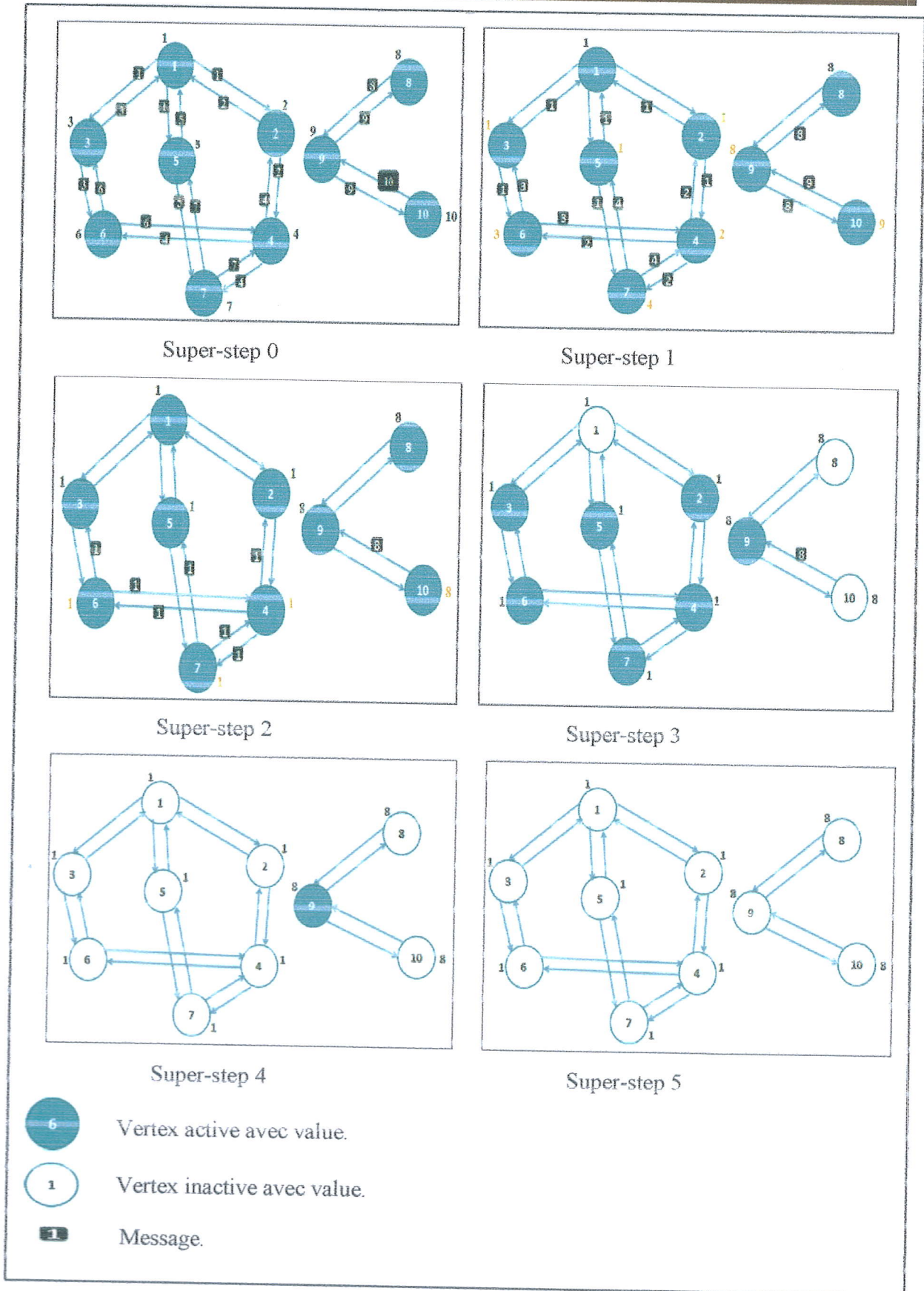


Figure 2. 15: Exemple d'algorithme Composante connexe sous Giraph.

Chapitre 3 :
Proposition des algorithmes de
graphe sous Giraph

3.1 Introduction

De nombreux réseaux peuvent être représentés par un graphe non orienté. Il en est ainsi par exemple des réseaux sociaux où la relation entre deux individus peut être représentée par l'arête d'un graphe dont les sommets sont les individus, de même en biologie l'interaction entre gènes peut être modélisée par un graphe. En Sociologie, les graphes représentent des individus liés par des relations sociales telles que l'amitié ou les relations de travail.

Une clique d'un graphe est un sous ensemble de sommets tous en relation deux à deux. Elle définit un sous graphe à n sommets et $n(n-1)/2$ arêtes. Plus particulièrement on s'intéresse aux cliques maximales et maximum qui contiennent tous les individus en relation les uns avec les autres. 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 nous allons concentrer sur la recherche de cliques maximales (MCE : Maximal Clique Exploration) et de cliques maximum dans un graphe.

Nous allons présenter l'algorithme de Bron-Kerbosh l'un des algorithmes les plus utilisés dans la théorie des graphes pour l'énumération de toutes les cliques maximales par la suite nous allons essayer d'adapter cet algorithme pour être s'exécute sous Giraph. Concernant l'énumération des cliques maximum nous allons présenter et adapter Bron-Kerbosh et une heuristique simple sous Giraph.

3.2 Problème d'énumération des cliques

3.2.1 Définition de la clique maximale

Une clique est un graphe tel que pour toute paire de sommets x, y l'arête $\{x, y\}$ existe. On parle aussi de graphe complet. Elle dite maximal lorsqu'elle est le plus grand sous graphe complet contenant tous ses nœuds [23]. Prenant par exemple le graphe de la figure 3.1. Dans ce graphe on trouve cinq cliques maximales $\{1, 2, 5\}$, $\{4, 5\}$, $\{2, 3\}$, $\{3, 4\}$, $\{4, 6\}$.

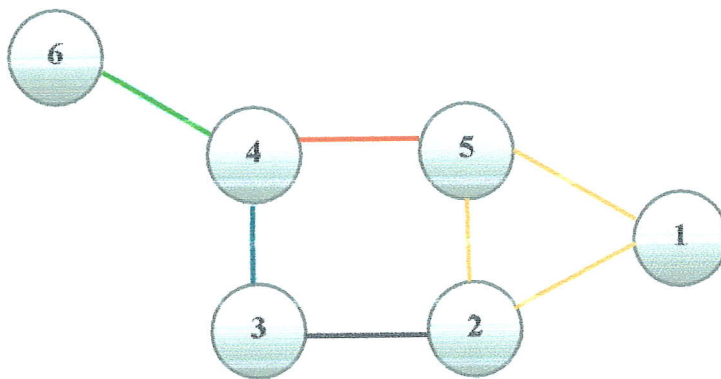


Figure3.1: Un graphe avec 5 cliques maximales.

3.2.2 Définition de la clique maximum

Le problème de la clique maximum (PCM) est un problème NP-Difficile consiste à déterminer un sous-ensemble de sommets S de cardinalité maximum dans un graphe $G=(V, E)$, tel que les sommets de S soient deux à deux adjacents. Prenant par exemple le graphe de la figure 3.2, dans ce graphe on trouve $\{4, 7, 6, 5\}$ comme clique maximum [15].

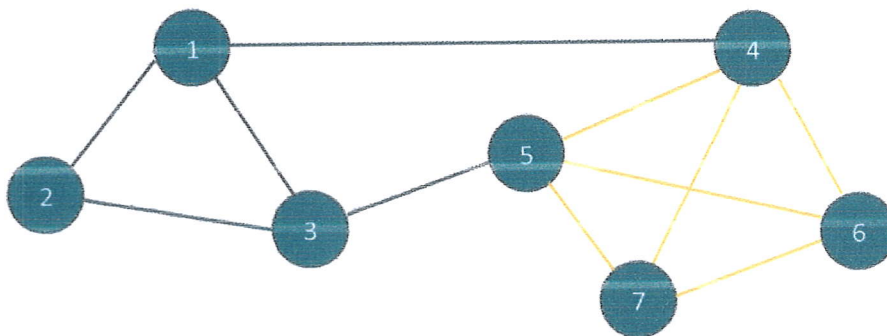


Figure3.2: Un graphe avec une clique maximum $\{4, 7, 6, 5\}$.

On doit distinguer entre la clique maximale et la clique maximum de sorte qu'une clique maximale n'est pas forcément une clique maximum. Maximale signifie qu'elle ne peut pas être agrandie. Maximum signifie qu'il n'en existe pas de plus grande.

3.2.3 Domaines d'applications des cliques

Le problème de la clique maximum (maximale) est un problème classique en optimisation combinatoire dans différents domaines :

- En Biologie, un réseau d'interactions protéine-protéine (IPP), spécifique d'une espèce, est un graphe dont les sommets sont des protéines et les arêtes leurs éventuelles

interactions, qui correspondent à des contacts physiques à un moment donné, dans un ou plusieurs tissus [27].

- En Sociologie, les graphes représentent des individus liés par des relations sociales, telles que l'amitié ou les relations de travail [27].
- En Informatique, le réseau Web consiste en un ensemble de pages connectées par des hyperliens.
- En Théorie de codage on cherche à trouver un code binaire le plus grande possible qui peut corriger certain nombre d'erreurs pour une taille donnée des mots binaires (vecteurs). Afin de corriger les erreurs, le code doit être composé de mots binaires parmi lesquels deux quelconques différents dans un certain nombre de positions de telle sorte qu'un mot erroné peut être détectée et corrigée. Le mot erroné est corrigé en le remplaçant par le mot du code [39].
- Dans la découverte de médicaments, il arrive souvent que plusieurs molécules de la structure apparemment sans rapport sont actives sur la même cible de drogue. Si l'on peut démontrer que les distances interatomiques entre un sous-ensemble d'atomes dans un médicament correspondent aux distances entre un sous-ensemble de taille similaire dans un second médicament, on peut alors postuler une commune pharmacophore qui représente l'interaction entre les deux médicaments avec la cible. Pour les médicaments, les sommets sont des atomes généralement, et le poids d'arête sont des distances interatomiques [28].

3.2.4 Algorithmes d'énumération des cliques

Il existe plusieurs algorithmes qui maintiennent la recherche des cliques maximales telles que Tomita, Tanaka, Takahashi et Bron-Kerbosch [16].

L'algorithme de Bron-Kerbosch a été décrit en 1973 par deux chercheurs hollandais, Joep Kerbosch et Coenraad Bron. Leur algorithme est largement utilisé pour trouver toutes les cliques maximales dans un graphe [12]. C'est un algorithme de retour arrière récursif qui est facile à comprendre, facile à coder, et a été montré pour bien fonctionner dans la pratique.

L'algorithme de Bron-Kerbosch, illustré par la figure 3.3 s'appuie sur trois ensembles disjoints des sommets R , P et X en tant qu'arguments, où R est un (éventuellement non-maximale) clique et $P \cup X = \Gamma(R)$ sont les sommets qui sont adjacentes à chaque sommet dans R . Les sommets P seront considéré être ajouté à la clique R , tandis que ceux de X doivent être

exclus de la clique. Ainsi, dans l'appel récursif, l'algorithme répertorie toutes les cliques dans $P \cup R$ qui sont maximale dans le sous-graphe induit par $P \cup R \cup X$.

L'algorithme choisit un candidat v dans P pour ajouter à la clique R , et fait un appel récursif dans lequel v a été déplacé de R à P . Dans cet appel récursif, elle restreint X aux voisins de v , puisque les non-voisins ne peuvent pas affecter la maximalité des cliques résultant. Lorsque les retours d'appels récursifs, v est déplacé vers X pour éliminer les tâches redondantes par d'autres appels à l'algorithme. Quand la récursivité atteint un niveau auquel P et X sont vides, R est une clique maximale et rapporté. Pour lister toutes les cliques maximales du graphe, cet algorithme récursif est appelé avec P égal à l'ensemble de tous les sommets du graphe et avec R et X vide.

Bron et Kerbosch décrivent également une heuristique appelée pivotement, ce qui limite le nombre des appels récursifs effectués par leur algorithme (voir la figure 3.4).

L'observation clé est que pour tout sommet u dans $P \cup X$, appelé un pivot, toute clique maximale doit contenir l'un des u de non-voisins. Par conséquent, nous retardons les sommets dans $P \cap \Gamma(u)$ à partir de étant ajouté à la clique jusqu'à ce que les appels récursifs à venir, avec l'avantage que nous faisons moins appels récursifs [12].

```
proc BronKerbosch( $P, R, X$ )
1: if  $P \cup X = \emptyset$  then
2:   report  $R$  as a maximal clique
3: end if
4: for each vertex  $v \in P$  do
5:   BronKerbosch( $P \cap \Gamma(v), R \cup \{v\}, X \cap \Gamma(v)$ )
6:    $P \leftarrow P \setminus \{v\}$ 
7:    $X \leftarrow X \cup \{v\}$ 
8: end for
```

Figure 3.3: Algorithme Bron-Kerbosch sans pivot [16].


```

Bron-Kerbosch Algorithm (Version 2, with Pivot)
R = {}
P = (V)
X = {}

proc BronKerbosch(P, R, X)
1.  if P ∩ X = {} then
2.      R is a maximal clique
3.  end if
4.  choose the pivot vertex u in P ∩ X as the vertex with the highest number of neighbors in P
5.  for each vertex v in P \ nbrs(u) do
6.      BronKerbosch(P ∩ nbrs(v), R ∪ (v), X ∩ nbrs(v))
7.      P = P \ v
8.      X = X ∪ v
9.  end for

```

Figure3.4: Algorithme Bron-Kerbosch avec pivot [16].

Exemple :

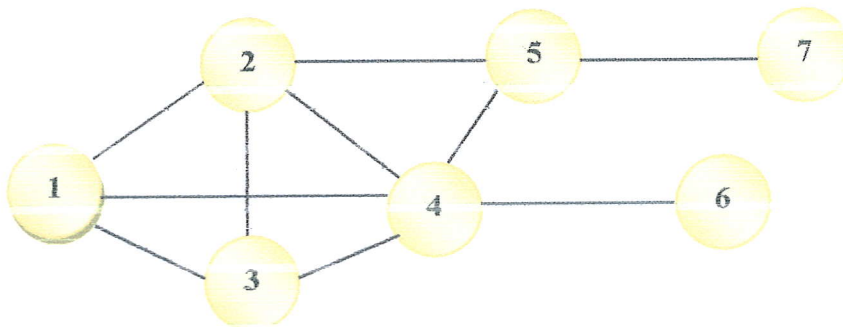


Figure3.5: Exemple de graphe avec 4 cliques maximales.

Appliquant l'algorithme de Bron-Kerbosch sur l'exemple de la figure 3.5 on obtient ce qui suit :

➤ Avec Pivot :

$R = \{\}$	$P = \{1, 2, 3, 4, 5, 6, 7\}$	$X = \{\}$	
Le pivot est 4 :			
$R = \{4\}$	$P = \{1, 2, 3, 5, 6\}$	$X = \{\}$	
Le pivot est 2 :			
$R = \{4, 2\}$	$P = \{1, 3, 5\}$	$X = \{\}$	
Le pivot est 5 :			
$R = \{4, 2, 1\}$	$P = \{3\}$	$X = \{\}$	
Le pivot est 3 :			
$R = \{4, 2, 1, 3\}$	$P = \{\}$	$X = \{\}$	4, 2, 1, 3 est une clique maximale.
$R = \{4, 2, 3\}$	$P = \{\}$	$X = \{1\}$	
Le pivot est 1 :			
$R = \{4, 2, 5\}$	$P = \{\}$	$X = \{\}$	4, 2, 5 est une clique maximale.
$R = \{4, 6\}$	$P = \{\}$	$X = \{\}$	4, 6 est une clique maximale.
$R = \{7\}$	$P = \{5\}$	$X = \{\}$	
Le pivot est 5 :			
$R = \{7, 5\}$	$P = \{\}$	$X = \{\}$	7, 5 est une clique maximale.

➤ Sans pivot:

$R = \{\}$	$P = \{1, 2, 3, 4, 5, 6, 7\}$	$X = \{\}$	
$R = \{1\}$	$P = \{2, 3, 4\}$	$X = \{\}$	
$R = \{1, 2\}$	$P = \{3, 4\}$	$X = \{\}$	
$R = \{1, 2, 3\}$	$P = \{4\}$	$X = \{\}$	
$R = \{1, 2, 3, 4\}$	$P = \{\}$	$X = \{\}$	1, 2, 3, 4 est une clique maximale.
$R = \{1, 2, 4\}$	$P = \{\}$	$X = \{3\}$	
$R = \{1, 3\}$	$P = \{4\}$	$X = \{2\}$	
$R = \{1, 3, 4\}$	$P = \{\}$	$X = \{2\}$	
$R = \{1, 4\}$	$P = \{\}$	$X = \{2, 3\}$	
$R = \{2\}$	$P = \{3, 4, 5\}$	$X = \{1\}$	
$R = \{2, 3\}$	$P = \{4\}$	$X = \{1\}$	
$R = \{2, 3, 4\}$	$P = \{\}$	$X = \{1\}$	
$R = \{2, 4\}$	$P = \{5\}$	$X = \{3, 1\}$	
$R = \{2, 4, 5\}$	$P = \{\}$	$X = \{\}$	2, 4, 5 est une clique maximale.
$R = \{2, 5\}$	$P = \{\}$	$X = \{4\}$	
$R = \{3\}$	$P = \{4\}$	$X = \{1, 2\}$	
$R = \{3, 4\}$	$P = \{\}$	$X = \{1, 2\}$	
$R = \{4\}$	$P = \{5, 6\}$	$X = \{1, 2, 3\}$	
$R = \{4, 5\}$	$P = \{6\}$	$X = \{2\}$	
$R = \{4, 6\}$	$P = \{\}$	$X = \{\}$	4, 6 est une clique maximale.
$R = \{5\}$	$P = \{7\}$	$X = \{2, 4\}$	
$R = \{5, 7\}$	$P = \{\}$	$X = \{\}$	5, 7 est une clique maximale.
$R = \{6\}$	$P = \{\}$	$X = \{4\}$	
$R = \{7\}$	$P = \{\}$	$X = \{5\}$	

3.2.5 Proposition des nouveaux algorithmes d'énumération des cliques sous Giraph

Dans cette section nous allons proposer un algorithme parallèle adapté à l'algorithme choisi précédemment (algorithme de Bron Kerbosh) pour l'environnement Giraph nommé EPCM (Énumération Parallèle de Clique Maximale).

Nous avons vu que le calcul sous Giraph se réalise en plusieurs Superstep. Dans chaque Superstep chaque sommet exécute une fonction utilisateur compute(). Pour adapter l'algorithme Bron Kerbosh en Giraph nous avons défini la fonction compute (), illustré par la figure 3.6 qui se déroule comme suit :

- Dans le Superstep 0 chaque vertex prend son identifiant et le met dans R avec la liste des voisins qui représente P et l'envoient à tous ses voisins.
- Dans le Superstep 1 nous allons utiliser trois ensembles R, P, X pour traiter les messages reçus avec un ensemble voisins qui contient les voisins de vertex. Pour chaque message reçu on extrait le R et P, ensuite on modifie P par l'intersection entre P et voisins. Si P est vide on teste si R n'existe pas dans X alors R est une clique maximale et on l'ajoute dans X. Si P n'est pas vide on envoie P+R à tous les éléments de P.

```

function Compute (vertex, messages) {
  if (getSuperstep()==0){
    R= vertex.getId (); P= Γ (vertex);
    SendMessageToAllEdges (P, R+P);
  } elif{
    for (message: messages) {
      extraire R , P;
      R.add (vertex.getId ());
      P=P ∩ Γ (vertex);
      If (P= {}) {
        if (R∉ X){
          vertex.setValue(R);
          X=X ∪ R;
        }
      } elif {
        sendMessage (P, R+P) ;
      }
    }
  }
  vertex.voteToHalt();
}

```

Figure3.6: Pseudo code de l'algorithme EPCM.

Exemple : Appliquant l'algorithme de la figure 3.6 sur l'exemple suivant (figure 3.7).

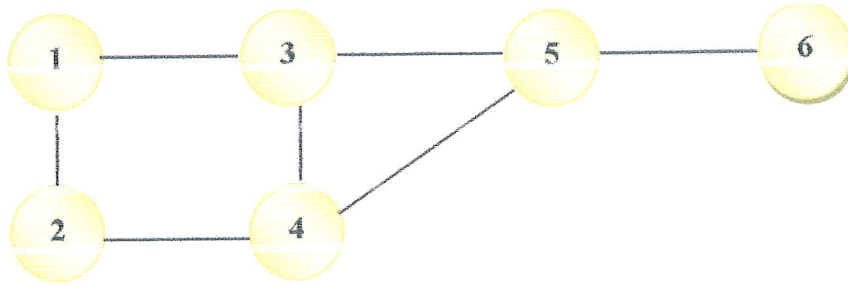
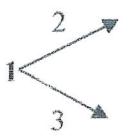
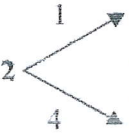

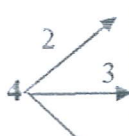
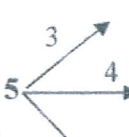


Figure 3.7: Exemple avec 5 cliques maximales.

Superstep0	Superstep1	Superstep2
$R = \{1\}$ $P = \{2,3\}$	$R = \{1,2\} P = \{ \}$ $R = \{1,2\}$ une clique maximale. $R = \{1,3\} P = \{ \}$ $R = \{1,3\}$ une clique maximale.	
$R = \{2\}$ $P = \{1,4\}$	$R = \{2,1\} P = \{ \}$ $R = \{2,1\}$ une clique maximale. $R = \{2,4\} P = \{ \}$ $R = \{2,4\}$ une clique maximale.	
$R = \{3\}$ $P = \{1,4,5\}$	$R = \{3,1\} P = \{ \}$ $R = \{3,1\}$ une clique maximale. $R = \{3,4\} P = \{5\} \xrightarrow{5}$ $R = \{3,5\} P = \{4\} \xrightarrow{4}$	$R = \{3,4,5\} P = \{ \}$ $R = \{3,4,5\}$ une clique maximale. $R = \{3,5,4\} P = \{ \}$ $R = \{3,5,4\}$ une clique maximale.
$R = \{4\}$ $P = \{2,3,5\}$	$R = \{4,2\} P = \{ \}$ $R = \{4,2\}$ une clique maximale. $R = \{4,3\} P = \{5\} \xrightarrow{5}$ $R = \{4,5\} P = \{3\} \xrightarrow{3}$	$R = \{4,3,5\} P = \{ \}$ $R = \{4,3,5\}$ une clique maximale. $R = \{4,5,3\} P = \{ \}$ $R = \{4,5,3\}$ une clique maximale.
$R = \{5\}$ $P = \{3,4,6\}$	$R = \{5,3\} P = \{4\} \xrightarrow{4}$ $R = \{5,4\} P = \{3\} \xrightarrow{3}$ $R = \{5,6\} P = \{ \}$ $R = \{5,6\}$ une clique maximale.	$R = \{5,3,4\} P = \{ \}$ $R = \{5,3,4\}$ une clique maximale. $R = \{5,4,3\} P = \{ \}$ $R = \{5,4,3\}$ une clique maximale.
$R = \{6\}$ $P = \{5\}$	$R = \{6,5\} P = \{ \}$ $R = \{6,5\}$ une clique maximale.	

Nou remarque qu'il y a des duplications des cliques maximales ce qui augmente la quantité des E/S et le nombre de messages. Pour cette raison, nous pouvons réaliser une optimisation de l'algorithme, de sorte que dans le Superstep 0, le sommet ayant l'identifiant le plus max par rapport à ses voisins n'envoie pas des messages et lui seulement qui signale la clique maximale trouvé. Pour bien illustrer cette optimisation, ci-dessous les résultats de déroulement de cette optimisation sur le même graphe de la figure 3.7.

Superstep0	Superstep1	Superstep2
$R = \{1\}$ $P = \{2,3\}$ 	$R = \{1,2\} P = \{ \}$ $R = \{1,3\} P = \{ \}$	$R = \{1, 2\}$ une clique maximale. $R = \{1, 3\}$ une clique maximale.
$R = \{2\}$ $P = \{1,4\}$ 	$R = \{2,1\} P = \{ \}$ $R = \{2,4\} P = \{ \}$	$R = \{2, 4\}$ une clique maximale.
$R = \{3\}$ $P = \{1, 4,5\}$ 	$R = \{3,1\} P = \{ \}$ $R = \{3,4\} P = \{5\}$ $R = \{3,5\} P = \{4\}$	$R = \{3, 4,5\} P = \{ \}$ $R = \{3, 4,5\}$ une clique maximale. $R = \{3, 5,4\} P = \{ \}$
$R = \{4\}$ $P = \{2, 3,5\}$ 	$R = \{4,2\} P = \{ \}$ $R = \{4,3\} P = \{5\}$ $R = \{4,5\} P = \{3\}$	$R = \{4, 3,5\} P = \{ \}$ $R = \{4, 5,3\} P = \{ \}$
$R = \{5\}$ $P = \{3, 4,6\}$ 	$R = \{5,3\} P = \{4\}$ $R = \{5,4\} P = \{3\}$ $R = \{5,6\} P = \{ \}$	$R = \{5, 3,4\} P = \{ \}$ $R = \{5, 4,3\} P = \{ \}$ $R = \{5, 6\}$ une clique maximale.

On remarque maintenant que les cliques maximales sont sans duplication et le sommet ayant l'Identifiant le plus max dans R qui signale la clique maximale. On peut remarquer aussi que le nombre de messages est réduis ainsi que le nombre de sommets impliquées par le

calcul dans chaque Superstep. Par conséquent, cela peut influencer de manière positive le temps d'exécution de l'algorithme.

Pour minimiser le nombre des Supersteps nécessaires pour l'énumération des cliques sous Giraph qui implique la réduction du cout d'exécution en termes des nombres des messages nécessaires, nous allons proposer un autre algorithme adapté à l'algorithme choisi précédemment (algorithme de Bron Kerbosh) nommé BKP-AP (Bron Kerbosh Parallèle Avec Pivot). Cet algorithme s'exécute seulement en 2 Supersteps. Pour le faire nous avons gardé le même principe des appels récursifs de l'algorithme de Bron-Kerbosh.

La méthode Compute () de ce nouvel algorithme est illustrée par la figure 3.8 et se déroule comme suit :

- Dans le Superstep 0 chaque vertex envoie l'identifiant avec la liste des voisins à tous ses voisins.
- Dans le Superstep 1 chaque vertex construit son propre ensemble P de sorte que P ne contient que l'identifiant de chaque vertex et ses voisins et leurs voisins en commun. Par la suite chaque vertex exécute le même algorithme Bron-KerboshAvecPivot ou Bron-KerboshSansPivot nommé BKP-SP (Bron Kerbosh Parallèle Sans Pivot) présenté précédemment, respectivement par (voir la figure 3.4) sur le sous graphe local obtenu.



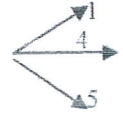
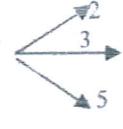
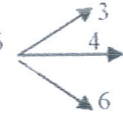

```

function Compute (vertex, messages){
    if (getSuperstep()==0){
        S= vertex.getId () U {Γ (vertex)};
        sendMessageToAllEdges(vertex, S);
    } elif (getSuperstep () =1) {
        R= {}, P= Γ (vertex), X= {};
        for (message: messages) {
            G=G U {message.Id, (message. Γ (vertex) ∩ P)};
        }
        P=All Vertices of G;
        if (P= {} && X= {}) {
            R=R U {vertex.getId ()};
        } elif {
            Bron_KerboshWithPivot(R, P, X);
        }
        Vertex.setValue(R);
    }
}

```

Figure3.8: Pseudo code de BKP-AP.

Exemple : Appliquant l'algorithme BK-AP illustré par la figure 3.8 sur le graphe de la figure 3.7 on obtient les résultats suivants :

Superstep0	Superstep1
<p>Vertex: 1</p> <p>S=123</p> 	<p>$R = \{\}, P = \{2, 3\}, X = \{\}$ Bron-Kerbosh ($\{\}, \{2, 3\}, \{\}$); $R = \{1\}, P = \{2, 3\}, X = \{\}$ Bron-Kerbosh ($\{1\}, \{2, 3\}, \{\}$); $R = \{1, 3\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{1, 3\}, \{\}, \{\}$); $R = \{1, 3\}$ une clique maximale. $R = \{1, 2\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{1, 2\}, \{\}, \{\}$); $R = \{1, 2\}$ une clique maximale.</p>
<p>Vertex: 2</p> <p>S=214</p> 	<p>$R = \{\}, P = \{1, 4\}, X = \{\}$ Bron-Kerbosh ($\{\}, \{1, 4\}, \{\}$); $R = \{2\}, P = \{1, 4\}, X = \{\}$ Bron-Kerbosh ($\{2\}, \{1, 4\}, \{\}$); $R = \{2, 4\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{2, 4\}, \{\}, \{\}$); $R = \{2, 4\}$ une clique maximale. $R = \{2, 1\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{2, 1\}, \{\}, \{\}$); $R = \{2, 1\}$ une clique maximale.</p>
<p>Vertex: 3</p> <p>S=3145</p> 	<p>$R = \{\}, P = \{1, 4, 5\}, X = \{\}$ Bron-Kerbosh ($\{\}, \{1, 4, 5\}, \{\}$); $R = \{3\}, P = \{1, 4, 5\}, X = \{\}$ Bron-Kerbosh ($\{3\}, \{1, 4, 5\}, \{\}$); $R = \{3, 4\}, P = \{5\}, X = \{\}$ Bron-Kerbosh ($\{3, 4\}, \{5\}, \{\}$); $R = \{3, 4, 5\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{3, 4, 5\}, \{\}, \{\}$); $R = \{3, 4, 5\}$ une clique maximale. $R = \{3, 1\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{3, 1\}, \{\}, \{\}$); $R = \{3, 1\}$ une clique maximale.</p>
<p>Vertex: 4</p> <p>S=4235</p> 	<p>$R = \{\}, P = \{2, 3, 5\}, X = \{\}$ Bron-Kerbosh ($\{\}, \{2, 3, 5\}, \{\}$); $R = \{4, 5\}, P = \{3\}, X = \{\}$ Bron-Kerbosh ($\{4, 5\}, \{3\}, \{\}$); $R = \{4, 5, 3\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{4, 5, 3\}, \{\}, \{\}$); $R = \{4, 5, 3\}$ une clique maximale $R = \{4\}, P = \{2, 5\}, X = \{\}$ Bron-Kerbosh ($\{4, 2\}, \{\}, \{\}$); $R = \{4, 2\}$ une clique maximale.</p>
<p>Vertex: 5</p> <p>S=5346</p> 	<p>$R = \{\}, P = \{3, 4, 6\}, X = \{\}$ Bron-Kerbosh ($\{\}, \{3, 4, 6\}, \{\}$); $R = \{5, 4\}, P = \{3\}, X = \{\}$ Bron-Kerbosh ($\{5, 4\}, \{3\}, \{\}$); $R = \{5, 4, 3\}, P = \{\}, X = \{\}$ Bron-Kerbosh ($\{5, 4, 3\}, \{\}, \{\}$); $R = \{5, 4, 3\}$ une clique maximale. $R = \{5\}, P = \{4, 6\}, X = \{\}$ Bron-Kerbosh ($\{5, 6\}, \{\}, \{\}$); $R = \{5, 6\}$ une clique maximale.</p>
<p>Vertex: 6</p> <p>S=6</p> 	<p>$R = \{\}, P = \{5\}, X = \{\}$ Bron-Kerbosh ($\{\}, \{5\}, \{\}$); $R = \{6\}, P = \{5\}, X = \{\}$ Bron-Kerbosh ($\{6, 5\}, \{\}, \{\}$); $R = \{6, 5\}$ une clique maximale.</p>

Pour éliminer les duplications des cliques maximales, seulement le vertex ayant l'identifiant le plus max dans R qui signale la clique maximale qu'il appartient.

Remarque : nous pouvons étendre cet algorithme pour énumérer les cliques maximums. Pour ce faire, nous avons ajouté à la méthode compute () illustrée par la figure 3.8 ce qui suit :

- Dans le Superstep 2, chaque vertex calcule la cardinalité de la clique maximale trouvé et l'envoie à tous ses voisins.
- Dans le Superstep supérieur à 2, chaque vertex compare la cardinalité reçu avec la cardinalité locale, si elle est supérieure de sa cardinalité locale, il envoie la nouvelle cardinalité à ses voisins.

3.3 Algorithme glouton pour le problème de clique maximum

Un algorithme glouton fait toujours un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale. La figure 3.7 présente le pseudocode d'un algorithme séquentiel glouton [9] proposé pour résoudre le problème de la clique maximum dans un graphe $G=(S, A)$.

```

Début
  C ← φ
  Cand ← S
  Tant que Cand ≠ φ Faire
    Soit Sj le sommet de Cand ayant le plus fort degré dans le sous-graphe induit par
    Cand
    C ← C ∪ {Sj}
    Cand ← Cand ∩ {Sj / (Si, Sj)}
  FinTanque
Fin

```

Figure3.9: Pseudo code d'algorithme Glouton.

Exemple : On applique l'algorithme Glouton sur le graphe de (la figure 3.2) et on obtient les résultats suivants qui sont illustré par la figure 3.10 :

- **Initialement:** $C = \{\}$ et $Cand = \{1, 2, 3, 4, 5, 6, 7\}$.
- **Itération 1:** 4 et 5 ayant le plus fort degré, on choisit le sommet 5 comme le sommet de "Cand" de plus grand degré, par conséquent $C = \{5\}$ et $Cand = \{1, 2, 3, 4, 5, 6, 7\} \cap \{3, 4, 7, 6\} \Rightarrow Cand = \{3, 4, 7, 6\}$.
- **Itération 2:** 4 est le sommet de plus fort degré, par conséquent $C = \{5, 4\}$ et $Cand = \{7, 6\}$. (Voir la figure 3.10 a. Itération2).
- **Itération 3:** 7 est le sommet de plus fort degré, par conséquent $C = \{5, 4, 7\}$ et $Cand = \{6\}$. (Voir la figure 3.10 b. Itération3).
- **Itération 4:** $C = \{5, 4, 7, 6\}$ et $Cand = \{\}$. (Voir la figure 3.10c. Itération4).

Donc la clique maximum est : $C = \{5, 4, 7, 6\}$.

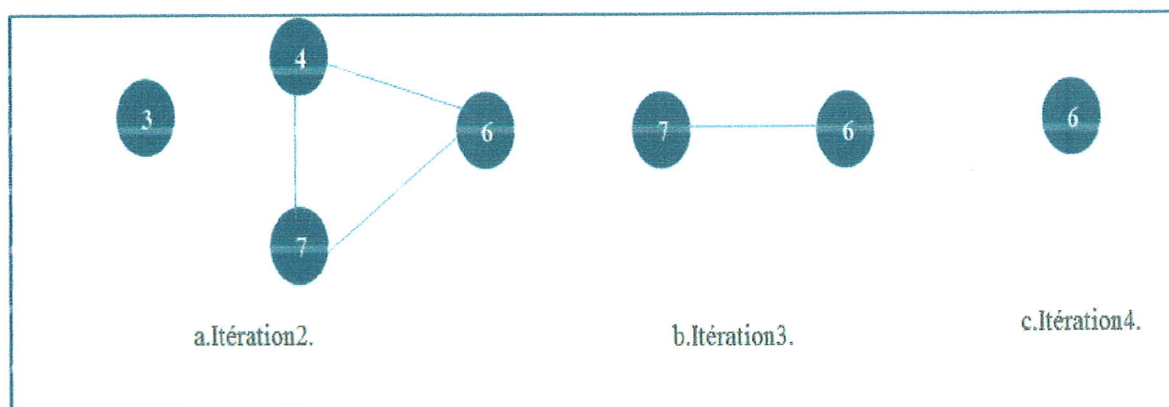


Figure3.10: Déroulement de l'algorithme Glouton sur le graphe de la figure 3.2.

3.3.1 Algorithme Glouton d'énumération de cliques maximum sous Giraph

Dans cette section nous avons proposé un algorithme parallèle adapté à l'algorithme choisi précédemment (algorithme Glouton) pour l'environnement Giraph nommé HPECM (Heuristique Parallèle d'Énumération des Cliques Maximum), qui calcule la clique maximum dans le graphe complet avec sa cardinalité. Par conséquent nous avons défini la méthode `compute()` illustré par la figure 3.11 qui se déroule comme suit :

- Dans le Superstep 0 chaque sommet prend son identifiant et la liste des voisins et l'envoient à tous ses voisins.
- Dans le Superstep 1 et pour le traitement de chaque message reçu nous allons utiliser un hashTable qui prend l'identifiant comme clé avec l'intersection des voisins de cet identifiant avec les voisins de ce vertex. Par la suite on utilise un ensemble Candidat

qui contient tous les voisins avec un ensemble CliqMax pour afficher la clique maximum et on fait un appel à la procédure Clique maximum (Voir la figure 3.12) qui calcul la clique maximum pour chaque vertex, ensuite le vertex calcul la cardinalité de la clique maximum qui l'appartient et l'envoie à tous ses voisins.

- Dans le superstep supérieur à 1 chaque vertex compare la cardinalité de la clique maximum reçu avec sa cardinalité local, lorsqu'elle est supérieure à la cardinalité local, il envoie la nouvelle cardinalité à ses voisins.

```

function Compute(vertex,messages){
    if(getSuperstep()==0){
        S= {vertex.getId ()} U {Γ (vertex)};
        sendMessageToAllEdges(vertex,S);
    } elif (getSuperstep()==1){
        Cand = {}, G = {};
        CliqMax=CliqMax U {(vertex.getId ())};
        Cand= {Γ (vertex)};
        for (message: messages) {
            G=G U {message.Id, (message.Γ (vertex) ∩ P)};
            Cliqmaximum (Cand, G, CliqMax);
            vertex.setValue (CliqMax);
            Change=false;
            Card=CliqMax.Size ();
            SendMessageToAllEdges (vertex, Card);
        } elif (getSuperstep ()>1) {
            for (message : messages){
                if (Card<message){
                    Card=message;
                    Change=true;
                }
            }
            if (Change=true){
                SendMessageToAllEdges (vertex, Card)
                vertex.setValue (+Card);
            }
        }
        vertex.voteToHalt ();
    }
}

```

Figure3. 11: Pseudo code de l'algorithme HPECM.

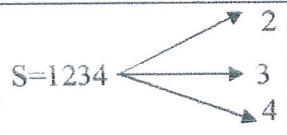
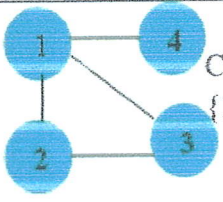
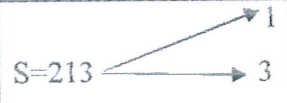
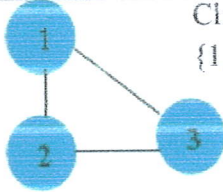
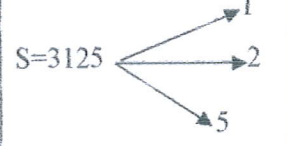
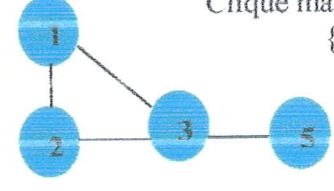
Le pseudo code de la fonction Cliqmaximum(Cand, G, CliqMax) est illustré par la figure 3.12 :

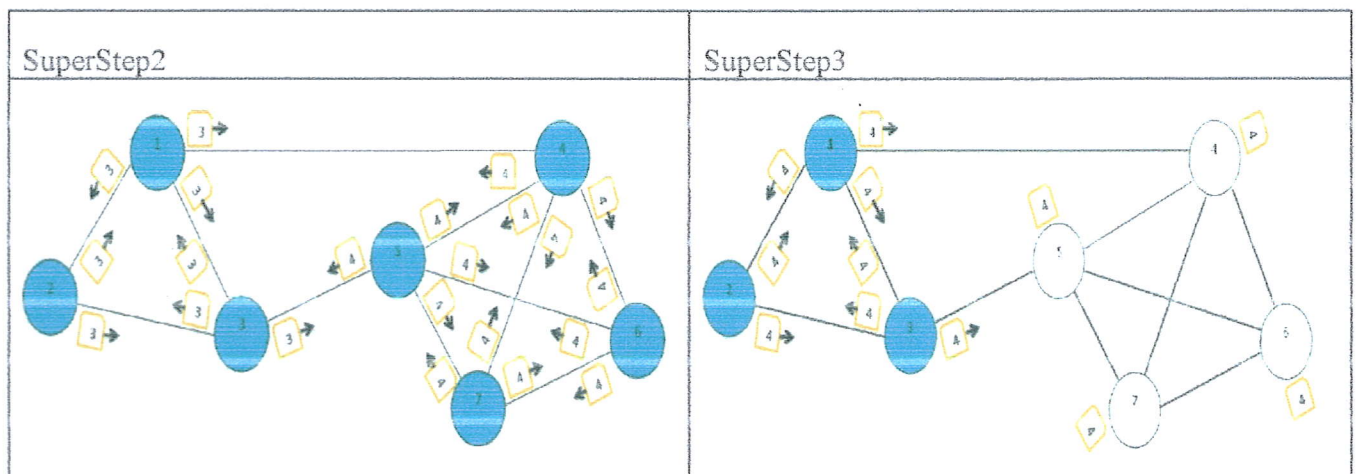
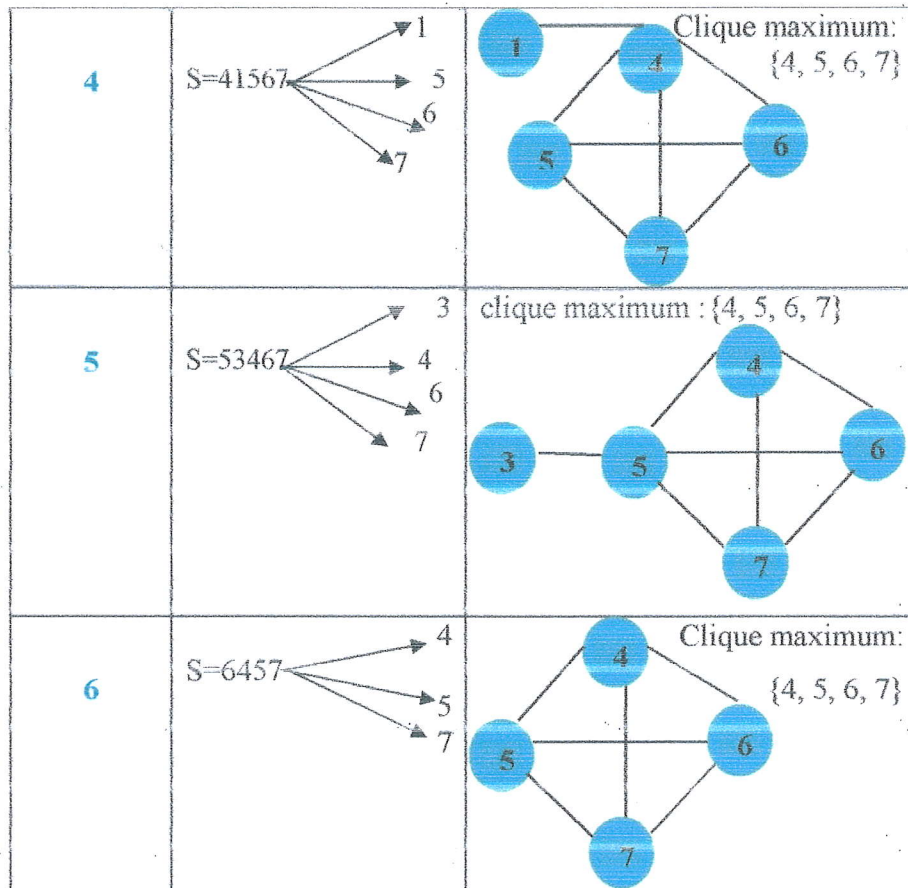
```

function Cliqmaximum (Cand,G,CliqMax){
    While(Cand ≠{})faire{
        Soit Sj le vertex ayant le plus fort degré dans G;
        CliqMax=CliqMax U {Sj};
        Cand=Cand / Sj;
        Cand= Cand ∩ {Γ (Sj)};
        G=G / Sj;
    }
}
    
```

Figure3.12: Pseudo code de la fonction qui calcule la clique maximum.

Exemple : Appliquant l'algorithme de la figure 3.11 sur le graphe de la figure 3.7.

Vertex	Superstep0	Superstep1
1	S=1234 	 <p>Clique maximum: {1,2,3}</p>
2	S=213 	 <p>Clique maximum: {1,2,3}</p>
3	S=3125 	 <p>Clique maximum: {1, 2, 3}</p>



3.4 Conclusion :

Dans ce chapitre nous avons proposé des algorithmes d'énumération des cliques sous Giraph, ces algorithmes sont basé sur des algorithmes séquentiels bien définis tels que Bron Kerbosh et Glouton. Dans le chapitre suivant nous allons implémenter ces algorithmes sous Giraph et les tester en réalisant des expérimentations.

Chapitre 4 :
Mise en œuvre et évaluation de
l'algorithme

4.1 Introduction :

Dans le chapitre précédent nous avons proposé des algorithmes parallèles pour le traitement des larges graphes sous Giraph. Dans ce chapitre nous allons implémenter et tester ces algorithmes sur des Benchmarks de données. Par la suite nous allons comparer ces algorithmes selon plusieurs critères. Finalement, nous allons tester ces algorithmes sous réseau, en créant un petit cluster de machines.

4.2 Environnement du travail :

D'abord nous avons choisi de travailler avec ubuntu 15.04 qui est un système d'exploitation Linux, à cause de la stabilité de Hadoop sur cette plateforme. Pour l'installation et la configuration de l'environnement du travail single node voir l'annexe B et pour la configuration multi-node voir l'annexe C. Concernant l'implémentation des algorithmes nous avons utilisé Java (jdk7).

Logiciel	Version	Logiciel pré-requis	Remarque
Système d'exploitation de base	Windows 7	/	64 bits
	Windows 8.1		
Outils de virtualisation	Oracle VM	Windows7	/
	VirtualBox	Windows8	
Système d'exploitation invité(Linux)	Ubuntu 15.04	Oracle VM VirtualBox	Master (slave1), slave2, slave3 :64bits
Hadoop	1.2.1	Ubuntu, Java version7 (jdk7)	https://archive.apache.org/dist/hadoop/core/hadoop-1.2.1/hadoop-1.2.1.tar.gz
Giraph	1.2.0	Hadoop	https://github.com/apache/giraph.git

Tableau 4.1: Tableau descriptif de l'environnement.

		Pc_Meryem		Pc_Massika	
Machine physique	Processeur	Intel(R) Core(TM) i7-5500U CPU @ 2.40 GHZ		Intel(R) Core(TM) i3-3214U CPU @ 1.80 GHZ	
	RAM	8,00 Go		4,00 Go	
	Espace disque	930 Go		464 Go	
Machine Virtuelle	Nœud	Master (slavel)	Slave2	Slave3	Slave 4
	Processeur	2	2	2	2
	RAM	2Go	2Go	2Go	2Go
	Espace disque	20Go	20Go	20Go	20Go

Tableau 4. 2: Tableau descriptif des machines virtuelles et physique.

4.3 Benchmarks de données choisis:

Pour bien tester et évaluer les algorithmes implémentés, dans un premier temps nous avons incluse dans notre étude le benchmarks de données réelle Facebook (<http://snap.stanford.edu/data/>), qui est un graphe orienté et nous avons le transformer en un graphe non orienté. La description de ce graphe est présentée dans le tableau 4.3. Mais malheureusement, ces graphes nécessitent des capacités matérielles importantes notamment pour l'exécution des algorithmes exactes. Ce qui rend impossible de les exécuter sur nos PC. Par la suite nous avons généré¹ quatre types de graphes non orienté nommés respectivement : Sp-graphe 1, Sp-graphe 2, Sp-graphe 3, Sp-graphe 4 (voir le Tableau 4.4, Tableau 4.5, Tableau 4.6, Tableau 4.7). Ces quatre graphes sont des graphes creux (sparse graphe). A chaque fois en augmente le nombre des arêtes en augmentant le degré moyen et le degré max.

Pour vérifier si les algorithmes d'énumération des cliques maximums permettent une énumération optimale ou proche à l'optimal, nous avons arrivé à tester l'algorithme glouton (HPECM) sur le benchmark de données DIMACS². Les résultats obtenus sont illustré par le tableau 4.4. Ces résultats montrent que l'algorithme permet une énumération proche à la solution optimale.

Benchmark de données	Nombre de sommets	Nombre d'arêtes
Facebook	4039	88234

Tableau 4.3: Benchmarks de données utilisés (non orienté).

¹ <https://sites.google.com/site/santofortunato/inthepress2>

² http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark

Instance	W(g)	Meilleur Résultat	Résultat obtenu	Nombre de Vertex	Nombre d'arête
C125.9	34	34	33	125	6963
C250.9	44	44	39	250	27984
C500.9	≥ 57	57	49	500	112332
DSJC500_5	13	13	12	500	125248

Tableau 4.4 : Résultats d'exécution de HPECM sur quelques graphes de DIMACS.

Sp-graphe 1	Nbre sommets	Nbre Arêtes	Degré moyen	Degré max	Taille fichier
	100	236	2	3	1.05ko
	500	1148	2	3	6.61ko
	1000	2310	2	3	13.6ko
	1500	3446	2	3	22.4ko
	2000	4622	2	3	31.0ko
	2500	5726	2	3	39.4ko
	3000	6876	2	3	47.9ko
	3500	8022	2	3	56.4ko
	4000	9172	2	3	65.0ko
	4500	10312	2	3	73.5ko

Tableau 4.5: Les données de Sp-graphe 1.

Sp-graphe2	Nbre sommets	Nbre Arêtes	Degré moyen	Degré max	Taille fichier
	100	486	5	10	1.79ko
	500	2320	5	10	11.0ko
	1000	4634	5	10	22.5ko
	1500	6936	5	10	37.8ko
	2000	9380	5	10	52.9ko
	2500	11740	5	10	67.5ko
	3000	14064	5	10	81.9ko

	3500	16300	5	10	95.7ko
	4000	18692	5	10	110ko
	4500	21076	5	10	124ko

Tableau 4.6: Les données de Sp-graphe 2.

Sp-graphe 3	Nbre_sommets	Nbre_arêtes	Degré_moyen	Degré_max	Taille_fichier
	100	658	7	10	3ko
	500	3290	7	10	15ko
	1000	6674	7	10	31ko
	1500	10118	7	10	51ko
	2000	13288	7	10	70ko
	2500	16770	7	10	90ko
	3000	19980	7	10	109ko
	3500	23316	7	10	128ko
	4000	26744	7	10	148ko
4500	30318	7	10	286ko	

Tableau 4.7: Les données de Sp-graphe 3.

Sp-graphe 4	Nbre_sommets	Nbre_Arêtes	Degré_moyen	Degré_max	Taille_fichier
	100	1500	15	20	4.69ko
	500	7622	15	20	30.8ko
	1000	15092	15	20	62.4ko
	1500	22842	15	20	104ko
	2000	30178	15	20	144ko
	2500	37654	15	20	183ko
	3000	45568	15	20	225ko
	3500	53092	15	20	265ko

	4000	60452	15	20	304ko
	4500	67998	15	20	344ko

Tableau 4.8: Les données de Sp-graphe 4.

4.4 Implémentation et évaluation expérimentale

4.4.1 Format d'entrée/sortie

Avant de commencer l'exécution de nos algorithmes, nous avons défini le format d'entrée/sortie de données. Le format d'entrée est de la forme : " vertexId dest1 dest2... destn". Un exemple de ce format est illustré par la figure 4.1. Le format de sortie est de la forme : "vertexId valeur". Un exemple d'exécution de l'algorithme HPECM sur le graphe illustré dans la figure 3.7 est présenté par la figure 4.2.

```

File: /user/local/giraph/input/6c.txt
-----
Goto : /user/local/giraph/input go
-----
Go back to dir listing
Advanced view/download options
-----
1 2 3
2 1 4
3 1 4 5
4 2 3 5
5 3 4 6
6 5

```

Figure 4.1: Exemple de Format d'entrée de données de graphe de la figure 3.7.

```
File: /user/local/giraph/output10/6cmxmu/part-m-00001
Goto : /user/local/giraph/output10/6cmx go
Go back to dir listing
Advanced view/download options

6 # 5 6 # Clique maximal # 2 # max= 3
5 # 3 4 5 # Clique maximum # 3 #
2 # 2 4 # Clique maximal # 2 # max= 3
1 # 1 3 # Clique maximal # 2 # max= 3
3 # 3 4 5 # Clique maximum # 3 #
4 # 3 4 5 # Clique maximum # 3 #
```

Figure 4.2: Résultats d'exécution d'algorithme HPECM.

Exemple : Soit le graphe de la figure 4.3. Le format d'entrée de données associé à ce graphe est illustré par la figure 4.4. Nous avons testé les différents algorithmes implémentés sur ce graphe et, comme exemple de format de sortie, les résultats obtenus de l'algorithme Op-EPCM sont illustré par la figure 4.5.

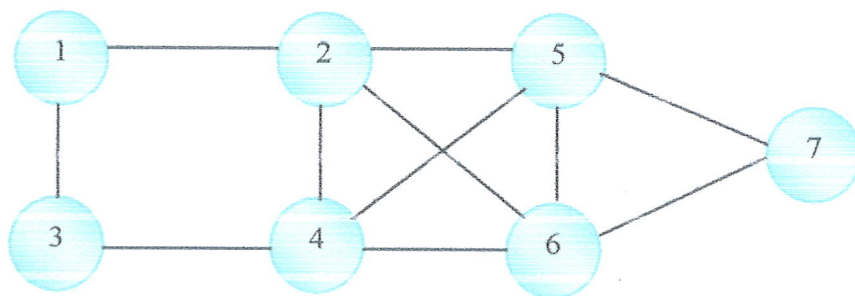


Figure 4.3: Graphe avec 5 cliques maximales.

```

File: /user/local/giraph/input/graph.txt


---


Goto : /user/local/giraph/input 


---


Go back to dir listing
Advanced view/download options


---


1 2 3
2 1 4 5 6
3 1 4
4 2 3 5 6
5 2 4 6 7
6 2 4 5 7
7 5 6

```

Figure 4.4: Format d'entrée de données de graphe de figure 4.3.

```

File: /user/local/giraph/output/graph/part-m-00001


---


Goto: /user/local/giraph/output/graph 


---


Go back to dir listing
Advanced view/download options


---


6 2 4 5 6 -1
5
7 5 6 7 -1
2 1 2 -1
1
3 1 3 -1
4 3 4 -1

```

Figure 4.5: Résultats d'exécution d'algorithme Op-EPCM sous Giraph sur le graphe de figure 4.3.

Remarque : Après l'exécution des algorithmes d'énumération des cliques maximales, nous avons remarqué que ces algorithmes permettent une énumération exacte de toutes les cliques maximales d'un graphe donné de manière efficace.

4.4.2 Expérimentation et analyse des résultats :

Dans cette section, nous allons présenter les différentes expérimentations réalisées. Les expérimentations Single node sont réalisées sur une machine virtuelle Ubuntu avec 2 processeurs et 2 GO de Mémoire.

Nous avons testé l'algorithme de Clique maximum (HPECM) sous Giraph. On utilisant le Benchmark de données illustré par le tableau 4.3. Les résultats du temps de CPU, le nombre des messages, le nombre des Supersteps obtenus sont illustrés respectivement par les Figures 4.6 et 4.7 et 4.8.

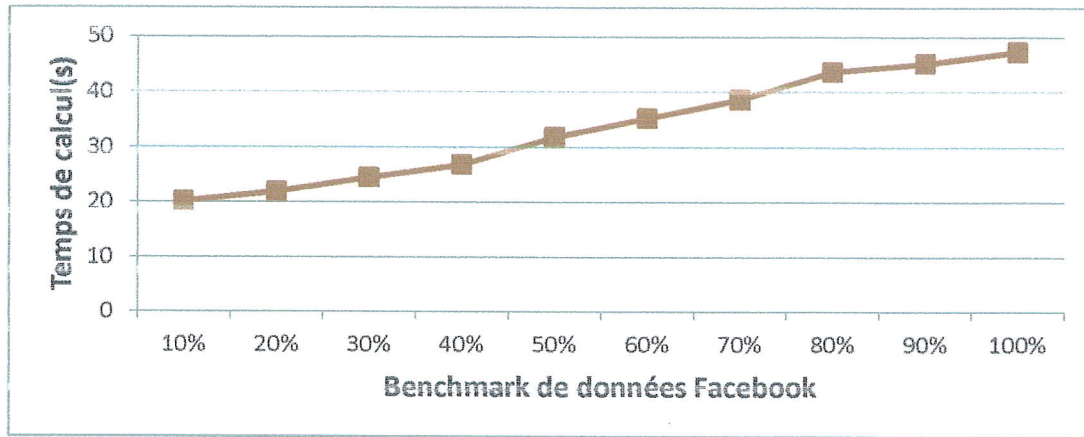


Figure 4.6: Résultats d'exécution d'algorithme HPECM sous Giraph en termes de temps de calcul.

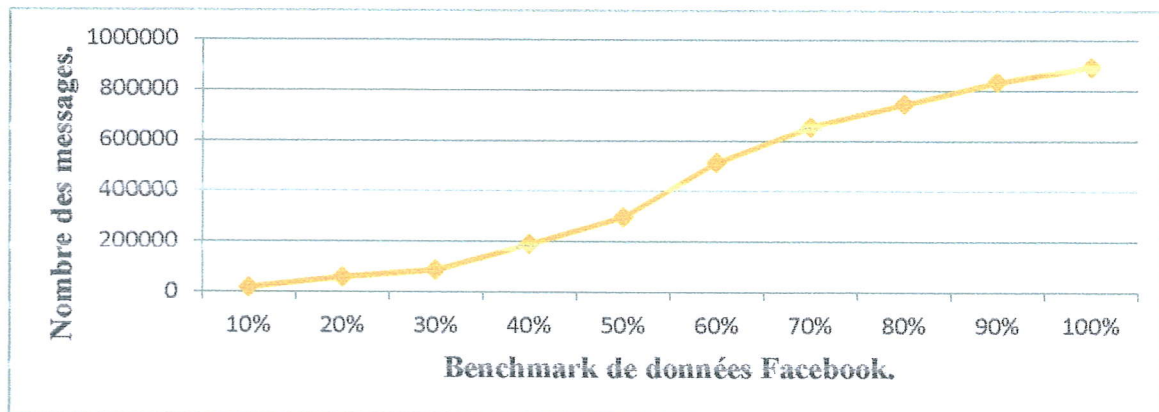


Figure 4.7: Résultats d'exécution d'algorithme HPECM sous Giraph en termes de message.

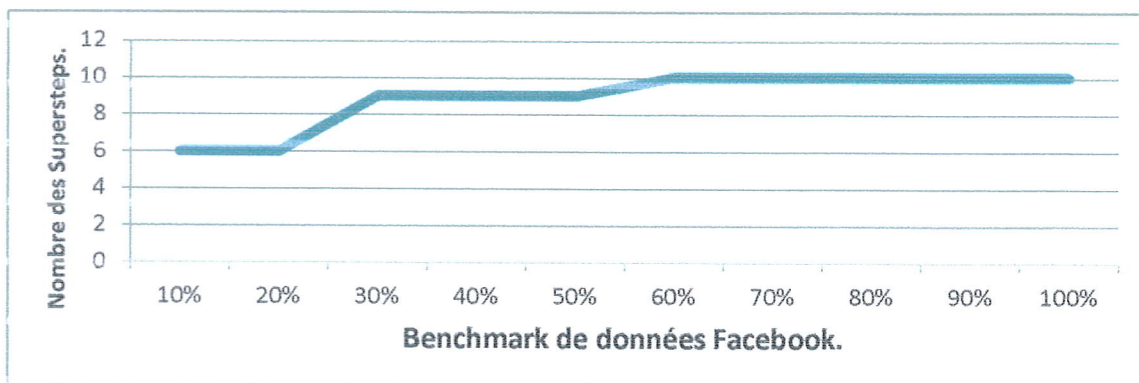


Figure 4.8: Résultats d'exécution d'algorithme HPECM sous Giraph en termes de Superstep

a) Test des algorithmes d'énumération des cliques maximales

Nous avons testé la scalabilité et la convergence des algorithmes d'énumération des cliques maximales (EPCM, Op-EPCM, BKP-AP, BKP-SP) sous Giraph. Pour ce faire nous avons utilisé les trois ensembles de données illustrées respectivement par les tableaux (tableau 4.4, tableau 4.5, tableau 4.6), chaque fois on augmente le nombre des sommets et le nombre des arêtes puis on examine le temps CPU et le nombre de message. Ces derniers augmentent d'une façon linéaire pour tous les algorithmes. Le nombre des Supersteps pour les deux algorithmes EPCM et Op-EPCM prend les mêmes valeurs de sorte que les Supersteps ont une relation avec le plus long chemin du graphe, l'algorithme BKP-AP et BKP-SP qui s'appuient sur la récursivité s'exécutent seulement en deux Supersteps. Les résultats obtenus illustrés respectivement par les figures 4.9, figure 4.10 et figure 4.11.

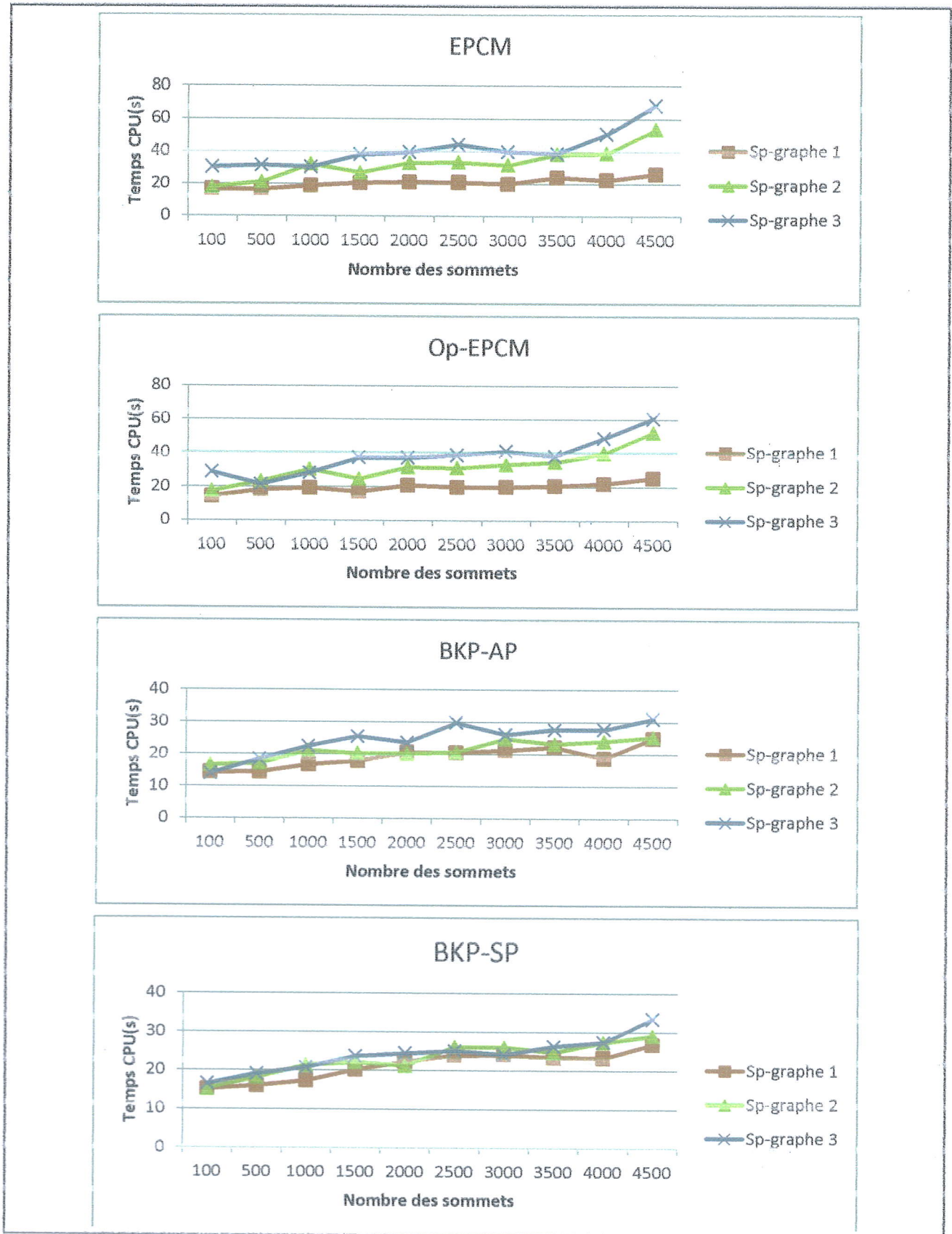


Figure 4. 9: Test de scalabilité des algorithmes d'énumération des cliques maximales en termes de temps CPU.

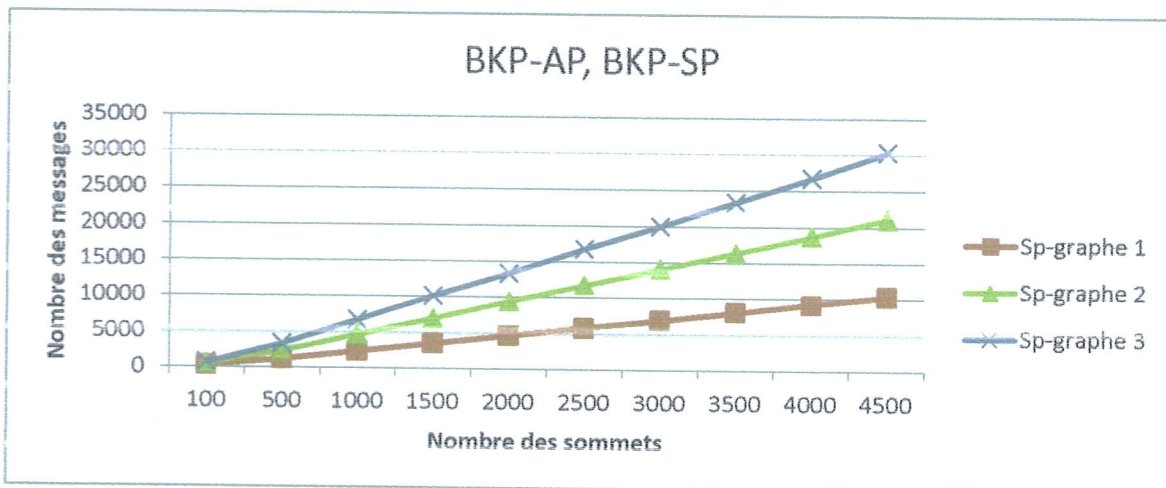
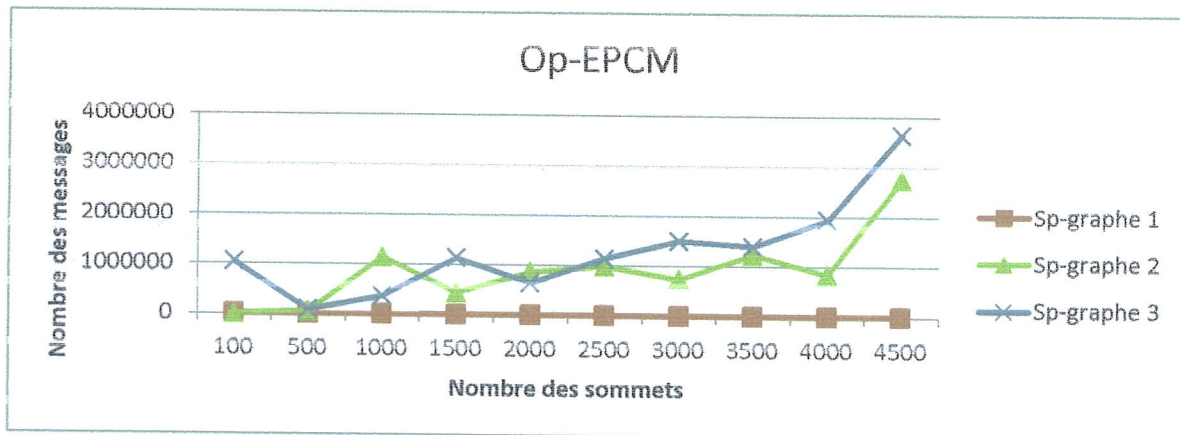
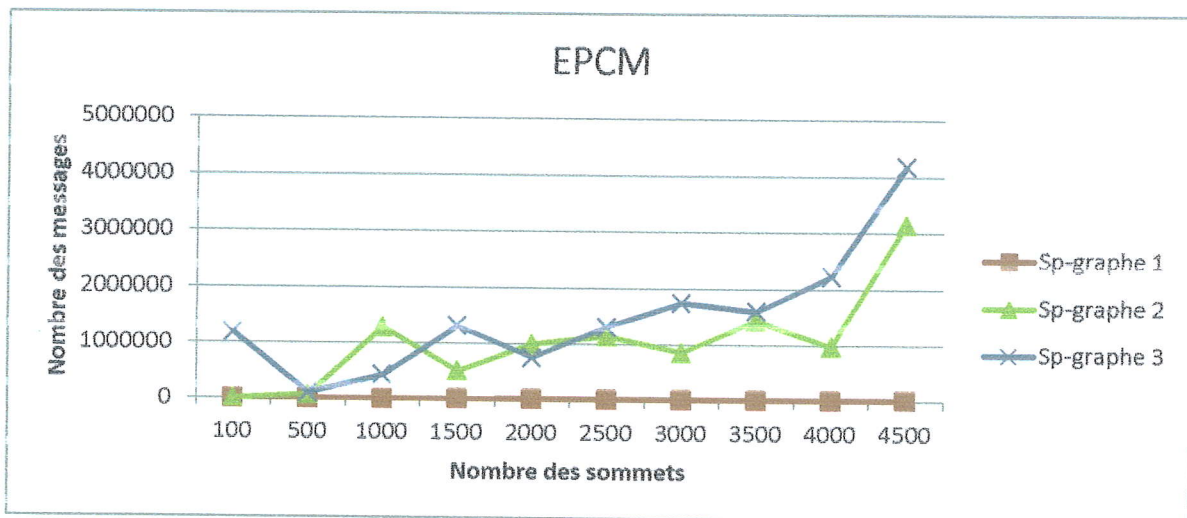


Figure 4.10: Test de scalabilité des algorithmes d'énumération des cliques maximales en termes de message.

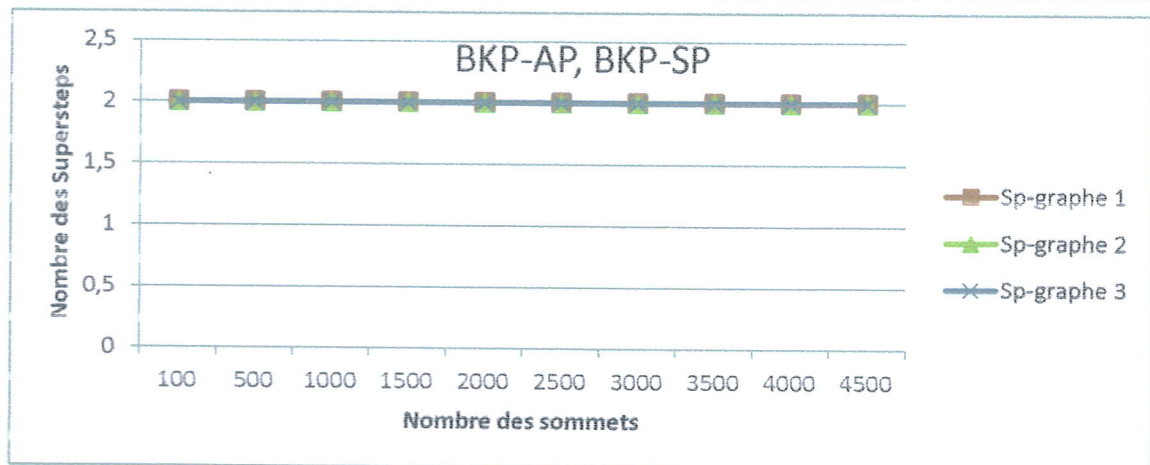
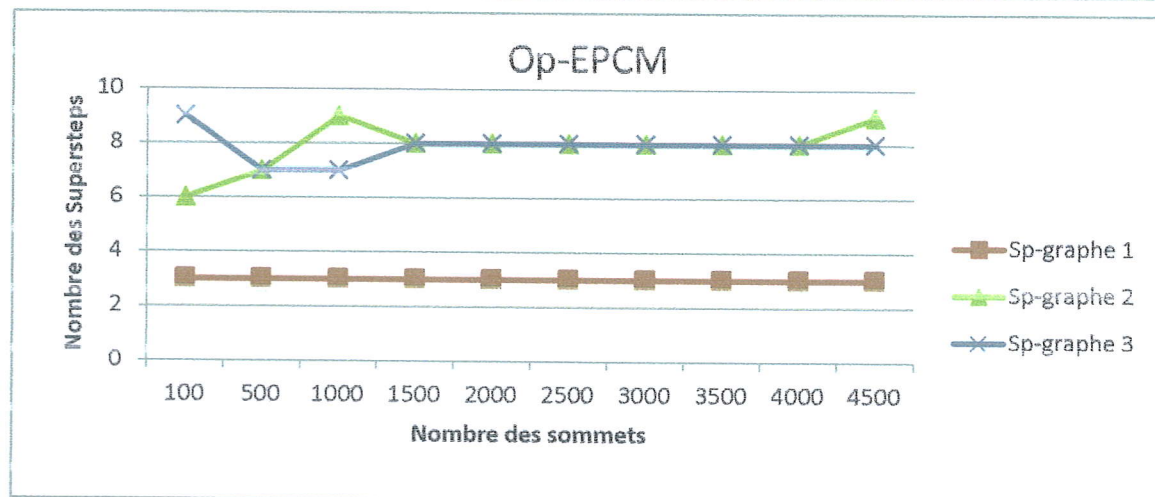
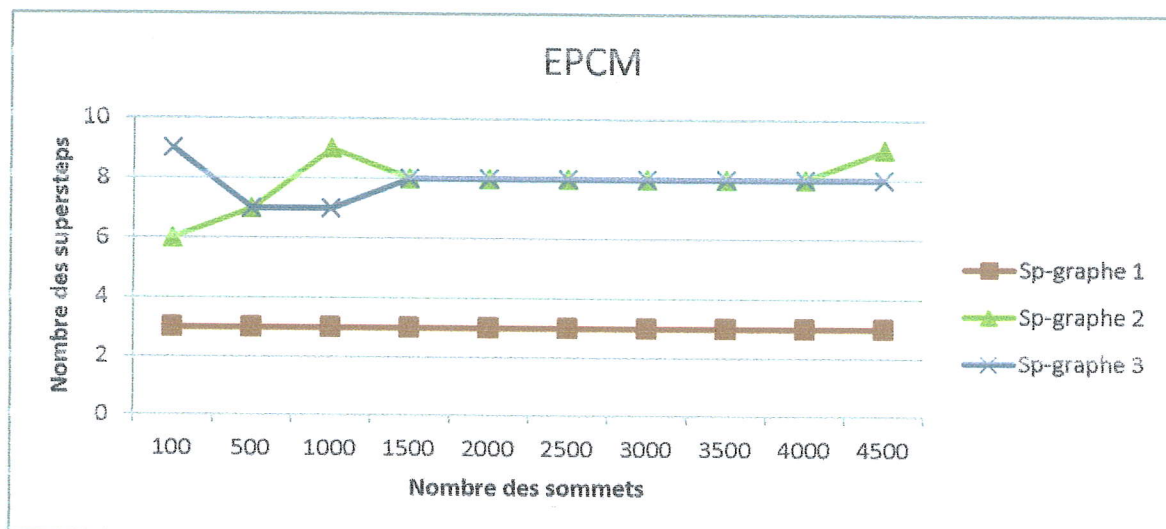


Figure 4.11: Test de convergence des algorithmes d'énumération des cliques maximales en termes de Superstep sous Giraph.

b) Test des algorithmes d'énumération des cliques maximums

Nous avons testé la scalabilité et la convergence des algorithmes d'énumération des cliques maximums de (HPECM, BKMP-AP, BKMP-SP) sous Giraph on utilisant les trois ensembles de données illustrées par les tableaux (tableau 4.4, tableau 4.5, tableau 4.7), Dans ces algorithmes le temps CPU, le nombre de message augmente d'une façon linéaire pour que le graphe devient grand, le nombre des Supersteps pour les trois algorithmes HPECM, BKMP-AP et BKMP-SP prend les mêmes valeurs, Les résultats obtenus illustré respectivement par la figure 4.12, figure 4.13, figure 4.14.

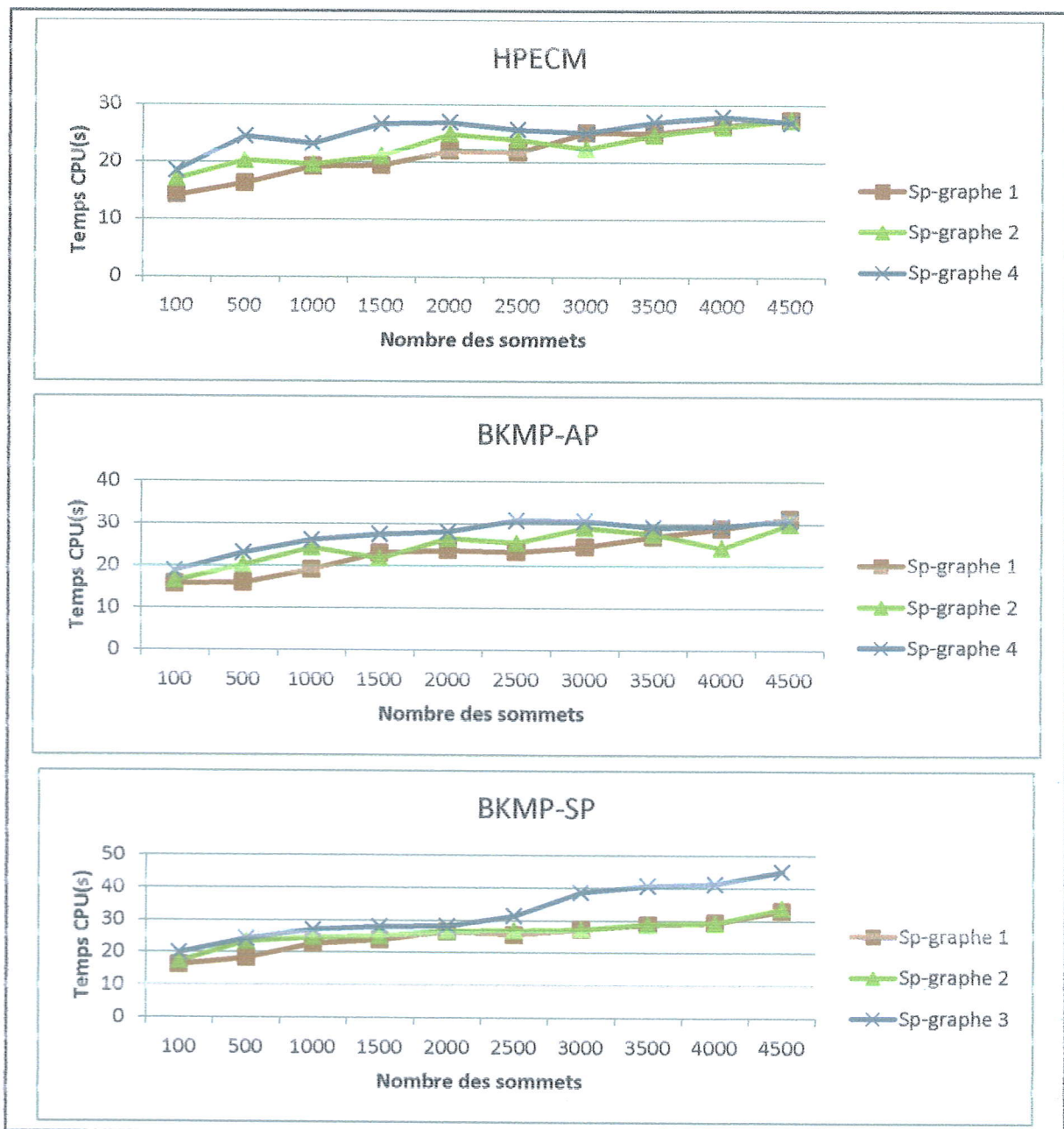


Figure 4. 12: Test de scalabilité des algorithmes d'énumération des cliques maximum sous Giraph en termes de temps CPU sous Giraph.

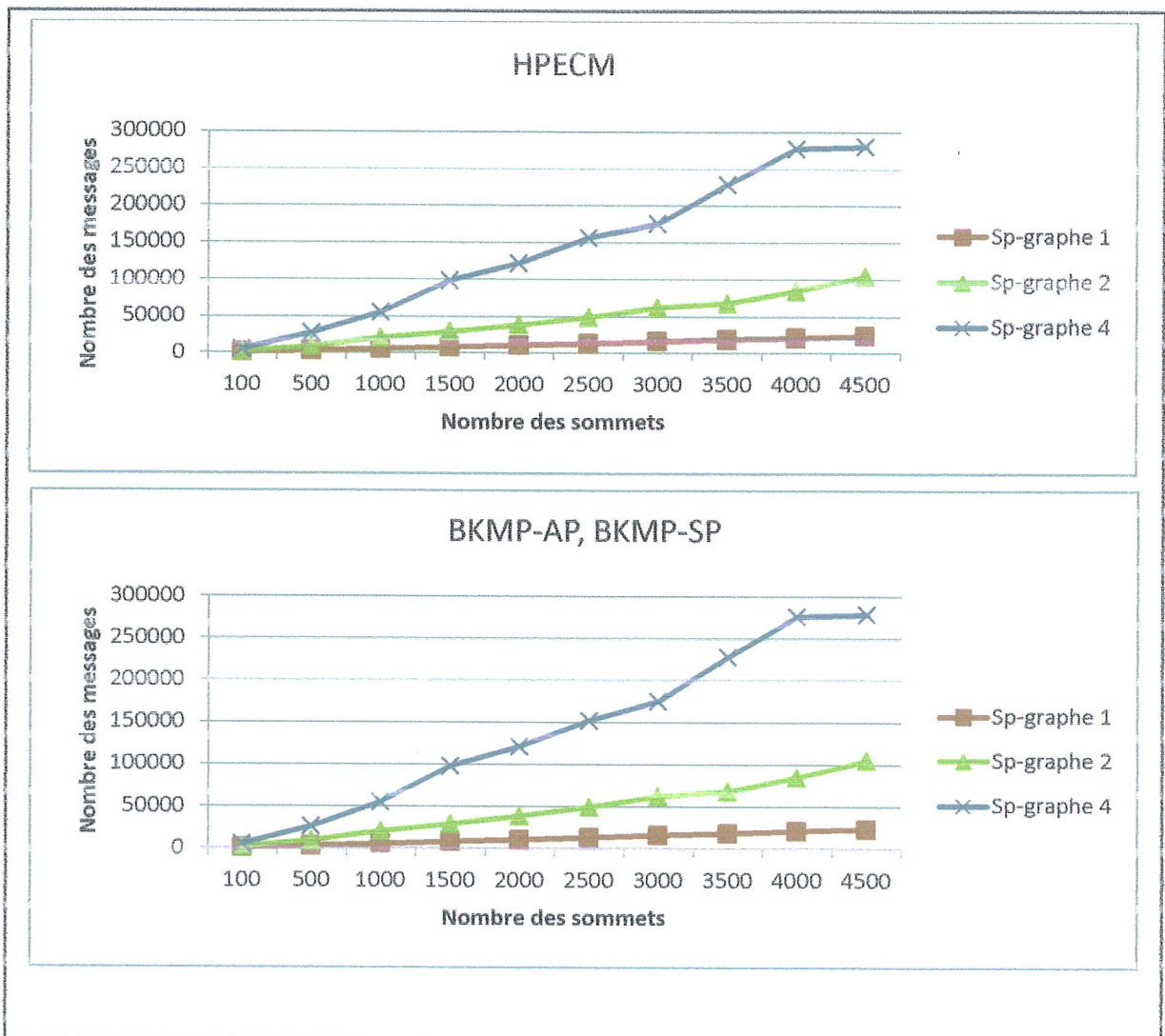


Figure 4.13: Test de scalabilité des algorithmes d'énumération des cliques maximum sous Giraph en termes de message sous Giraph.

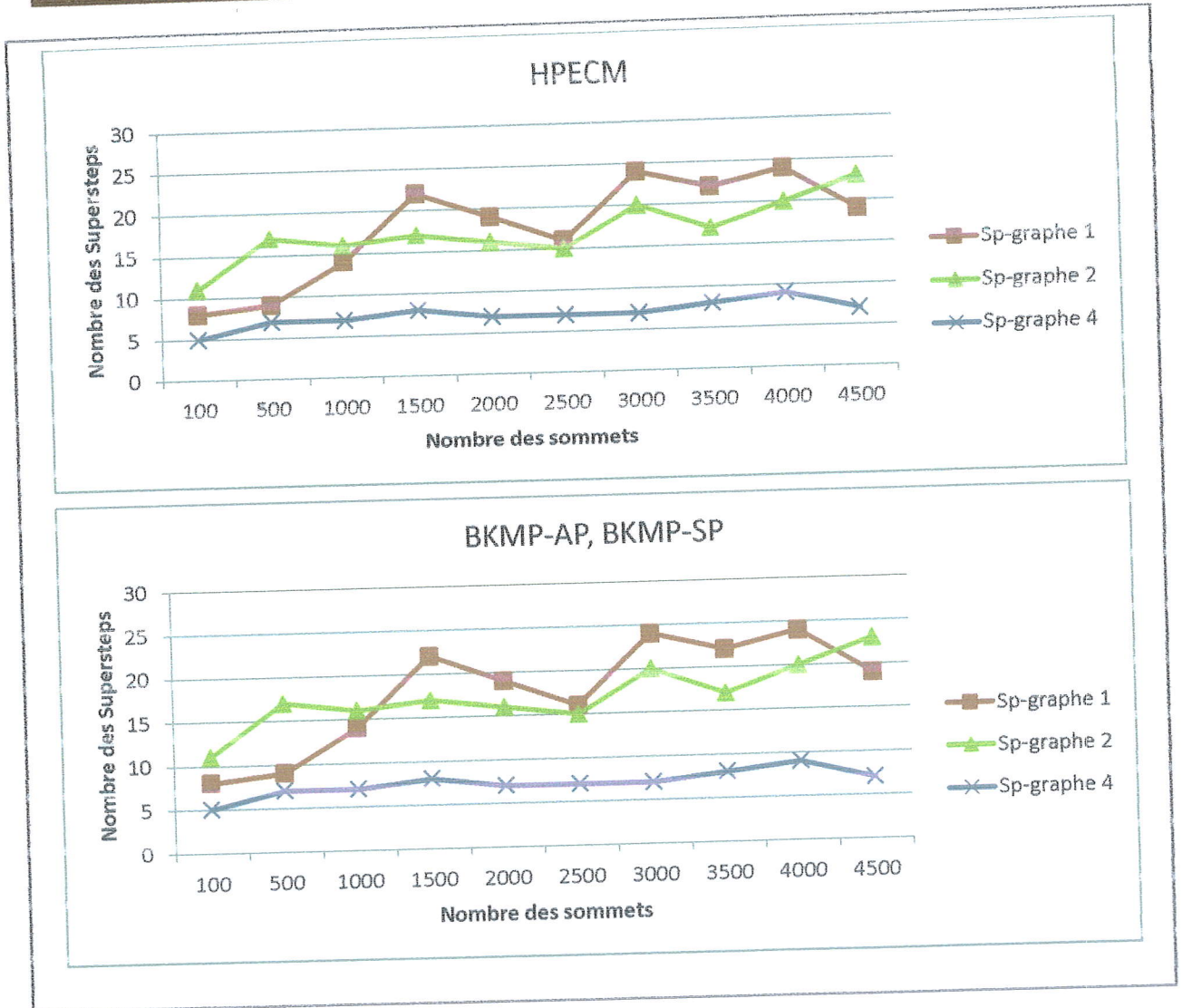


Figure 4.14: Test de convergence des algorithmes d'énumération des cliques maximum sous Giraph en termes de Superstep sous Giraph.

C) Comparaison entre les algorithmes d'énumération des cliques maximales

Dans cette section nous allons comparer les algorithmes d'énumération des cliques maximales proposés en termes du nombre de messages, nombre de Supersteps et temps CPU. Les résultats obtenus sont présentés respectivement par les figures 4.15, 4.16 et 4.17. Nous remarquons que les algorithmes BKP-AP et BKP-SP donne les meilleurs résultats que soit pour le nombre de Superstep, le nombre de message et le temps CPU. Ces deux algorithmes s'exécutent en deux Supersteps seulement ce qui a diminué le nombre de messages ainsi que le temps CPU. EPCM s'appuie sur l'envoi des messages de sorte que chaque sommet de graphe participe à l'envoi de message.

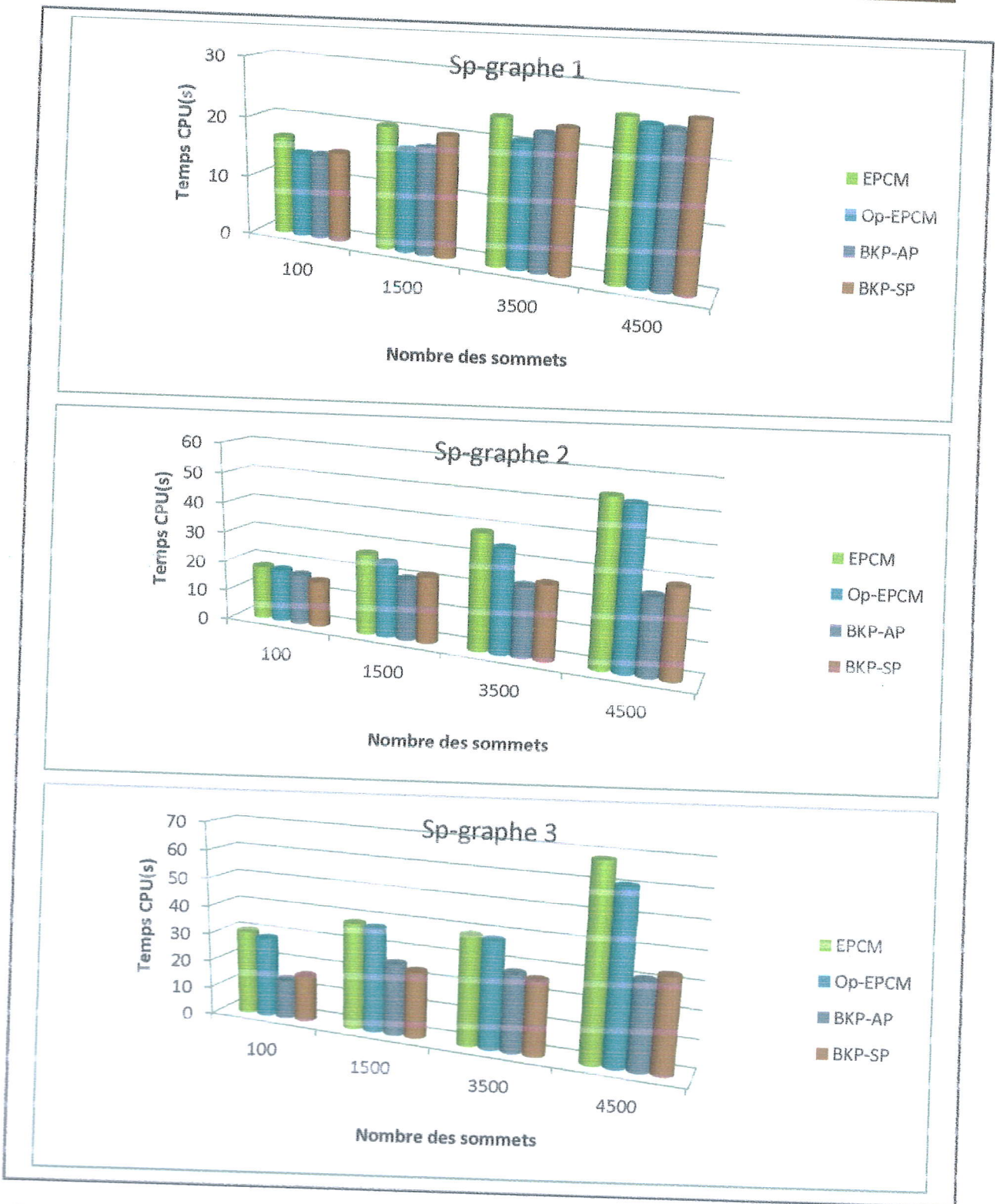


Figure 4.15: Résultats de Comparaison entre les algorithmes de cliques maximales en termes de temps CPU sous Giraph.

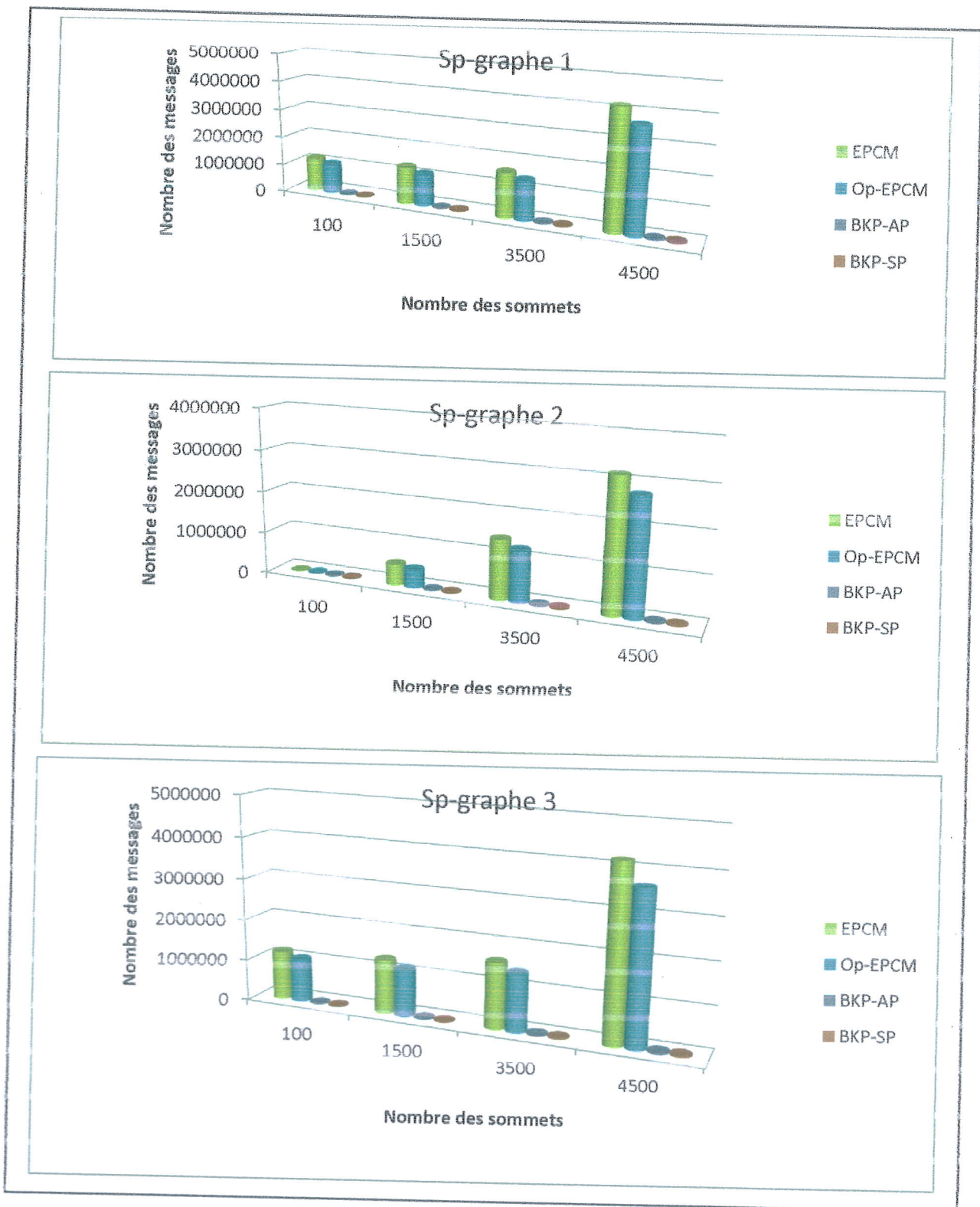


Figure 4.16: Résultats de Comparaison entre les algorithmes de cliques maximales en termes de nombre de message.

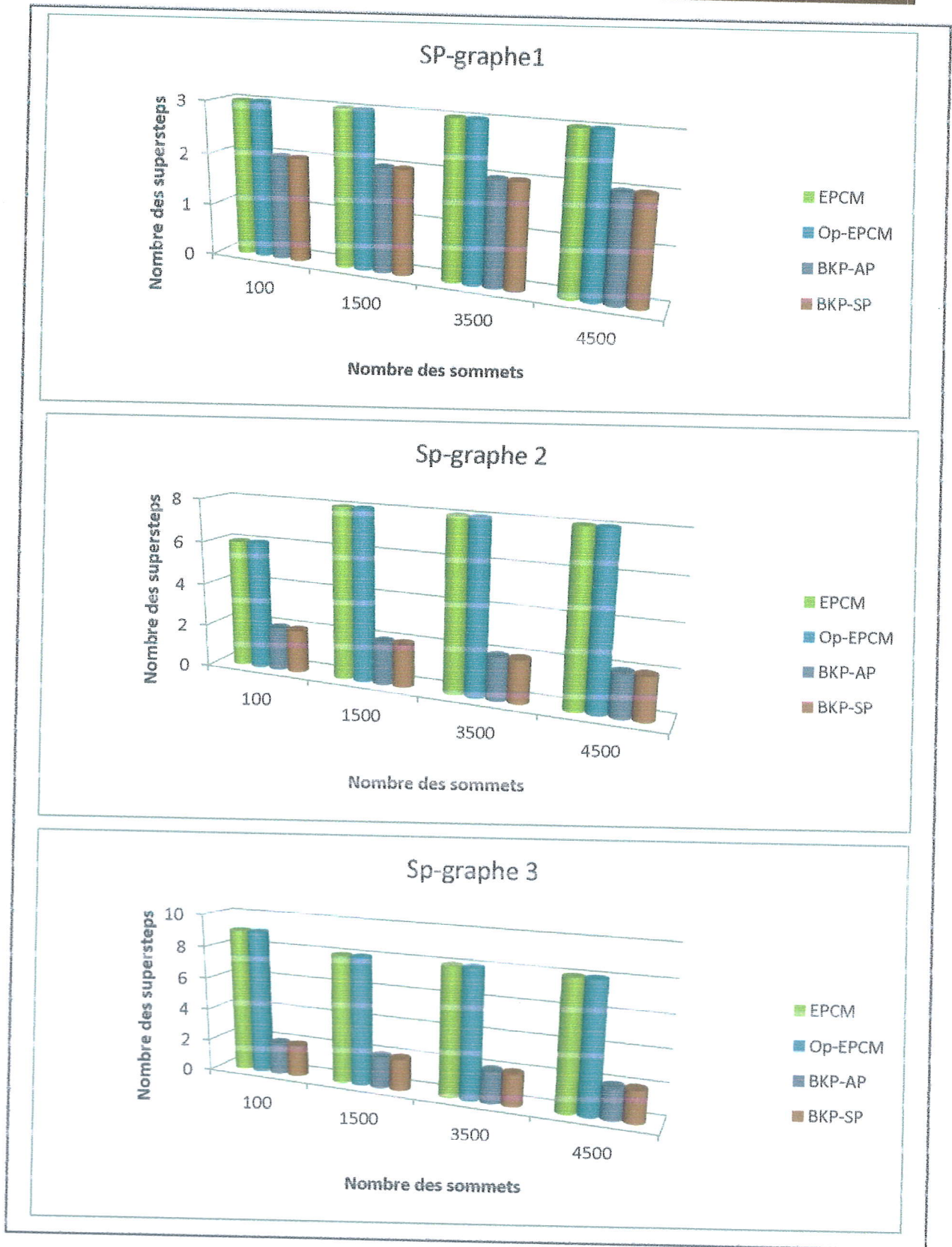


Figure 4.17: Résultats de comparaison entre les algorithmes d'énumération des cliques maximales en termes de Superstep sous Giraph.

d) Comparaison entre les algorithmes d'énumération des cliques maximums

Dans cette section nous allons comparer entre les algorithmes proposés d'énumération des cliques maximums sous Giraph en termes de nombre de Superstep, nombre de message ainsi que en termes de temps CPU. Les résultats de comparaison sont illustrés respectivement par les figures 4.18, 4.19 et 4.20. Les résultats montrent que l'algorithme HPECM donne le meilleur temps d'exécution par rapport aux autres algorithmes (BKMP-AP et BKMP-SP). Cela est bien raisonnable, du fait que HPECM écarte quelques sommets du calcul et à chaque fois il ne garde que le sommet de plus grand degré et ses voisins. Et pour la même raison, BKMP-AP donne un meilleur temps CPU par rapport à BKMP-SP, où tous les sommets sont concernés par les appels récursifs.

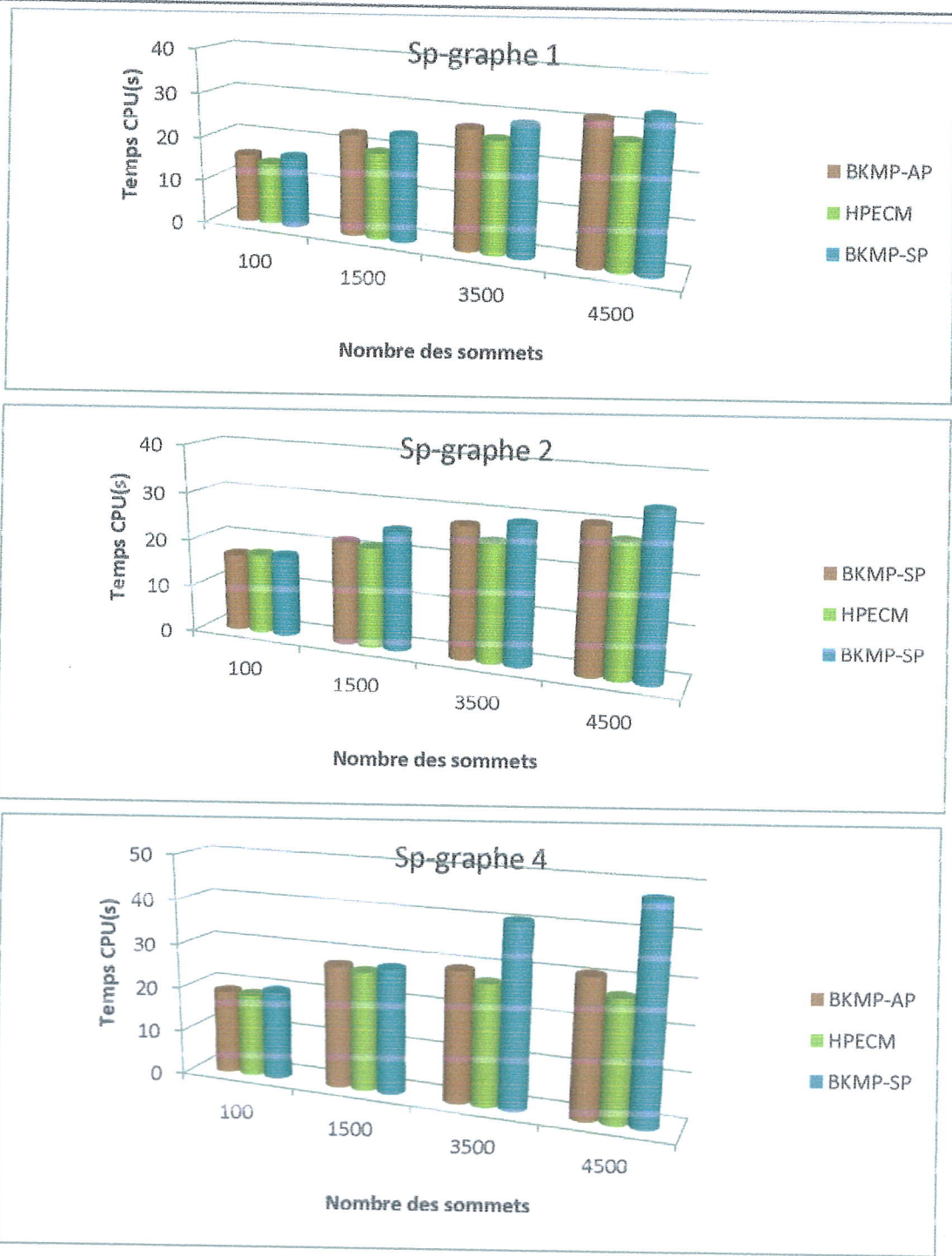


Figure 4.18: Résultats de comparaison des algorithmes de cliques maximums en termes de temps CPU sous Giraph.

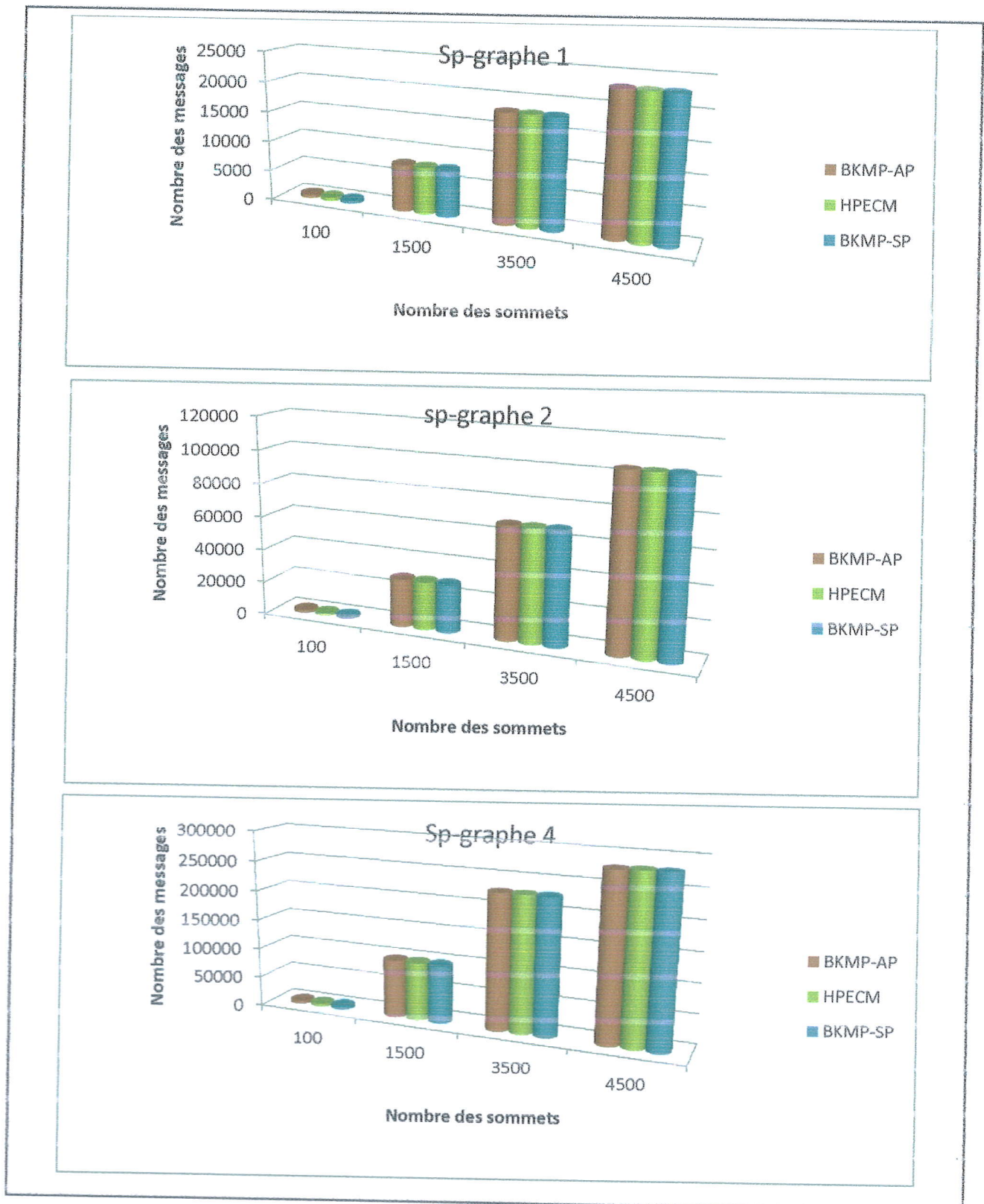


Figure 4.19: Résultats de comparaison des algorithmes de clique maximum en termes de nombre de message sous Giraph.

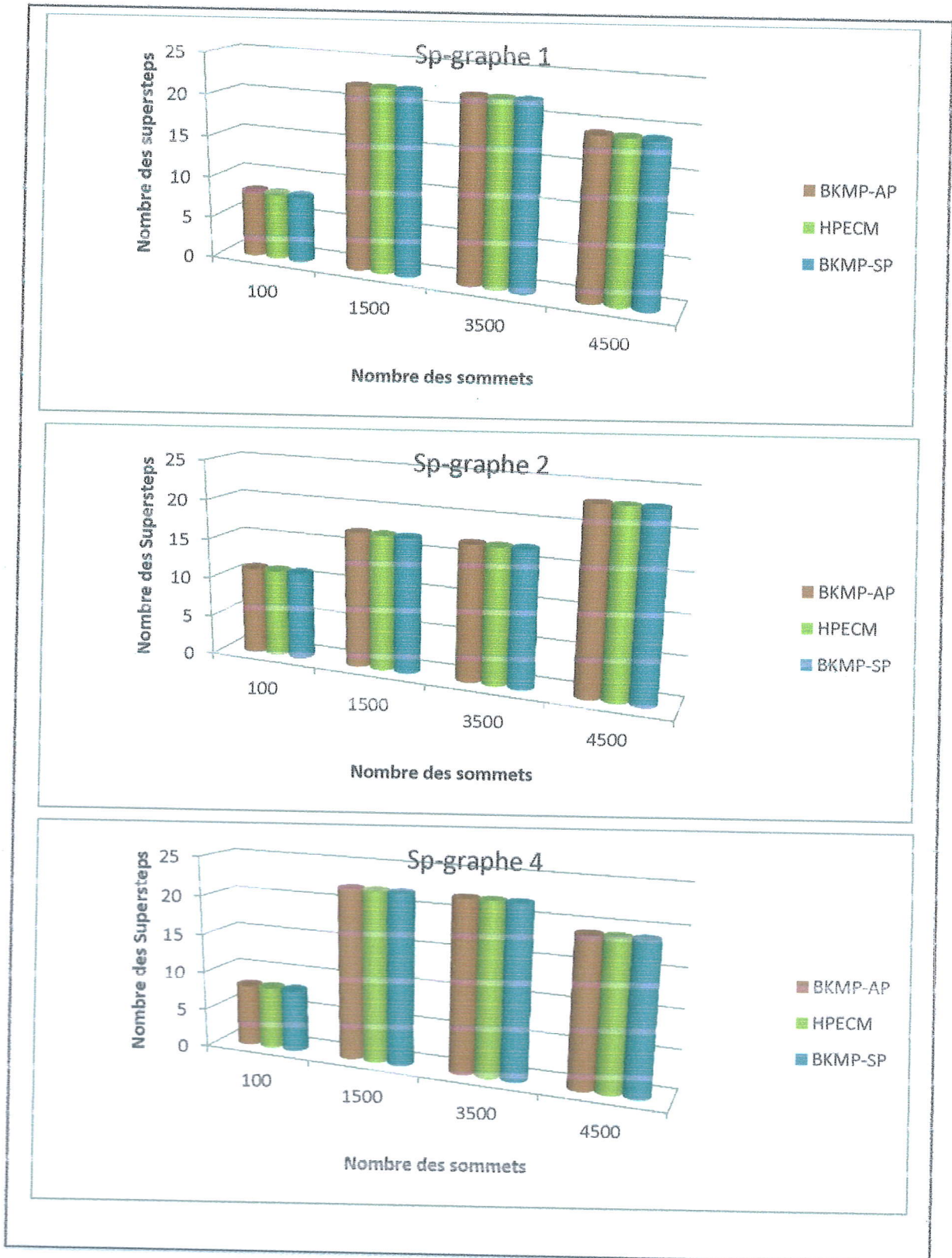


Figure 4.20: Résultats de comparaison entre les algorithmes de clique maximum en termes de nombre de Superstep sous Giraph.

e) Comparaison entre les algorithmes implémentés dans un environnement multi-node

Pour réaliser une comparaison entre les algorithmes implémentés, nous avons installé un petit cluster de quatre node, un master et trois slaves. Les résultats obtenus sont présentés par la figure. Nous avons remarqués que l'algorithme HPECM toujours donne le meilleur temps CPU par rapport aux autres algorithmes suivis par l'algorithme BKMP-AP (voir la figure 4.21). Concernant les algorithmes d'énumérations des cliques maximales, BKP-AP donne le meilleur temps CPU suivis par BKP-SP (voir la figure 4.22).

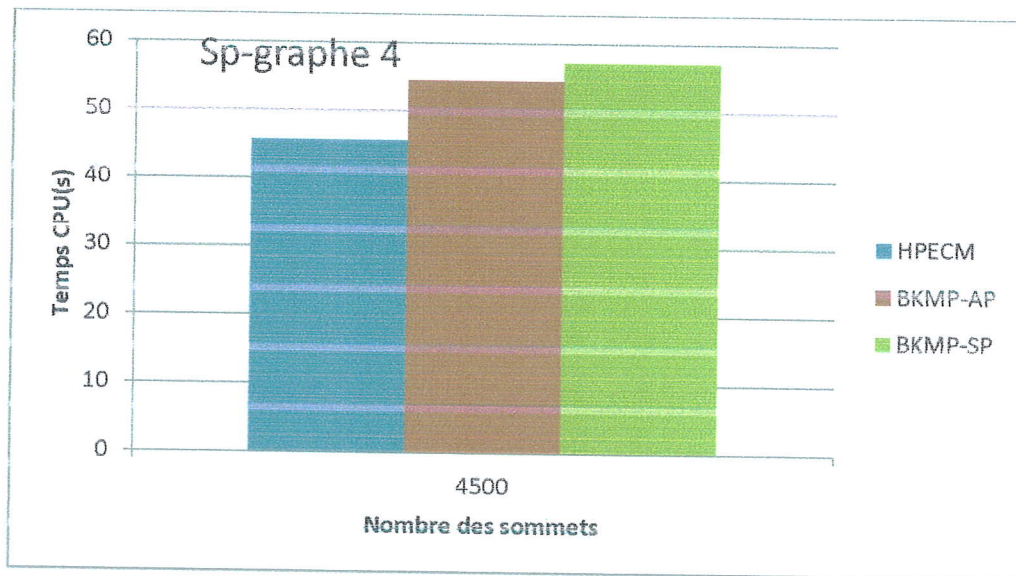


Figure 4.21 : Comparaison entre les algorithmes d'énumération des cliques maximums dans un environnement multi-node.

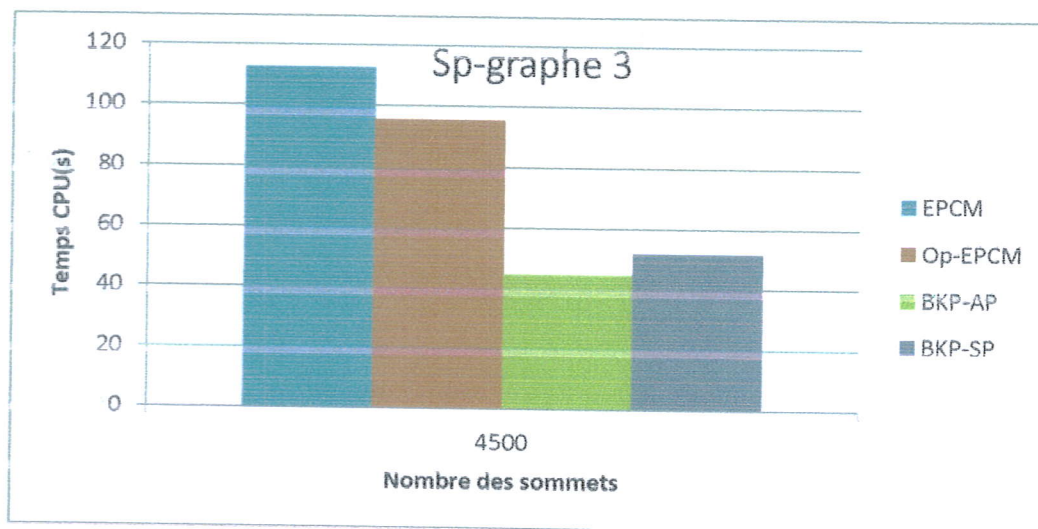


Figure 4.22 : Comparaison entre les algorithmes d'énumération des cliques maximales dans un environnement multi-node.

4.5 Conclusion

Dans ce chapitre nous avons testé la scalabilité de nos algorithmes proposés sous Giraph selon plusieurs critères. Ainsi nous avons comparé entre ces algorithmes en termes de temps CPU, nombre de message et nombre de Superstep. Les résultats obtenus montrent que l'algorithme exact implémenté permet une énumération exacte de toutes les cliques maximales et maximum dans un graphe donné. Par contre l'heuristique ne donne pas toujours la solution optimale. En termes de temps CPU, l'heuristique Glouton est la plus rapide, elle fonctionne bien même sur des graphes denses dans un temps raisonnable.

Conclusion générale :

L'objectif de notre projet était d'adapter des algorithmes de graphe séquentiels et de les implémenter dans un environnement distribué Hadoop/Giraph. Pour atteindre cet objectif, nous avons étudié les algorithmes d'énumération des cliques maximales et de clique maximum les plus célèbres qui se trouvent dans la littérature. Nous avons choisi un algorithme exact, l'algorithme de BK, et une heuristique gloutonne.

Les algorithmes choisis sont adaptés et sont implémentés sous Giraph et sont testés sur des benchmark de données. Les résultats obtenus montrent que l'algorithme exact implémenté permet une énumération exacte de toutes les cliques maximales et maximum dans un graphe donné. Par contre l'heuristique ne donne pas toujours la solution optimale. Ainsi, nous avons comparé ces algorithmes selon différents critères. En termes de temps CPU, l'heuristique Glouton est la plus rapide, elle fonctionne bien même sur des graphes denses dans un temps raisonnable.

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

En perspective on peut proposer :

- ✓ Evaluation des algorithmes proposés sur des graphes denses.
- ✓ Etude expérimentale des algorithmes proposés sur des larges clusters avec des larges graphes.
- ✓ Valider les algorithmes implémentés sur une plateforme Cloud publiques tels qu'Amazon.
- ✓ Adapter d'autres algorithmes d'énumération de cliques et réaliser des comparaisons avec ces algorithmes.

Problèmes rencontrés

- ✓ Ce projet étant très compliqué, nous avons rencontré de nombreux problèmes lors de l'installation et la configuration des outils qui nécessitent un débit élevé, notamment en ce qui concerne l'installation de Giraph.
- ✓ Manque de documentation pour la configuration d'éclipse avec Giraph, ce qui nous a obligés de créer le code java utilisant un éditeur de texte simple, le compiler et l'exécuter sur l'invite de commande.

Bibliographie

- [1] **Ahmed Ayadi, Jean-Sébastien Calvier & Adrian Guarrera.** Big Data et NoSQL, Rapport de recherche ,université de Nice Sofia Antipoli, France (2013).
- [2] **A.Hadoop.Hadoop.**<http://hadoop.apache.org>, (2011), (consulté le 15 février 2016).
- [3] **Akkoyunlu, E. A.** The enumeration of maximal cliques of large graphs, SIAM Journal on Computing, Vol. 2, No. 1 : pp. 1-6, (1973).
- [4] **B. paul-antoine.** Big Data : l'analyse de données en masse. http://paul-antoinebisgambiglia.univ-corse.fr/Big-Data-1-analyse-de-donnees-en-masse_a153.html. (2014). (Consulter le 30 janvier 2016).
- [5] **B. Christophe.** Les bénéfices d'Hadoop surpassent- ils les problèmes de la technologie?. <http://www.lemagit.fr/actualites/2240207836/Les-benefices-dHadoop-surpassent-ilsles-problèmes-de-la-technologie>. (2013). (Consulter le 20 Février 2016).
- [6] **Mission. Sciences et technologies de l'information et de la communication.** Hadoop, une technologie en plein essor. <http://www.bulletinselectroniques.com/actualites/72829.htm>. (2013). (Consulter le 26 fevrier 2016).
- [7] **B. Robillard.** Arborescences et représentations informatiques des graphes. 13 décembre 2007.
- [8] **Bachir Bekka,** Le théorème de Perron-Frobenius, les chaines de Markov et un célèbre moteur de recherche, Février 2007.
- [9] **Busygin, S.** A New Trust Region Technique for the Maximum Clique Problem. Report submitted, <http://www.busygin.dp.ua>. (2002).
- [10] **Claudio Martella, Roman Shaposhnik, Dionysios Logothetis.** Practical Graph Analytics with Apache Giraph, Apress Media, California, 2015.
- [11] **Chiquet, J.** Réseaux biologiques in Gazette. 2011.
- [12] **C. Bron and J. Kerbosch.** Algorithm 457: finding all cliques of an undirected graph. Commun. ACM 16(9):575–577, 1973.
- [13] **D. Thomas, C. Mickaël, V. Jérémie.** Rapport d'étude sur le Big Data. 05 octobre 2012.

- [14] **Daphne Vichot**, les quatre piliers d'une solution de gestion des Big Data.
- [15] **E. Balas and C. Yu**. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15 :1054–1068, 1986.
- [16] **E. Tomita, A. Tanaka, and H. Takahashi**. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.* 363(1):28–42, 2006.
- [17] **G. Plouin**. *Cloud Computing Et SaaS, Une repture décive pour l'informatiquen d'entreprise*, Dunod, Paris, (2009).
- [18] **Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski**. Pregel: a system for large- scale graph processing. In *Proceedings of the 2010 international conference on Management of data*, pages 135.146. ACM, 2010.
- [19] **G. Malewicz, H. Matthew, J. Aart**. Pregel: A System for Large-Scale Graph Processing.
- [20] <http://reseau-informatique.prestataires.com /conseils/virtualisation> (consulté le 05 Février 2016).
- [21] <http://www.astrosurf.com/luxorion/big-data-mining.htm>. (Consulter le 20 fevrier 2016).
- [22] <http://giraph.apache.org>. (Consulter le 10 Avril 2016).
- [23] <http://www.ensta.fr> (consulté le 12 Avril 2016).
- [24] Intel IT Center. *Guide de planification: Démarrer avec le Big Data. Des mesures Pour les responsables informatiques afin d'avancer avec le logiciel.* FÉVRIER 2013.
- [25] **Jean-Manuel Mény** IREM de LYON, parcours d'un graphe ISN 2013.
- [26] **Jérôme Richard**, Étude et parallélisation d'algorithmes pour l'analyse de grands graphes de réseaux sociaux. Université Orléans (2012-2013), INSA Centre Val de Loire, LIFO EA 4022, France.
- [27] **Jean-Baptiste Angelelli, Alain Guénoche et Laurence Reboul** : Détection de communautés, disjointes ou chevauchantes, dans les réseaux IML, 163 Av. de Luminy, 13288 Marseille cedex 9 guenoche@iml.univ-mrs.fr.
- [28] **J. Jeffry Howbert (jhowbert), Jacki Roberts (jackirob)**: The Maximum Clique Problem. CSEP 521 Applied Algorithms Winter 2007.
- [29] **K. Kannan**. Big Data Analytics. IBM Corporation, July, 2013. IBM Research Labs.

- [30] **Khaled Tannier**, MapReduce, Algorithme de parallélisation des traitements.
- [31] **L. Jimmy, S. Michael**. Design Patterns for Efficient Graph Algorithms in. University of Maryland, College Park.
- [32] **M.Audin**. etat de l'art sur le cloud computing et adaptation au logiciel libre. 2009.
- [33] **M.Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stocia, M. Zaharia**. Above the Clouds : A Berkely View of Cloud Computing. Technical Report n°UBC/EECS-2009-28, Berkely, (2009).
- [34] Markinson Understanding the difference between virtualization-and-cloud-computing . 6 Février 2013, consulté le 02 mars 2016. www.ubistor.com .
- [35] **Marie-Pierre Hamel et David Marguerit**, département Questions sociales. Analyse des big data. Quels usages, quels défis.
- [36] **M. Peter, C. Michael**. Tackling Big Data. NIST Information Technology Laboratory. Computer Security Division.
- [37] **M. Claudio**. Google Pregel is out. But what is Pregel? <http://blog.acaro.org/entry/pregel-is-out-but-what-is-pregel>. June 21, 2010. (Consulter le 10 mars 2016).
- [38] **Mirko Kämpf**. How-to: Write and Run Apache Giraph Jobs on Apache Hadoop. <http://blog.cloudera.com/blog/2014/02/how-to-write-and-run-giraph-jobs-On-hadoop/>, Février 2014.
- [39] **Marco Boudinich**, the Maximum Clique Problem
E-mail: mbh@trieste.infn.it.
- [40] Pseudo code for Hadoop PageRank . [salsahpc, http://salsahpc.indiana.edu/csci-b649-spring-2014/projects/project3.html](http://salsahpc.indiana.edu/csci-b649-spring-2014/projects/project3.html). Consulté le 16/Avril/2016.
- [41] **R. Beyya, C.S. Yeo, S. Venugopal**. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In HPCC'08, Proceedings of the 10th IEEE International Conference Computing and Communications, pp. 5-13, Dalian, China, (2008).
- [42] **Ralf Lammel**, Google's MapReduce Programming Model-Revisited. Microsoft Corp., Redmond, WA 22 January, 2006, Draft.
- [43] **Simon BLUM**. Les instituts de sondages sont-ils prêts à l'utilisation des BigData?, Master MIAGE en apprentissage, 2012-2013.

- [44] **Stefane Fermigier**. Big Data & open source: Une convergence inévitable? version 1.0 - mars 2012.
- [45] **S. Christine**. Théorie des graphes et optimisation dans les graphes.
- [46] **Sherif Sakr**, Data Processing of large -scale graph, 10 juin 2013.
- [47] Tom White, Hadoop : the definitive guide, Sebastopol, CA : O'Reilly, 2012.
- [48] Université d'Orléans. Représentation des graphes en informatique. Année 2005- 2006.
- [49] **Yann Gourvennec**. Les 5 défis du Big Data selon Talend.
<http://blog.businessdecision.com/bigdata/2015/03/defis-big-data/>, 3 mars 2015. Consulté le 26/03/2016.

Annexe A :

A.1 Les Commandes de Giraph et Hadoop :

A.1.1 Les Commandes de Giraph :

Le tableau ci-dessus est une description des paramètres de l'exemple de Giraph qu'ont exécuté :

Nom du paramètre	Description
-vif	Le VertexInputFormat pour l'emploi.
-Vip	Le chemin à partir de laquelle le HDFS VertexInputFormat charge les données.
-of	L'OutputFormat pour l'emploi.
-op	Le chemin d'accès dans HDFS à laquelle l'OutputFormat écrit les données.
-w	Nombre de travailleurs (le nombre de cartographes qui ont commencé à être en parallèle).

Tableau A.1 Tableau des commandes de Giraph.

A.1.2 Les Commandes de Hadoop :

Pour obtenir la liste complète des commandes, il suffit d'exécuter la commande suivante **bin/hadoop dfs**.

Le tableau ci-dessus contient quelque commande de Hadoop et ses descriptions :

Commande	Description
-mkdir <path>	Créer un dossier dans HDFS pour y stocker un fichier.
bin/hadoop fs -copyFromLocal <chemin fichier local> <URI>	A partir du system de fichier local, copier un fichier dans un dossier créé dans HDFS.
bin/hadoop fs -ls <path>	Afficher les informations qui concernent le fichier.

bin/start-dfs.sh	Pour démarrer NameNode, DataNode, SecondaryNameNode.
bin/start-mapred.sh	Pour démarrer JobTracker, TaskTracker.

Tableau A.2 Tableau distributif des commandes de Hadoop.

A.2 Test de Giraph :

A.2.1 Exemple de ShortestPath

Avec Giraph et Hadoop déployées, nous exécutons le premier job de Giraph : Simple ShortestPaths Computation, qui lit comme entrée les données d'un graphe sous forme d'un fichier texte (voir la figure A.1) et calcule la distance minimale à partir d'un noeud source à tous les autres noeuds. Le noeud source (le noeud source a une distance 0) est toujours le sommet numéro 1 dans le fichier d'entrée. On va utiliser le format d'entrée JsonLongDoubleFloatDoubleVertexInputFormat.

1- Copie le fichier vers HDFS :

```
[1,4.3,[[2,2.1],[3,0.7]]]
[2,0.0,[[4,1.1],[6,1.2]]]
[3,1.8,[[1,0.7]]]
[4,2.2,[[5,3.2]]]
[5,3.3,[[4,1.0]]]
[6,2.5,[]]
```

Figure A.1 TestShortestPath.txt.

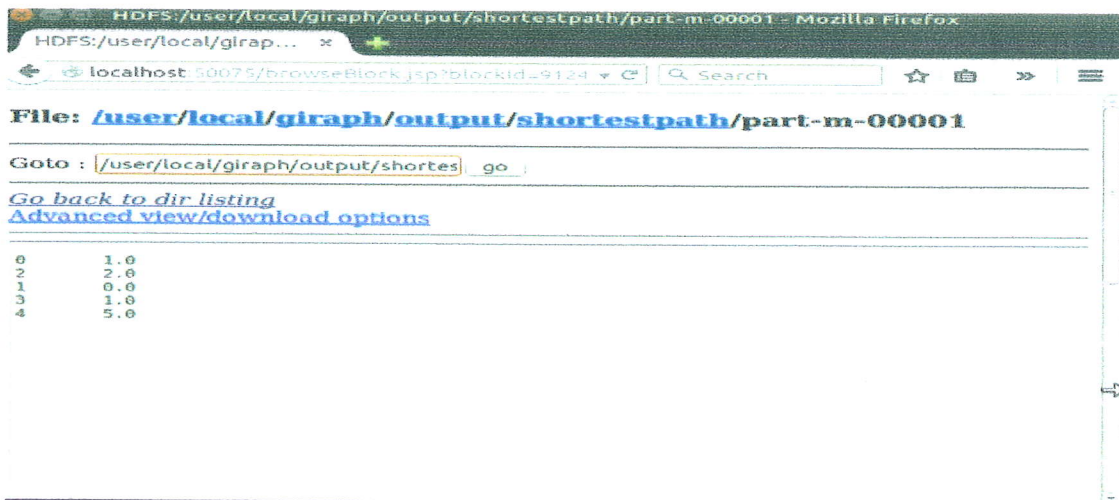
```
$sudo mkdir -p /user/local/giraph/input
$HADOOP_HOME/bin/hadoop dfs -copyFromLocal TestShortestPath.txt
/user/local/giraph/input/ TestShortestPath.txt
```

2- L'execution de l'exemple

```
$HADOOP_HOME/bin/hadoop jar $GIRAPH_HOME/giraph
examples/target/giraphexamples-1.1.0-for-hadoop-0.20.203.0-jar-with-
dependencies.jar org.apache.giraph.GiraphRunner
org.apache.giraph.examples.SimpleShortestPathsComputation - vif
```

```
org.apache.giraph.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat -vip
/user/giraph/input/TestShortestPath.txt -vof
org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op /user/giraph/output
-w 1
```

3- Résultat :



A.2.2 Exécution de Page Rank dans Giraph :

➤ Input :

```
[0,1,[[1,1],[3,3]]]
[1,2,[[0,1],[2,2],[3,1]]]
[2,3,[[1,2],[4,4]]]
[3,4,[[0,3],[1,1],[4,4]]]
[4,5,[[3,4],[2,4]]]
```

```
meryem@master:~$ $HADOOP_HOME/bin/hadoop jar $GIRAPH_HOME/giraph-examples/target
/giraph-examples-1.2.0-SNAPSHOT-for-hadoop-1.2.1-jar-with-dependencies.jar org.a
pache.giraph.GiraphRunner org.apache.giraph.examples.SimplePageRankComputation -
vif org.apache.giraph.io.formats.JsonLongDoubleFloatDoubleVertexInputFormat -vip
/user/local/giraph/input/TestPR.txt -vof org.apache.giraph.io.formats.IdWithVal
ueTextOutputFormat -op /user/local/giraph/output/pagerankTest2 -w 1 -mc org.apac
he.giraph.examples.SimplePageRankComputation\SimplePageRankMasterCompute
```

➤ Output :

File: [/user/local/giraph/output/pagerankTest2/part-m-00001](#)

Goto : go

[Go back to dir listing](#)
[Advanced view/download options](#)

```
0 0.16682289373110673
2 0.17098446873203233
1 0.2417888079775044
3 0.24178880797750438
4 0.17098446873203233
```

A.2.3 Exécution de Composant Connexe dans Giraph :

➤ Input :

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

```
meryem@master:~$ SHADOOP_HOME/bin/hadoop jar $GIRAPH_HOME/giraph-examples/target/giraph-examples-1.2.0-SNAPSHOT-for-hadoop-1.2.1-jar-with-dependencies.jar org.apache.giraph.GiraphRunner org.apache.giraph.examples.ConnectComponentsComputation -vif org.apache.giraph.io.formats.IntIntNullTextInputFormat -vip /user/local/giraph/input/compccon.txt -vof org.apache.giraph.io.formats.IdWithValueTextOutputFormat -op /user/local/giraph/output/compcconnectTest1 -w 1 -ca SplitMasterWorker=false
```

➤ Output:

master 50075/browseBlock.jsp?blockId=79515212531259903268b Search

File: [/user/local/giraph/output/compcconnectTest1/part-m-00001](#)

Goto : go

[Go back to dir listing](#)
[Advanced view/download options](#)

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

Annexe B :

B.1 Installation de Hadoop Singale Node :

➤ **Mise à jour de système :**

Avant toutes installations de nouveaux paquets, mettez à jour le cache des paquets sur votre machine. La commande suivante téléchargera la nouvelle liste des paquets proposés par le dépôt.

```
$ sudo apt-get update
```

➤ **Java :**

Hadoop nécessite une version Java 7 ou au moins Java 6. Pour ce faire la version 7 de Java sera utilisée via la distribution OpenJDK. Voir commande ci-dessous pour installer OpenJDK 7 sur un Ubuntu :

```
$ sudo apt-get install openjdk-7-jdk
```

Après l'installation, Pour assurer que la version Java est correctement installée. On exécute la commande :

```
$ java -version
```

Par conséquent, on obtient comme résultat :

```
java version "1.7.0_91"  
OpenJDK Runtime Environment (IcedTea 2.6.3) (7u91-2.6.3-0ubuntu0.15.04.1)  
OpenJDK 64-Bit Server VM (build 24.91-b01, mixed mode)
```

Figure B. 1: Version de java installée.

Pour information, le chemin d'installation d'OpenJDK7 sur notre machine est :
/usr/lib/jvm/java-7-openjdk-amd64

- **Installation SSH (Secure Shell) et génération de clé :**

Hadoop requiert SSH pour la gestion de ses noeuds (local ou à distance). Pour installer

ssh on utilise la commande suivante :

```
$ sudo apt-get install ssh
```

Pour activer la connexion sans mot de passe, on génère une nouvelle clé SSH avec un mot de passe vide :

```
$ sudo apt-get install ssh  
$ ssh-keygen -t rsa -P ""
```

On aura comme résultat :

```
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/meryem/.ssh/id_rsa): /home/meryem/.ssh/id_rsa  
Created directory '/home/meryem/.ssh'.  
Your identification has been saved in /home/meryem/.ssh/id_rsa.  
Your public key has been saved in /home/meryem/.ssh/id_rsa.pub.  
The key fingerprint is:  
79:6c:91:52:9b:e3:56:ce:ab:3c:70:9e:ef:69:67:4d meryem@meryem-VirtualBox  
The key's randomart image is:  
+---[RSA 2048]---+  
  .           |  
  .+          |  
  . *         |  
  = *        |  
  S * 0       |  
  .+ . . E    |  
  + . . 0     |  
  .+...0 .    |  
  0=+0       |  
+-----+-----+
```

Figure B. 2: Exemple de la clé ssh générée.

Autoriser l'accès à cette machine avec la clé SSH récemment créée :

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

La dernière chose à réaliser est de tester la connexion SSH :

```
$ ssh localhost
```

- **Installation de Hadoop :**

Télécharger une version stable de Hadoop à partir de site d'Apache suivant :
<http://apache.mesi.com.ar/hadoop/common/hadoop1.2.1/hadoop.1.2.1.tar.gz>

- Extraire le fichier d'archives de Hadoop dans le répertoire de Hadoop :

```
$ sudo tar xzf hadoop-1.2.1.tar.gz  
$ sudo mv hadoop-1.2.1 /usr/local/hadoop
```

-Changer le propriétaire de tous les fichiers pour l'utilisateur de Hadoop.

```
$ sudo chown -R user: user Hadoop
```

B.1.1 Configuration de Hadoop single node :

Nous avons opté pour une installation de Hadoop sur un seul noeud. Le même ordinateur va servir à la fois pour le noeud maître (Namenode et JobTracker) et le noeud esclave (Datanode et TaskTracker). Les configurations suivantes de Hadoop sont nécessaires:

➤ **Modification des variables d'environnement :**

Pour modifier les variables d'environnement utilisées par Hadoop, on utilisant la commande:

```
$ nano ~/.bashrc
```

Ensuite nous avons ajouté les lignes suivantes à la fin de fichier:

```
export HADOOP_HOME=/usr/local/Hadoop  
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk- amd64
```

On sauvegarde et on ferme le fichier, ensuite nous avons tapé la commande suivante:

```
$ source ~/.bashrc
```

Pour faire un simple test des commandes Hadoop nous écrivons :

```
$ HADOOP_HOME/bin/hadoop fs
```

➤ **Mise à jour des fichiers de configuration de Hadoop :**

Un ensemble de configuration de Hadoop sont nécessaires. Les fichiers de configuration se trouvent dans le répertoire \$HADOOP_HOME/conf :

1- Editer le fichier conf/hadoop-env.sh (\$gedit hadoop-env.sh) et ajouter les lignes suivants à la fin de fichier :

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
```

Spécification de chemin d'installation de java et activation d'IPv4

2- Crée le répertoire de stockage temporaire :

```
$ sudo mkdir -p /usr/local/Hadoop
$ sudo chown ubuntu:ubuntu /usr/local/hadoop
$ sudo chmod 750 /usr/local/Hadoop
```

Dans les fichiers core-site.xml, mapred-site.xml et hdfs-site.xml, nous avons ajouté les lignes qui sont entre <configuration> et </configuration> :

3- nano \$HADOOP_HOME/conf/core-site.xml et ajouter les lignes suivantes entre

<configuration>...</configuration> :

```
core-site.xml x
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides here -->
<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:8020</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/usr/local/tmp/hadoop</value>
</property>
</configuration>
```

Spécifier le port utilisé

Spécifier le répertoire que Hadoop va utiliser pour sauvegarder les données

Figure B. 3: Mise à jour de core-site.xml.

4- Mise à jour du fichier conf/mapred-site.xml :

```

mapred-site.xml x
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>
  <property>
    <name>mapred.reduce.tasks</name>
    <value>3</value>
  </property>
  <property>
    <name>mapred.tasktracker.map.tasks.maximum</name>
    <value>3</value>
  </property>
  <property>
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
    <value>3</value>
  </property>
</configuration>

```

Spécifier le nombre maximum des tâches Map et Reduce

Figure B. 4: Mise à jour de mapred-site.xml.

5- Mise à jour du fichier conf/hdfs-site.xml :

```

hdfs-site.xml x
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>dfs.data.dir</name>
    <value>/usr/local/tmp/hadoop/dfs/name/data</value>
    <final>true</final>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>/usr/local/tmp/hadoop/dfs/name</value>
    <final>true</final>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>

```

Spécification de nombre de réplication de fichier.

Figure B. 5: Mise à jour de hdfs-site.xml.

6- Formatage du système de fichier HDFS mis en place :

Avant d'utiliser notre noeud Hadoop pour la première fois, il est recommandé de formater le système de fichier avant de démarrer les processus NameNode, SecondaryNameNode et DataNode. Pour lancer le formatage, nous avons utilisé la commande suivante :

```

$HADOOP_HOME/bin/hadoop namenode - format

```

7- Lancement les éléments (SecondaryNameNode, JobTracker, TaskTracker, NameNode, DataNode). Pour démarrer tous ces éléments à la fois, il suffit de faire la commande suivante :

```
$HADOOP_HOME/bin/start-all.sh
```

8- Réaliser un simple test par la commande jps :

```
meryem@meryem-VirtualBox:~$ cd /usr/local/hadoop
meryem@meryem-VirtualBox:/usr/local/hadoop$ jps
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
2423 SecondaryNameNode
2508 JobTracker
4388 Jps
2263 DataNode
2653 TaskTracker
2111 NameNode
```

Figure B. 6: Test de démarrage des éléments de Hadoop.

9- Interfaces web de Hadoop : Hadoop est livré avec des interfaces web qui sont par défaut disponible à ces adresses :

http:// localhost: 50070/ : interface web de NameNode.

http:// localhost: 50075/ : interface web de DataNode.

http:// localhost: 50090/ : interface web de SecondaryNameNode.

http:// localhost: 50030/ : interface web de JobTracker (voir la figure 4.7).

http:// localhost: 50060/ : interface web de TaskTracker.

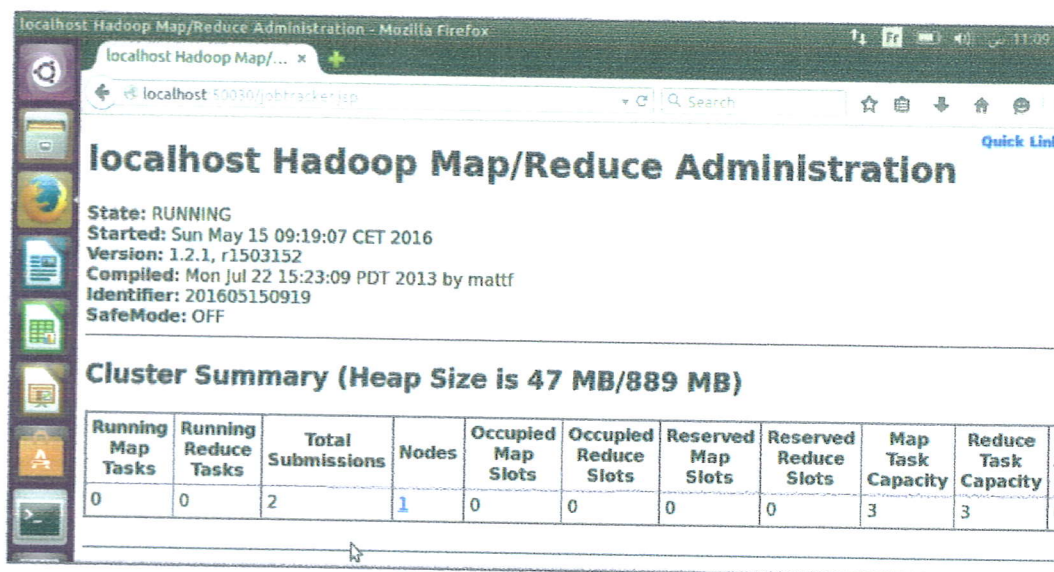


Figure B. 7: Interface web de Hadoop (Interface de JobTracker).

Arrêt des jobs Hadoop : L'arrêt des jobs Hadoop est rendu possible par l'utilisation de la commande suivante :

```
$HADOOP_HOME/bin/stop-all.sh
```

B.2 Installation de Giraph

Giraph prend en charge plusieurs versions de Hadoop. Dans cette section, nous allons décrire l'installation et l'exécution de Giraph sur un seul nœud Hadoop: 1.2.1

1- Installer Git et Maven en exécutant les commandes suivantes:

```
$ sudo apt-get install git          pour obtenir les nouvelles version maven et giraph
$ sudo apt-get install maven        Giraph nécessite Maven, pour supporter de multiples
                                     versions de Hadoop.
```

2- Installation de Giraph :

```
$cd /usr/local/
$sudo git clone https://github.com/apache/giraph.git
$sudo chown -R user:user giraph
```

```
export GIRAPH_HOME=/usr/local/giraph
```

3- Emballer Giraph dans des fichiers JAR en exécutant les commandes suivantes :

```
$source $HOME/.bashrc
$cd $GIRAPH_HOME
$mvn package -DskipTests
```

Lorsqu'on termine la fenêtre suivante s'affiche et indique que l'installation se fait avec succès (voir la figure B.8).

```
INFO] -----
INFO] Reactor Summary:
INFO] -----
INFO] Apache Giraph Parent ..... SUCCESS [9:33.283s]
INFO] Apache Giraph Core ..... SUCCESS [6:56.035s]
INFO] Apache Giraph Examples ..... SUCCESS [31.148s]
INFO] Apache Giraph Accumulo I/O ..... SUCCESS [3:30.016s]
INFO] Apache Giraph HBase I/O ..... SUCCESS [5:13.744s]
INFO] Apache Giraph HCatalog I/O ..... SUCCESS [3:46.980s]
INFO] Apache Giraph Hive I/O ..... SUCCESS [33:56.091s]
INFO] Apache Giraph Cora I/O ..... SUCCESS [1:56.574s]
INFO] Apache Giraph Rexster I/O ..... SUCCESS [0.710s]
INFO] Apache Giraph Rexster Kibble ..... SUCCESS [30.546s]
INFO] Apache Giraph Rexster I/O Formats ..... SUCCESS [4:20.262s]
INFO] Apache Giraph Distribution ..... SUCCESS [39.966s]
INFO] -----
INFO] BUILD SUCCESS
INFO] -----
INFO] Total time: 1:11:12.435s
INFO] Finished at: Tue Mar 31 12:44:22 PDT 2015
INFO] Final Memory: 85M/218M
INFO] -----
```

Figure B. 8: Succès de l'installation de Giraph.

Annexe C :

C.1 Configuration de Hadoop multi nœud :

Pour faire la configuration de Hadoop multi noeuds, nous avons fait des copies de la machine virtuelle single node configuré précédemment :

une maitre et esclave et les autres sont des esclaves (slave1, slave2, slave3).

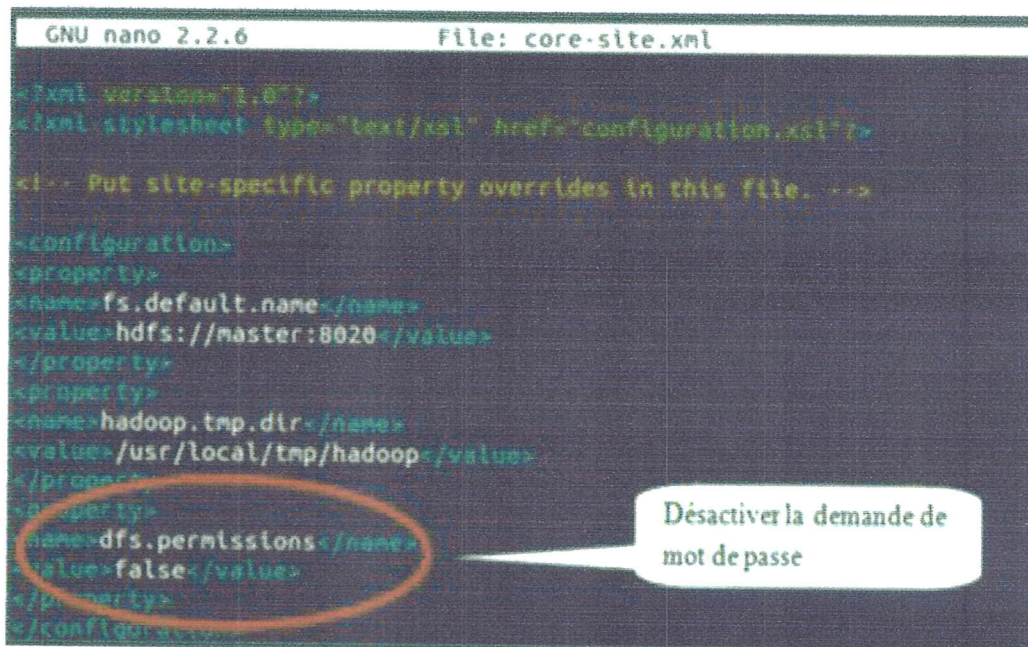
1- Désactivation de Pare- feu et nous assurons que les ports utilisés par Hadoop sont ouverts :

```
$ sudo ufw status Si elle est active, désactivez- le : $ufw disable
```

2- La modification sera seulement dans le fichier **core-site.xml** de toutes les machines (voir la figure C.1).

```
$ cd /usr/local/hadoop/conf
```

```
$ sudo gedit core-site.xml
```



```
GNU nano 2.2.6 File: core-site.xml
<?xml version="1.0"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://master:8020</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/usr/local/tmp/hadoop</value>
</property>
<property>
<name>dfs.permissions</name>
<value>>false</value>
</property>
</configuration>
```

Figure C.1 Configuration du fichier core-site.xml.

3- La modification de fichier hosts pour tous les nœuds. En ajoutant l'adresse IP de chaque nœud et le nom d'hôte (voir la figure C.2).

```
$sudo gedit /etc/hosts
```

```
192.168.0.5 master
192.168.0.9 slave1
192.168.0.6 slave2
192.168.0.10 slave3
192.168.0.11 slave4

# The following lines are desirable for IPv6 capable hosts
::1    ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Figure C.2 La mise à jour de fichier hosts.

4- La modification de nom de chaque nœud (master, slave1, slave2, slave3, slave4), on utilisant :

```
$ sudo gedit /etc/hostname
```

5- La modification de fichier masters, on met le nom de maître (master):

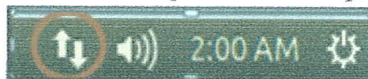
```
$cd /usr/local/hadoop/conf
$sudo gedit masters
```

C.1.1 La création de réseau entre les machines :

Pour faire communiquer les différents machines virtuelles, il faut qu'ils sont apparaitre dans le même sous réseau. Alors les adresses IP sont configurées manuellement.

Étape1 : La modification d'adresses IP de chaque nœud. Cliquer sur l'icône de

connexion voir l'image suivant :



- Ethernet connections.
- Edit.

- IPv4 Setting manuelle.
- entrer l'adresse et Masque de réseau.
- Save (voir la figure C.3).

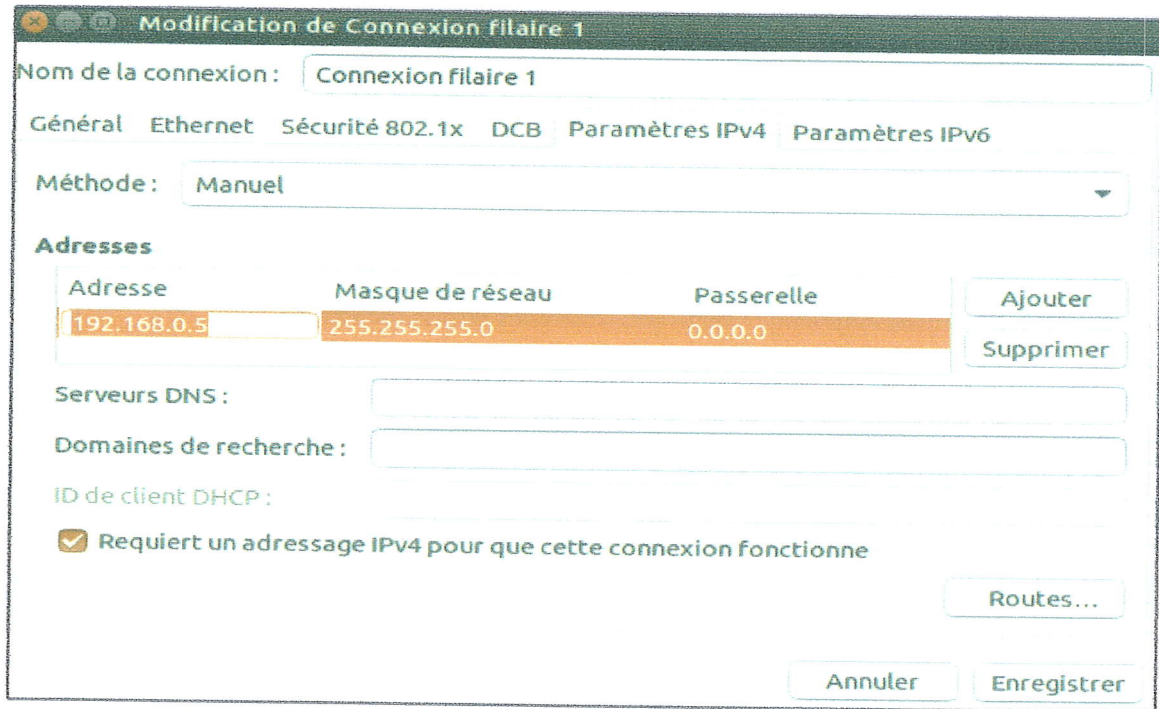


Figure C.3 La modification d'adresse IP de nœud master.

Cette étape se répète dans tous les nœuds avec les changements des adresses IP (voir tableau C.1).

Nœud	Adresse IP	Remarque
Master	192.168.0.5	Ces adresses IP sont les même adresses IP de configuration.
Slave1	192.168.0.9	
Slave2	192.168.0.6	
Slave3	192.168.0.10	
Slave4	192.168.0.11	

Tableau C.1 Tableau des adresses IP de chaque nœud.

Étape2 : Changement d'adresses IP des machines physique :

- Ouvrir le centre de réseau et partage
- connexion au réseau local

➤ saisi l'adresse IP manuellement :

La machine	Adresse IP
Machine1 (PC_Meryem)	192.168.0.7
Machine2 (PC_Massika)	192.168.0.8

Étape3 : Changement de réseau de chaque machine virtuelle

Cliquer avec le boutons droite sur le nom de la machine (par exemple master) paramètre, réseau, accès par pont, ok. (Voir la figure C.4).

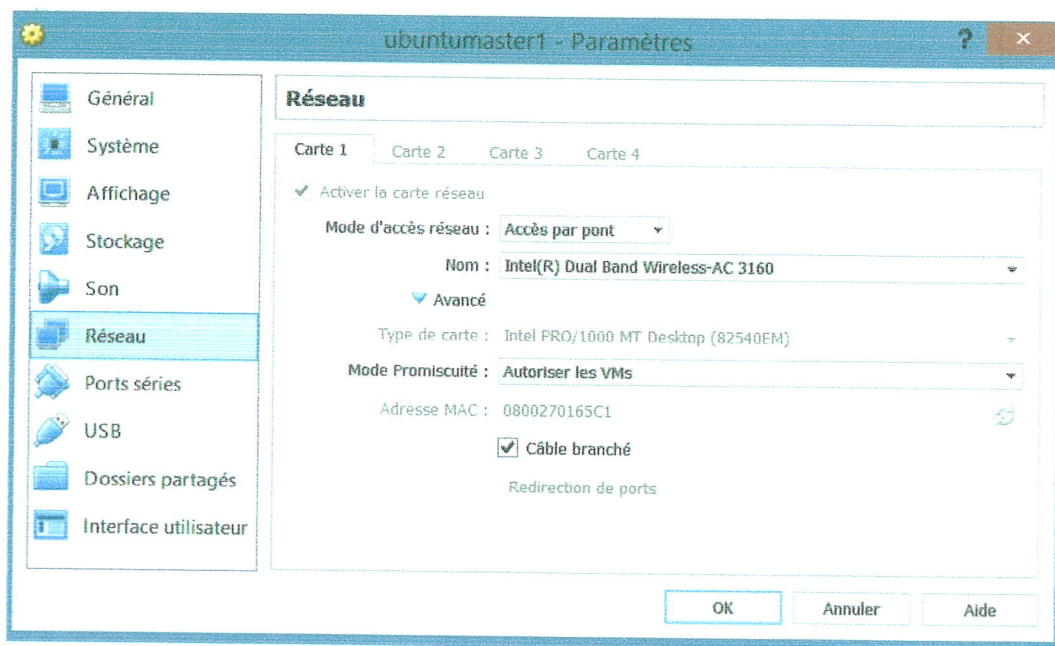


Figure C.4 Changement de network adapté.

Étape5 : Test de réseau :

Ping entre les machines par exemple : Ping entre la machine slave2(PC_Massika) à la machine master (PC_Meryem).

```
meryem@slave2:~$ ping master
PING master (192.168.0.5) 56(84) bytes of data:
64 bytes from master (192.168.0.5): icmp_seq=1 ttl=64 time=1.06 ms
64 bytes from master (192.168.0.5): icmp_seq=2 ttl=64 time=0.880 ms
64 bytes from master (192.168.0.5): icmp_seq=3 ttl=64 time=0.868 ms
64 bytes from master (192.168.0.5): icmp_seq=4 ttl=64 time=0.975 ms
64 bytes from master (192.168.0.5): icmp_seq=5 ttl=64 time=0.823 ms
^C
--- master ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/ndev = 0.823/0.921/1.062/0.092 ms
```

Ping entre la machine master (PC_Meryem) à la machine slave2 (PC_Massika)

```
meryem@master:~$ ping slave2
PING slave2 (192.168.0.6) 56(84) bytes of data:
64 bytes from slave2 (192.168.0.6): icmp_seq=1 ttl=64 time=1.22 ms
64 bytes from slave2 (192.168.0.6): icmp_seq=2 ttl=64 time=0.891 ms
64 bytes from slave2 (192.168.0.6): icmp_seq=3 ttl=64 time=0.837 ms
64 bytes from slave2 (192.168.0.6): icmp_seq=4 ttl=64 time=1.33 ms
64 bytes from slave2 (192.168.0.6): icmp_seq=5 ttl=64 time=0.749 ms
64 bytes from slave2 (192.168.0.6): icmp_seq=6 ttl=64 time=0.492 ms
64 bytes from slave2 (192.168.0.6): icmp_seq=7 ttl=64 time=0.751 ms
^C
--- slave2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6009ms
rtt min/avg/max/mdev = 0.492/0.897/1.331/0.269 ms
```

Étape6 : Installation de ssh dans chaque noeud de telle sorte qu'ils peuvent communiquer entre eux sans aucune invite de mot de passe.

```
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/id_rsa.pub nom_machine@ master
$ ssh-copy-id -i ~/.ssh/id_rsa.pub nom_machine @ slave1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub nom_machine @ slave2
$ ssh-copy-id -i ~/.ssh/id_rsa.pub nom_machine @ slave3
$ ssh-copy-id -i ~/.ssh/id_rsa.pub nom_machine @ slave4
$ chmod 750 ~/.ssh/authorized_keys
```

Étape9 : Démarrer tous les nœuds :

Dans la machine maître, on va démarrer les noeuds ont utilisant la même commande :

```
$ HADOOP_HOME/bin/start-all.sh
```

a. Résultat de démarrage de master.

```
meryem@master:~$ jps
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
5083 Jps
4688 SecondaryNameNode
4935 TaskTracker
4778 JobTracker
4542 DataNode
4390 NameNode
```

b. Résultat de démarrage de slave2.

```
meryem@slave2:~$ jps  
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar  
2777 TaskTracker  
2828 Jps  
2664 DataNode
```

Résumé

Aujourd'hui, les graphes sont devenus omniprésents. Ils sont utilisés pour modéliser divers types de réseaux : réseau Internet, réseau de transport, réseaux sociaux etc. Cependant ces graphes peuvent être de grande taille (Big-graph) et il devient difficile de les traiter efficacement sur une seule machine. Cela signifie qu'il y a un besoin pour les algorithmes qui peuvent être facilement parallélisés.

Au cours de ces dernières années et avec l'apparition du Cloud Computing, qui a fait une réelle évolution dans le monde des technologies de l'information et de la communication, des nouveaux paradigmes de traitement parallèle de larges graphes ont été proposés, tels que Pregel de Google et son implémentation open source Giraph.

Actuellement Giraph qui s'appuie sur le modèle BSP, popularisé par le projet Pregel de Google, est le plus utilisé. Il est conçu pour exécuter les algorithmes de graphes itératifs à travers des clusters de machines. L'objectif de ce travail est d'adapter et d'implémenter un algorithme de graphe séquentiel (énumération de cliques, composantes connexes,...) dans un environnement distribué Hadoop/Giraph.

Mots clés: Cloud Computing, Pregel, Giraph, Hadoop, Big-Graph, Algorithme.

Abstract

Today, the graphs have become ubiquitous. They are used to model various types of networks: Internet network, transmission network, social networking etc. However, these graphs can be large (Big-graph), and it becomes difficult to treat effectively on a single machine. This means there is a need for algorithms that can be easily parallelized.

In recent years and with the advent of cloud computing, which is a real evolution in the world of information technology and communication, new parallel processing paradigms for large graphs have been proposed, such as Pregel Google and its open source implementation Giraph.

Currently Giraph which supports the BSP model, popularized by the Pregel project Google is the most used. It is designed to run on iterative graphs through clusters of machines. The objective of this work is to adapt and implement a sequential algorithm (enumeration of cliques, related components,...) in a distributed environment Hadoop / Giraph.

Key words: Cloud Computing, Pregel, Giraph, Hadoop, Big-Graph, Algorithme.

