الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de 8 Mai 1945 – Guelma –
Faculté des Mathématique, d'Informatique et des Sciences de la matière
Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Système informatique

Thème :

Etude de Données NoSQL sur Hadoop/MapReduce

Encadré Par :

Dr. Aicha AGGOUNE

Présenté Par :

Mr. Hocine BELHAOUES

Juin 2018

Remerciement

Tout d'abord, je remercie Allah qui m'a donné la puissance, le courage et la détermination nécessaire pour finaliser ce travail.

Je tiens à exprimer ma profonde gratitude à mon encadreur Dr. Aicha AGGOUNE pour ses conseils judicieux et son encadrement qualifié qui m'ont permis d'améliorer grandement la qualité de ce mémoire. Qu'elle trouve ici l'expression de mon très grand respect et toute ma gratitude

Je tiens à remercier tous les enseignants, qui ont assuré notre formation durant notre cycle universitaire

Je tiens à remercier tous les personnes qui m'ont aidé et soutenu de près ou de loin, à réalisation ce travail

Dédicaces

Je dédie ce travail à :

Mes chers parents ma mère et mon père

Mes frères

Et tous les membres de ma famille

Ainsi que tous mes amis et collègues

À tous ceux qui m'ont enseigné

Résumé

Actuellement, les données ont connu une croissance exponentielle que les SGBD relationnels ne peuvent pas assurer le stockage et le traitement de ces données volumineuses. De ce fait, des nouveaux SGBD sont apparus dédiés à administrer de grandes quantités de données hétérogènes baptisées "Données NoSQL". Ce travail vise à étudier ce nouveau type de données "NoSQL" via l'environnement distribué Hadoop de fondation Apache sous Ubuntu server GNU/Linux. Le travail est divisé en deux grands volets : (i) apprendre et mettre en œuvre ce nouvel environnement qui assure le stockage et le traitement distribués de données NoSQL avec son paradigme de programmation MapReduce (ii) étudier et évaluer quelques types de données NoSQL en utilisant des SGBD NoSQL propre pour chaque type de données. Il s'agit de manipuler trois systèmes de gestion de données NoSQL: Hadoop avec son système de fichiers distribué HDFS, Hbase et MongoDB. L'étude réalisée montre l'efficacité et la performance de ce type de données.

Mots-clés: Données NoSQL, Hadoop, HDFS, MapReduce, HBase, MongoDB.

Abstract

Recently, the data has exponential growth that relational DBMSs can't ensure the storage and the processing of this large data. Indeed, the new DBMS appeared which dedicated to administrating large amounts of heterogeneous data called "NoSQL". This work aims to study this new type of data via Hadoop, a distributed environment of Apache foundation under Ubuntu server GNU / Linux. The work is divided into two main parts: (i) to learn and to implement this new environment that provides distributed storage and processing of NoSQL data with its MapReduce paradigm programming (ii) to study and to evaluate some kinds of NoSQL data by using DBMS NoSQL own of each type of data. It's about handling three data NoSQL management systems: Hadoop with its distributed file system HDFS, HBase, and MongoDB. The study carried out illustrates the efficiency and the performance of this type of data.

Keywords: NoSQL data, Hadoop, HDFS, MapReduce, HBase, MongoDB.

Table des Matières

Chapitre 01. Les Données NoSQL

Chapitre 02. Le framework Hadoop

Chapitre 03. Mise en œuvre et étude de données NoSQL

Liste des Figures

Chapitre 01. Les Données NoSQL

Chapitre 02. Le framework Hadoop

Chapitre 03. Mise en œuvre et étude de données NoSQL

Liste des Tables

Chapitre 01. Les Données NoSQL

Chapitre 03. Mise en œuvre et étude de données NoSQL

INTRODUCTION GENERALE

Introduction générale

1. Contexte du travail

Ces dernières décennies, les données qui circulent sur le web ou sont utilisées dans des grandes entreprises deviennent de plus en plus volumineuses et hétérogènes. Le stockage de très grandes quantités de données pose des problèmes d'analyse et de traitement qui peuvent aller du simple ralentissement du système, jusqu'à la panne complète. Ainsi, le temps d'accès aux ressources devient trop long et certaines analyses sont trop complexes pour être réalisée par les SGBD relationnels. Le volume important de ces données amène souvent à les gérer par des nouveaux SGBD dédiés à administrer de grandes quantités de données, c'est ce que les Anglo-Saxons ont appelé le Big Data [4]. Le stockage et le traitement de big data est impossible par la plupart des SGBD relationnel actuels [12]. C'est dans ce contexte, l'apparition des nouveaux types des SGBD appelés SGBDs NoSQL qui manipulent de big data qui se nomment également les données NoSQL (Not Only SQL).

Les données NoSQL sont un sujet très à la mode, issues du terme big Data. Elles sont souvent volumineuses et structurellement hétérogènes [19]. Contrairement aux bases de données relationnelles (BDR), les bases de données NoSQL n'imposent pas de schéma stable et ne présente pas la valeur Null qui prend un espace considérable. Il existe quatre types de bases de données NoSQL [4, 7, 29, 30] :

- **Les bases de données clé-valeur :** c'est le type de données NoSQL le plus simple dont l'exécution est le plus rapide et ses données sont représentées par un couple clé-valeur. Une valeur peut être de n'importe quel type où chaque valeur stockée est associée à une clé unique qui permet d'exécuter des requêtes non SQL sur la valeur.
- **Les bases de données orientées colonnes :** Elles sont semblables aux tables relationnelles d'une base de données sauf que les données sont stockées par colonne, non par ligne. Les colonnes sont dynamiques, elles peuvent être variées d'une ligne à un autre.
- **Les bases de données orientées document :** c'est le type de données le plus adapté au monde de l'internet. Ces données sont proches de bases de données clé-valeur mais la valeur est un document en format BISON, JSON ou XML.
- **Les bases de données orientées graphe :** Ces bases sont fondées sur la théorie des graphes pour la modélisation des données complexes reliées par des relations. Ce modèle est composé d'un moteur de stockage de nœuds et un mécanisme de description de relations et de propriétés qui leur sont rattachées.

Toutes ces bases de données NoSQL vont être gérées par un SGBD propre pour elles. Il existe sur le marché plus de 225 SGBD NoSQL [4]. De plus, la mise en place d'un environnement distribué est indispensable pour assurer à la fois le stockage et le traitement distribués de données NoSQL. L'un des outils les plus puissants pour assurer la gestion distribuée de données NoSQL est le Framework Hadoop avec son système de fichiers distribué HDFS qui permet d'assurer le stockage et l'intégrité de ces données [5].

2. Problématique abordée

Les bases de données NoSQL représentent une solution idéale pour répondre à de nouvelles problématiques liées à la gestion de données dans des environnements distribués à grande échelle. Ces bases de données sont classifiées en quatre catégories ce qui nous permet de les examiner et les étudier avant d'être utilisées dans les domaines et les secteurs qui manipulent quotidiennement des volumes de données très important tels que la santé, le marketing, la bioinformatique, etc.

3. Objectif du travail

L'objectif de ce mémoire est double. Il s'agit, d'une part, de comprendre et mettre en œuvre le framework Hadoop pour le stockage et le traitement des données dans un environnement distribué via des clusters. Et d'autre part, d'effectuer une étude sur les données NoSQL en utilisant ce framework avec d'autres SGBDs NoSQL.

4. Organisation du mémoire

Ce mémoire est constitué de trois chapitres : deux chapitres s'intéressent à l'état de l'art sur les données NoSQL et le framework Hadoop et un chapitre présente notre contribution.

Chapitre 01 : Les données NoSQL

Ce premier chapitre décrit dans un premier lieu les évolutions des bases de données puis nous introduisons les notions de base sur les données volumineuses qui sont : le Big Data, Internet of things et Cloud Computing. Une grande partie du chapitre est consacrée à la présentation de données NoSQL, nous commencerons par les limites de bases de données relationnelles, puis nous passerons par les caractéristiques de données NoSQL, leurs types, leur architectures et nous terminerons par la solution Big Table de Google comme exemple de base de données volumineuse et distribuée.

Chapitre 02 : Le Framework HADOOP

Le contenu de ce chapitre présente en détail les principaux composants du framework Hadoop avec son architecture et son paradigme de programmation MapReduce. Le reste du chapitre présente les SGBD NoSQL choisi pour effectuer notre contribution dans le chapitre suivant.

Chapitre 03 : Mise en œuvre et étude de données NoSQL

Le dernier chapitre est consacré à la présentation de notre contribution qui vise à mettre en œuvre les données NoSQL en utilisant le framework Hadoop et d'autres SGBD NoSQL. Il sert à étudier et évaluer l'utilisation de données NoSQL. Une analyse des différents résultats sont aussi décrites.

Nous terminerons ce mémoire par une conclusion générale, dans laquelle on va récapituler notre contribution et nous envisagerons quelques perspectives pour les futurs travaux.

Chapitre 01

Les Données NoSQL

1. Introduction

Dans ces dernières décennies, les recherches dans le domaine de bases de données sont concentrées sur la manière d'organiser les données, potentiellement de grandes quantités, afin d'en optimiser le stockage et la recherche. Cette évolution croissante de données rend leur manipulation presque impossible pour les bases de données relationnelles ainsi que les SGBDR qui ont atteignent vite leurs limites, ce qui nécessite de trouver une solution pour stocker et gérer ces données de grandes entreprises. Une solution adoptée par les développeurs de base de données qui consiste à développer plusieurs produits distincts. Ces produits visent à manipuler d'une façon distribuée les données volumineuses, c'est ce que les Anglo-Saxons ont appelé le Big Data. C'est dans ce contexte, l'apparition des nouveaux types des SGBD appelés SGBD NoSQL manipulent de big data qui se nomment également les données NoSQL (Not Only SQL).

Dans ce chapitre, nous présentons une brève évolution des bases de données. Par la suite, nous évoquerons l'émergence de Big data, d'internet of things et la business intelligence. Le reste du chapitre décrit les données NoSQL et la technique de traitement distribué de données NoSQL, plus particulièrement le paradigme MapReduce de Google.

2. Les évolutions des bases de données

Une base de données est une collection de données stockées, organisées et structurées pour être manipuler par un système de gestion de données [17]. Elle permet de mettre des données à la disposition d'utilisateurs pour une consultation, une saisie et une mise à jour. Le terme Database, l'expression anglaise qui est apparu en 1964 pour désigner une base de données (BD) [17].

Dès 1950 apparaît l'idée de construire des systèmes qui sont capable de traiter et organiser les données, ce qui rend les bases de donnée plus flexible et facile à manipuler. Vers la fin des années 1960, on commence à voir l'apparition des premiers SGBD (Système de Gestion de Base de Données) en anglais DBMS (Database Management System) qui sont un ensemble de logiciels prenant en charge la structuration de la base de données, le stockage, la mise à jour et la maintenance des données. Autrement dit, un SGBD permet de décrire, modifier, interroger et administrer les données. C'est, en fait, l'interface entre la base de données et les utilisateurs. [1]

Le modèle hiérarchique est le premier type de modélisation de données sur un système informatique, né dans les années 1950 [7]. Ainsi, une base de données hiérarchique est composée d'enregistrement qui contiennent des champs et des relations qui sont créées entre types d'enregistrements parents et types d'enregistrements fils, ce qui forme un arbre [7].

En 1960, le modèle réseau est apparu pour améliorer le modèle hiérarchique par l'intégration d'un langage de navigation « COBOL » (COmmon Business Oriented Language) [1]. Il est basé sur la navigation dans les entités via les pointeurs [1].

Très vite ces systèmes on aboutés à des limites, dans le fait qu'ils sont incapable de structurer des modèles complexes et de répondre aux besoins réels et de suivre l'évolution des systèmes d'information.

Ce n'est qu'à partir de 1970 qu'apparait la deuxième génération de SGBD : les systèmes relationnels. Edgar Frank Codd est un informaticien britannique qui propose en 1970 le modèle relationnel de données [8]. Ce modèle est basé sur le stockage de valeurs des données dans des tables ou relations et la manipulation de ces données à l'aide d'un langage basé sur l'algèbre relationnelle, sans se préoccuper du stockage physique des données [8]. Ce langage est le SQL (sigle de Structured Query Language, en français langage de requête structurée) qui est un langage informatique normalisé servant à exploiter des bases de données relationnelles et manipule des données (rechercher, ajouter, modifier ou de supprimer des données dans les bases de données) d'une façon déclarative [10].

La première version de SQL est apparue en 1986 par le groupe de normalisation ANSI X3. 135-1986 et internationalisée au niveau de l'ISO par le groupe ISO 9075:1987. Le SQL1 propose deux niveaux : SQL1-87 et SQL1-89 par l'ajout d'autres fonctionnalités telles que les contraintes d'intégrités [3]. En 1992, l'apparition de la deuxième version de SQL appelé SQL2, qui donne une solution aux lacunes de SQL1 par la possibilité de modifier et supprimer le schéma par les commandes Alter et Drop respectivement. De plus, à partir de SQL2, il est possible de manipuler les vues et les déclencheurs [3].

Le SQL3 c'est une extension de SQL2 développée par le groupe de normalisation ANSI X3 H2 et internationalisé au niveau de l'ISO par le groupe ISO/IEC JTC1/SC21/WG3 [9]. Il est adopté en 1999 des nouveaux aspects, tels que l'intégration de l'objet au relationnel et l'intégration de multimédia via les types de larges objets, appelés les types LOB (Large Object) [9].

Les SGBD relationnels ont pris une place importante sur le marché. Ils répondent de façon satisfaisante aux besoins des applications classiques de gestion à titre d'exemple, gestion de pharmacie, gestion de stock, gestion de comptes bancaires, etc.

Une extension du modèle relationnel appelée Modèle Objet relationnel qui a pour but de remédier les limites du relationnel par l'intégration de l'approche orientée objet dans la définition et la manipulation des tables relationnelles [21].

L'inconvénient majeur du modèle objet relationnel réside dans le fait qu'il présent deux niveaux d'abstraction différents : la relation et l'objet.

D'autres évolutions de bases données ont été présentées, nous nous basons sur le cours de bases de données avancées afin de décrire brièvement ces évolutions [21].

Le modèle Oriente Objet a été proposé dont les éléments de la base données orientée objet BDOO sont des objet qui puissent être gérés par un système de type SGBDOO (Ex. O2). Les BDOO sont caractérise par : Structure de données complexe, les données et le traitement ne sont plus séparés, L'héritage et chaque objet a son propre identité.

Le modèle Objet relationnel représente le modèle le plus répandu au monde avec des outils très performants, néanmoins ce modèle ayant des critiques majeurs : difficulté d'exprimer la sémantique des données et d'intégrer des bases de données hétérogènes. Une solution consiste à attribuer aux bases de données décrivent le même domaine, une description sémantique sous forme de connaissances (Knowledge).

Parmi des nombreux modèles sémantique existants, base de données à base Ontologies (BDBO) qui vise à définir une sémantique aux données. Une BDBO présente deux caractéristiques : i) Gérer à la fois des ontologies et des données, ii) associer à chaque données le concept ontologique qui en définit leur sens.

Avec l'évolution et l'explosion de la quantité d'information stockée dans les bases de données, plusieurs problèmes sont apparus pour gérer ces masses de données. De ce fait, il est nécessaire de mettre en œuvre des nouvelles techniques permettant de faciliter le stockage et la manipulation de ce type de données.

Une nouvelle technique apparait sous le nom Base de données déductive ou logique, idée est de coupler une base de données à un ensemble de règles logiques qui permettent d'en déduire de l'information.

Une base de données déductive est constituée, de : i) une base de données extensionnels dont l'extension est conservée explicitement dans la base de données et ii) une base de données intentionnelle dont l'extension est définie par des règles de déduction.

Une base de données multimédia (BDMM) est consacrée spécialement au stockage et au traitement de contenu de données multimédias (texte, audio, image, vidéo). Les BDMM attirent l'attention des développeurs et des chercheurs dont le but d'assurer une représentation intégrale de ces données volumineuses au moyen des nouveaux types de données complexes utilisées dans la plupart des SGBD, appelés les types LOBs (Large Object) [22] [23].

Le modèle relationnel bien que reste le modèle le plus puissant et le plus utiliser dans le monde professionnel, beaucoup de données ne peuvent pas être représentées et manipulées par ce modèle à cause de leur structure complexe, hétérogènes, variables et volumineuses tels que celles de Google, Facebook, Yahoo, etc. Ces caractéristiques de données tendent à émerger un nouveau type de données appelé données NoSQL. Ces données fait partie de la notion de Big data ou grosse données.

Nous allons présenter dans le reste de ce chapitre la notion de Big data et ses concepts relatifs puis une description détaillée de données NoSQL avec ses propriétés fondamentales.

3. L'émergence du Big Data

Le Big Data est une histoire récente, il est apparu pour désigner un ensemble de données volumineuses qui sont difficile à définir et à manipuler par le modèle relationnel ou objet relationnel. Dans ce contexte, trois notions sont étroitement liées à ce type de données qui sont : big data, internet of things et business intelligence.

3.1. Le Big Data

Le terme Big Data ou grosse données a été utilisé à l'origine par Doug Laney dans son rapport de recherche en 2001 [24]. Au début, ce terme se définie par trois caractéristiques fondamentales nommées 3V qui sont : le volume, la vélocité et la variété. Par la suite, cette définition a été améliorée par l'ajout de deux autres caractéristiques (véracité et valeur) ce qui présente le big data par cinq propriétés fondamentales baptisées 5V [16].

Jason Bloomberg définie le terme big data comme suit: *"Big Data: a massive volume of both structured and unstructured data that is so large that it's difficult to process using traditional database and software techniques."* [25]. En d'autre terme, le big data désigne le volume important de données structurées et non structurées qui sont tellement volumineuses, il est difficile de les traiter en utilisant des bases de données traditionnelles et des techniques logicielles. Les 5V de big data sont [16] :

- **Volume :** Le volume fait référence aux grandes quantités de données générées chaque secondes. Le big data offre des outils pour stocker, accéder, et analyser ces données volumineuses.
- **Vélocité :** La vélocité est la vitesse à laquelle la donnée est générées et se modifiées. Le big data doit d'être performant pour analyser la donnée.
- **Variété :** C'est la diversité des formats des données et l'hétérogénéité de type de données. Plus de 80 % de la donnée est qualifiée de 'non-structurée', comme les courriels, photos, vidéos, sons, messageries, etc. Le big data offre la capacité de réunir toutes ces données et de les analyser.
- **Véracité :** La véracité est le degré de vérité de données qui sont parfois de mauvaise qualité et non exactes.
- **Valeur :** qui fait référence à la valeur de données proprement dites, il s'agit d'être capable de se concentrer sur les données ayant une réelle valeur.

La figure suivante résume les propriétés 5V de big data.

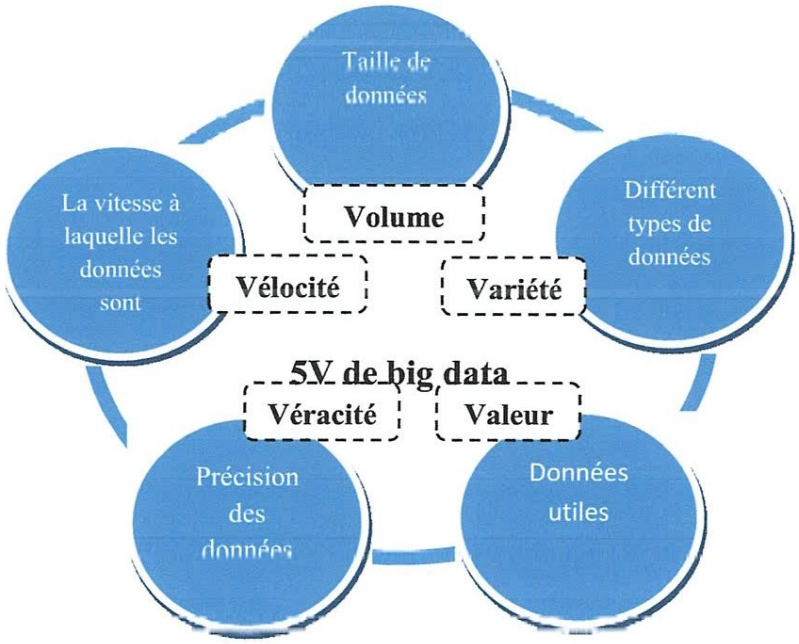


Figure 1.1: *Les 5V de BigData [16]*

Au début de l'apparition de Big Data, on a cru que c'était difficile à gérer et à manipuler, toutefois, il présente un certain nombre d'avantages qui rendent son utilisation efficace. Deux principaux avantages qui sont :

- La capacité de traiter des grands volumes de données.

- La prise de décision facile en ayant une vision avec certitude.

3.2. Internet of Things (IoT)

A partir de l'an 2003, de nombreux appareils connectés à Internet, communiqués avec leur environnement et échangés des données, qui offrent aux personnes de plus en plus de services facilitant leurs activités [26]. Ces appareils sont des objets qui peuvent être une simple carte à puce, les smartphones, ordinateurs, tablettes et télévisions connectées qui sont capables d'aller chercher des contenus sur Internet. L'accroissement du nombre d'appareils connectés sur internet par rapport le nombre de personnes donne lieu à une nouvelle notion appelée Internet des objets ou en anglais Internet of Things le terme qui est le plus souvent utilisé.

Internet of Things ou IoT, est considéré comme la troisième évolution de l'Internet web 3.0. Celle-ci donnera lieu à des applications révolutionnaires capables de transformer profondément notre mode de vie, et notre façon d'apprendre, de travailler et de nous divertir. Internet des objets connectés représente les échanges d'informations et de données provenant de dispositifs du monde réel avec le réseau Internet [14]. L'IoT correspond au moment où il y a eu plus « de choses ou d'objets » connectés à Internet que de personnes [26].

L'IoT est donc un réseau de réseaux qui permet, via des systèmes et des dispositifs d'identification électronique normalisés et unifiés de pouvoir récupérer, stocker, transférer et traiter, sans discontinuité entre les mondes physiques et virtuels, les données s'y rattachant [13].

3.3. Business intelligence (BI)

La business intelligence est le terme le plus utilisé que son traduction en français par l'informatique décisionnelle. Elle désigne les moyens, les outils et les méthodes qui permettent de collecter, consolider, modéliser et restituer les données d'une entreprise en vue d'offrir une aide à la décision et de permettre à un décideur d'avoir une vue d'ensemble de l'activité traitée [15].

La business intelligence repose sur une architecture basée sur 3 concept théoriques qui se viennent de R. Kimball, B. Inmon et D. Linstedt [15] :

1. **La collection de données** : les données sont issues de sources hétérogènes (fichiers Excel, base de données, service web, données massives) et stockées dans un entrepôt de données.

2. **La restructuration de données**: les données doivent être restructurées, enrichies et agrégées pour permettre aux décideurs d'interagir avec les données sans avoir à connaître leur structure de stockage physique.
3. **La diffusion de données**: les données collectées et restructurées peuvent être utilisées dans divers services d'entreprise tels que finance, production, comptabilité, gestion de ressources humaines via un système d'information.

4. Vers les données NoSQL

Les données qui circulent sur le web ou sont utilisées dans des grandes entreprises deviennent de plus en plus volumineuses et hétérogènes. C'est dans ce contexte, un nouveau domaine technologique a vu le jour appelé le Big Data qui est inventé par les géants du web, au premier rang comme Yahoo, Google et Facebook. Le stockage de très grandes quantités de données pose des problèmes d'analyse et de traitement qui peuvent aller du simple ralentissement du système, jusqu'à la panne complète. Ainsi, le temps d'accès aux ressources devient trop long et certaines analyses sont trop complexes pour être réalisée par les SGBD relationnels. En effet, des nouveaux types des SGBD sont capables de gérer le big data appelés SGBD NoSQL. Ainsi, leurs données se nomment également les données NoSQL (Not Only SQL).

Dans cette section, nous allons présenter dans un premier lieu les lacunes de bases de données relationnelles, par la suite nous présenterons la notion de données NoSQL, leurs caractéristiques et leur types.

4.1. Les lacunes des bases de données relationnelles

Le modèle relationnel existe maintenant depuis 48 ans en tant que le modèle de données le plus puissant, ayant les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité) garantissant la fiabilité des transactions. Une transaction atomique (Atomicité) est un ensemble d'opérations qui doit être valide intégralement ou totalement annulée. Elle doit maintenir le système en cohérence (La propriété de Cohérence) par rapport à son état (validés ou annulée). Elle isole ces données des autres transactions (Isolation) de façon à ce qu'elle ne puisse pas lire des données en cours de modification. Enfin, lorsqu'une transaction est validée, le nouvel état est durablement inscrit dans le système (La durabilité) [4].

Par ailleurs, les données de grandes sociétés telles que Yahoo, Google et Facebook ne peuvent pas être stockées par une simple base de données relationnelle et traitées par une simple transaction. Les bases de données relationnelles ne peuvent pas gérer ce volume important de données. Dû aux propriétés importantes de ce type de données, le modèle relationnel connait quelques lacunes qui sont présentées dans le reste de cette sous-section.

4.1.1. La scalabilité de données

Le terme scalabilité ou en anglais Scalability désigne l'aptitude d'un produit ou d'un système de conserver ses propriétés fonctionnelles malgré un changement de taille ou de certains paramètres (par exemple l'augmentation du nombre d'utilisateurs pour un serveur) [27]. Dans le cas de données volumineuses, la scalabilité de base de données est limitée par une simple distribution qui ne permet pas facilement la flexibilité et la cohérence. En plus, pour la sécurité de fonctionnement, il faut que les données soient répliquées, ce qui complique encore un peu plus leur utilisation. Les SGBD relationnels ne sont pas adaptés aux environnements distribués requis par le volume important de données, la seule solution est d'appliquer une segmentation de tables relationnelles [6]. C'est donc une segmentation intelligente qui permet une montée en charge, mais cette architecture montre tout de même des limites, c'est là que le NoSQL entre en jeu.

4.1.2. La structure de données

La volumétrie et l'hétérogénéité de données entraînent la multiplication des relations, et par conséquent des opérations de jointure lors du traitement des requêtes. Ces opérations engendrent des requêtes complexes faisant intervenir plusieurs entités (les tables en l'occurrence) [4]. L'intégrité référentielle permet de s'assurer que les liens entre les entités sont valides. La mise en œuvre de tels mécanismes a un coût considérable dès lors que l'on se trouve dans de gros volumes de données.

4.1.3. La cohérence de données dans un environnement distribué

Les propriétés ACID d'une base de données bien que importantes dans un modèle relationnel, elles posent un grand problème de performances, de complexité et de cohérence de données dans un environnement distribué. Deux problèmes majeurs sont apparus [7]:

i. Le stockage : la redondance des données car chaque donnée est répliquées sur chaque serveur.

ii. Le coût : lié à l'insertion, la modification et la suppression est très grand, car on ne peut valider une transaction que si on est certain qu'elle a été effectuée sur tous les serveurs et le système fait patienter l'utilisateur durant ce temps.

En 2002, Seth Gilbert et Nancy Lynch du MIT (Massachusetts Institute of Technology) ont conclu qu'il n'est pas possible de réaliser un système qui soit à la fois ACID et distribué [4]. La méthode consiste donc à distribuer et dupliquer les données sur plusieurs serveurs, ainsi augmenter les performances et résister aux pannes.

4.1.4. La limite de théorème de CAP

Le professeur Eric Brewer de l'université de Californie à Berkeley fit une présentation intitulée «Towards Robust Distributed Systems» dans le symposium sur les principes de l'informatique distribuée (Principles of Distributed Computing, PODC) organisé par l'ACM [28]. Brewer est venu avec un théorème de CAP, l'acronyme de trois propriétés de base de données distribuées qui sont : la cohérence (Consistency), la disponibilité (Availibility) et la résistance au partitionnement (Partition tolerance) [28].

- **Consistency** : Tous les nœuds du réseau voient exactement les mêmes données, même lorsqu'il y a des mises-à-jour.
- **Availability** : Les données d'un système distribué sont dupliquées ce qui garantit que toutes les requêtes reçoivent une réponse.
- **Partition tolerance :** La capacité d'un système à faire face à l'ajout et à la suppression dynamique de nœuds qui sont considérés comme une partition réseau propre qui doivent pouvoir fonctionner de manière autonome.

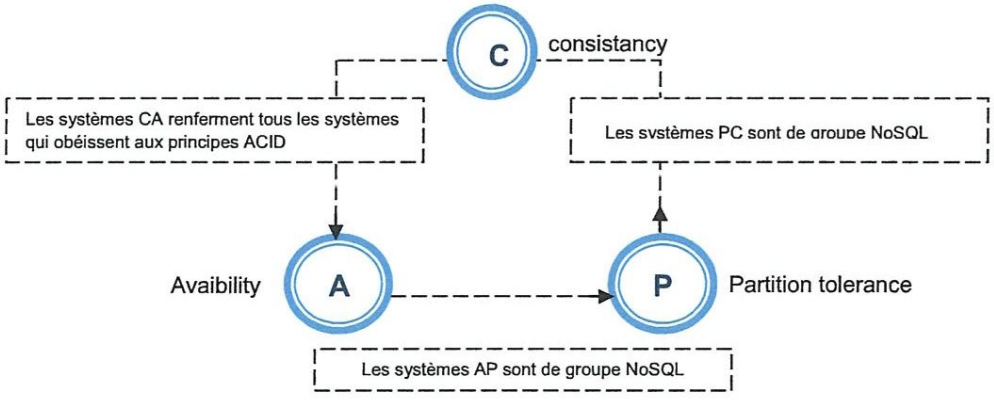


Figure 1.2: *Le Théorème de CAP [28]*

Le théorème de CAP dit que seules deux de ces trois contraintes peuvent être respectées à un instant T car elles se trouvent en opposition [28]. Les bases de données relationnelles adhèrent

aux deux premières propriétés (la consistance et la disponibilité) qui sont de propriété ACID, tandis que les bases de données NoSQL se concentrent sur les deux dernières (la disponibilité et la résistance au partitionnement).

4.2. Définition de données NoSQL

L'étude de données NoSQL (Not only SQL) est un sujet très à la mode, issues du la technologie big Data. Les données NoSQL sont souvent volumineuses et structurellement hétérogènes [29]. Elles peuvent être de données structurées comme les bases de données relationnelles, semi structurées comme les documents et non structurées comme les données multimédias. En effet, les bases données NoSQL sont capable de stocker et de restituer des données volumineuses, hétérogènes sans utiliser un modèle mathématique solide comme les bases de données relationnelles [10].

Les bases de données NoSQL ne vient pas remplacer les bases de données relationnelles mais proposer une alternative pour résoudre les problèmes liés au Big data et d'améliorer la performance notamment dans le temps de réponse qui doit être proportionnel au volume de données. De plus, elles sont scalables pour le traitement distribué de données volumineuses et elles ne sont pas respectées les propriétés ACID. Les bases de données NoSQL suit le théorème CAP, elles privilégient la disponibilité à la cohérence.

L'utilisation de données NoSQL introduit l'acronyme BASE qui correspond aux propriétés suivantes [29]:

- **Basically Available** (disponibilité basique) : selon le théorème de CAP, les données sont essentiellement disponibles.
- **Soft state** (état doux): indique que l'état du système peut changer avec le temps, même sans entrée. C'est à cause du modèle de cohérence éventuel.
- **Eventually consistant** (éventuellement consistant) : une donnée répliquée peut, à un instant donné, ne pas avoir la même version sur les différents nœuds.

4.3. Les types de données NoSQL

Contrairement aux bases de données relationnelles, les données NoSQL n'a pas un schéma de données conceptuel et non pas un type à part entière. Le NoSQL regroupe quatre (4) grandes familles de base de données.

4.3.1. Les bases de données clé-valeur :

Les bases de données clé-valeur est le type de données NoSQL le plus simple dont l'exécution est le plus rapide et ses données sont représentées par un couple clé-valeur. Une valeur peut être de n'importe quel type où chaque valeur stockée est associée à une clé unique qui permet d'exécuter des requêtes non SQL sur la valeur [4] [30]. Une base de données clé/valeur convient parfaitement à un accès aux données au moyen d'une clé, comme dans le cas de la recherche d'un livre d'après son numéro, le numéro est la clé et toutes les autres informations concernant le livre sont la valeur. La clé est obligatoirement connue. En revanche, la valeur est un ensemble de données sans signification qui doivent être interprétées après lecture [18]. Ce type de base de données NoSQL implique donc des cas d'utilisation très spécifiques tels que : la détection de fraude en temps réel, IoT, e-commerce, gestion de cache, fichiers de logs, chat, etc. Des examples des SGBD de base de données clé-valeur : Riak, Redis et Voldemort de LinkedIn.

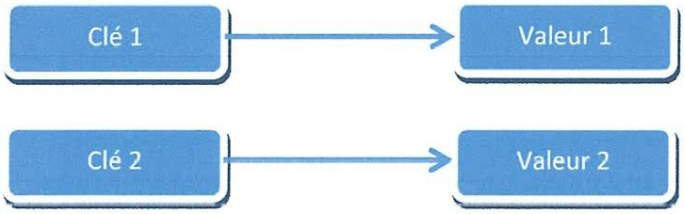


Figure 1.3: *Bases de données clé-valeur [30]*

Les bases de données clé-valeur utilisent 4 opérations CRUD [30]:

- create(clé,valeur) : crée un couple (clé,valeur)
- read(clé) : lit une valeur à partir de sa clé
- update(clé,valeur) : modifie une valeur à partir de sa clé
- delete(clé) : supprime un couple à partir de sa clé

4.3.2. Les bases de données orientées colonnes :

Les bases de données orientées colonnes sont semblables aux tables relationnelles d'une base de données sauf que les données sont stockées par colonne, non par ligne. Les colonnes sont dynamiques, elles peuvent être variées d'une ligne à un autre [10]. Il n'existe pas de schéma fixe ; autrement dit les noms et les clés des colonnes peuvent varier. Une base de données orientée colonnes convient aux données faisant l'objet de peu d'écritures, pour lesquelles les propriétés ACID ne sont pas impératifs et dont le schéma est variable [10]. Cassandra et HBase sont deux exemples des bases données orientées colonnes.

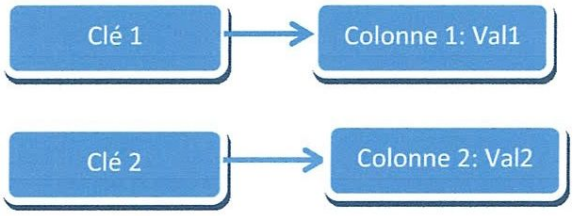


Figure 1.4: *Bases de données orientées colonnes [10]*

Le schéma d'une base de données orientée colonnes montrent que le stockage de données en colonne permet de gagner un espace considérable, spécialement lors des stockages des tuples incomplets contenant des valeurs *« null »*. La figure suivante illustre une comparaison entre le schéma relationnel et le schéma orienté colonne.

ID	Prénom	Opérateur	Localité
1	Alain		Meyrin
2	Adriano	Swisscom	
3	Sébastien		
4			Carouge
5			Onex

Schéma: données d'une base de données relationnelle

Prénom	Opérateur	Localité
Alain(1)	Swisscom(2)	Meyrin(1)
Adriano(2)		Carouge(4)
Sébastien(3)		Onex(5)

Schéma: données d'une base de données orientée colonnes

Figure 1.5: *Comparaison entre le schéma relationnel et le schéma orienté colonne [30].*

4.3.3. Les bases de données orientées document :

Les bases de données orientées document est le type de données le plus adapté au monde de l'internet. Ces données sont proches de bases de données clé-valeur mais la valeur est un document en format JSON (JavaScript Object Notation) ou XML [30]. Une base de données orientée documents peut être interrogée sur tous les composants du schéma défini, alors qu'une base de données clé/valeur ne peut être interrogée que sur sa clé [20]. L'avantage des bases de données orientées documents est de pouvoir récupérer un ensemble d'informations structurées hiérarchiquement depuis une clé contrairement au modèle relationnel où on a besoins d'effectuer plusieurs jointures entre tables [30]. MongoDB, CouchDB et MarkLogic sont des exemples de SGBD orientées document.

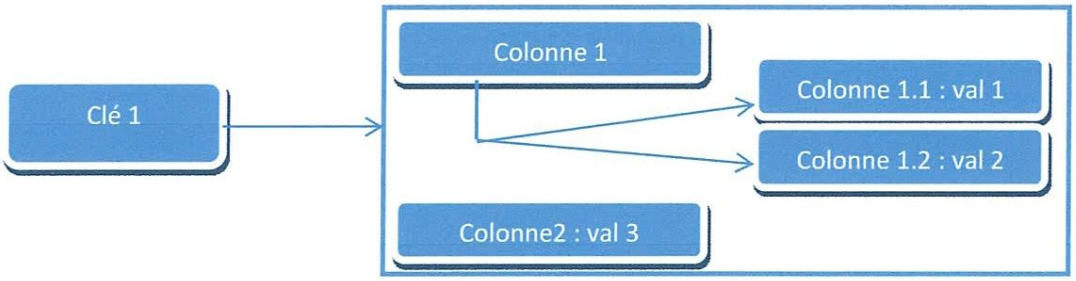


Figure 1.6: *Bases de données orientées document [30]*

4.3.4. Les bases de données orientées graphe :

Ces bases sont fondées sur la théorie des graphes pour la modélisation des données complexes reliées par des relations. Ce modèle est composé d'un moteur de stockage de nœuds et un mécanisme de description de relations et de propriétés qui leur sont rattachées [4]. Neo4j, HyperGraphDB et FlockDB sont des bases de données orientées graphe. Les réseaux sociaux (Facebook, Twitter,etc) sont un bon exemple pour présenter et résoudre le problème de complexité associés à des millions d'utilisateurs qui sont reliés entre eux de différentes manières.

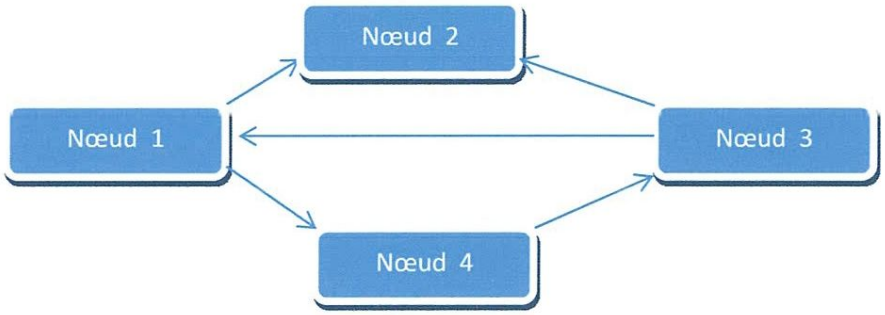


Figure 1.7: *Bases de données orientées graphe [4]*

Les SGBD NoSQL sont de plus en plus utilisés pour les mises en œuvre à l'échelle du Web et du Big Data. Chaque catégorie de SGBD NoSQL est adaptée à des types d'applications et d'utilisations distincts, ce qui a donné naissance à un nouveau terme, fréquemment employé dans la communauté NoSQL : la persistance polyglotte [18]. Il s'agit de recourir à différents systèmes de bases de données pour différentes applications et utilisations, en fonction de la manière dont la base de données gère les besoins de l'application.

5. Traitement distribué de données NoSQL

Dans le monde du NoSQL il n'y a pas de langage standard comme SQL. L'interrogation des bases de données NoSQL se fait au niveau applicatif à travers principalement le paradigme de programmation parallèle baptisé MapReduce. Le MapReduce est popularisé par Google, il est principalement utilisé pour la manipulation et le traitement d'un nombre important de données au sein d'un cluster de nœuds [6]. Il est très utilisé dans le milieu NoSQL et qui vise à produire des requêtes distribuées. En effet, nous allons présenter dans un premier temps la solution google pour la gestion de données volumineuses et les différents sorts d'architecture distribuée, et dans un second temps, nous présenterons le paradigme MapReduce.

5.1. La solution Google

Avec l'évolution Google a affronté le problème de gestion et de manipulation de grands volumes de données, c'est pourquoi elle a cherché une solution visant à ressouder ce grand problème en relevant ce défi. Google doit non seulement gérer un volume extrêmement important de données afin d'alimenter son moteur de recherche, mais aussi alimenter ses différentes offres, comme Google Maps, YouTube, Gmail, etc. [3]

Google a développé un système de fichiers distribué nommé GoogleFS, pendant plus de dix ans ce système est resté propriétaire et utilisé seulement chez Google. En 2003 déjà, le système de fichiers distribué de Google a été reconnu mondialement et avait fait ses preuves pour offrir un environnement de stockage redondant, fiable et résistance, fonctionnant sur un cluster constitué d'un grand nombre de machines de moyenne puissance (*commodity hardware*) [3].

Fin 2003, au symposium de l'ACM *(Association for Computing Machinery)* sur les principes de systèmes d'exploitation à Lake George, des ingénieurs de Google (Sanjay Ghemawat, Howard Gobioff et Shun-Tak Leung) fient une présentation de Google FS, intitulée *The Google File System* [3].

En 2004, à l'occasion du sixième symposium OSDI *(Operating System Desing and Implementation)* à San Francisco, Jeffrey Dean et Sanjay Ghemawat présentèrent une autre solution technique de programmation. Le titre de la présentation était *MapReduce,* il s'agissait, en plus de stocker les données sur GoogleFS (GFS), de pouvoir effectuer des traitements sur ces données de façon distribuée. Cette solution est inspirée des langages fonctionnels et basée sur deux primitives, les fonctions Map et Reduce. La fonction Map permet d'effectuer un traitement sur une liste de valeurs, en le réalisant sur GFS et en

regroupant les résultats grâce à la fonction Reduce. Google avait réussi à bâtir un environnement de traitement distribué qui lui a permis de résoudre un grand nombre de problèmes [3]. D'après la figure ci-dessous, un fichier est un ensemble des partitions contiguës des chunks qui sont répliquées sur différent nœuds.

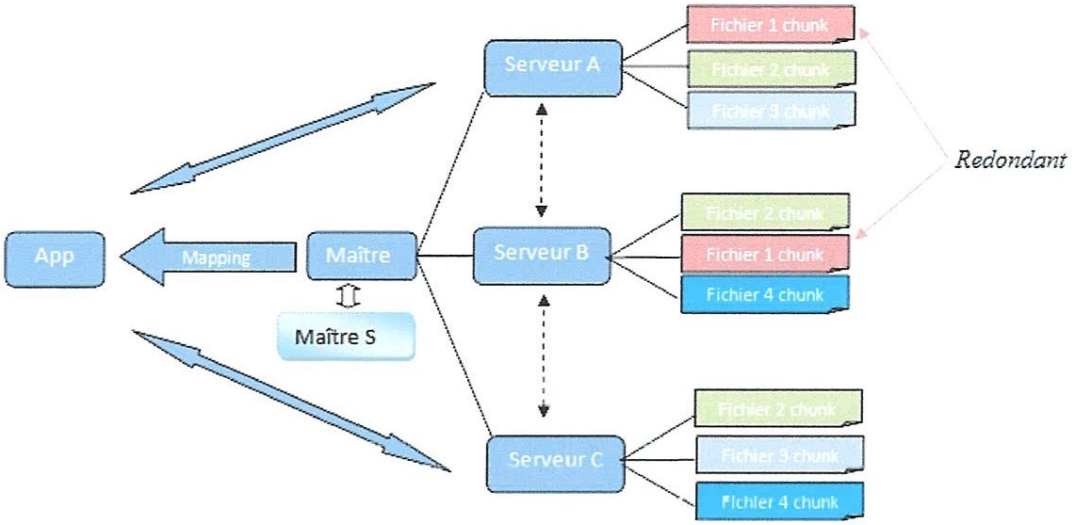


Figure 1.8: *Schéma de Google File System [3]*

Pour manipuler les données volumineuses, il s'agit d'un framework destiné à manipuler ce type de données sur une architecture totalement distribuée.

5.2. Architecture distribuée

La distribution consiste à découper une grande collection en fragments en fonction d'une clé ; et placer chaque fragment sur un serveur. Maintenir un répertoire indiquant que telle clé se trouve dans tel fragment sur tel serveur [32]. Le principe de l'architecture distribuée de big data est de diviser ses données pour mieux les régner et les rendre hautement disponible puis de les distribuer sur des serveurs, de façon transparente de multiplier les nœuds. L'architecture de Big data est divisée en six (6) couches de bas vers le haut [31]:

Les couches de surveillance et de sécurité : elles assurent le contrôle et la protection de données volumineuses.

Couche d'infrastructure Hadoop : elle comprend les serveurs virtuels (VMware).

Couche de gestion de plateforme Hadoop : cette couche assure le traitement et l'analyse de données suivant le paradigme MapReduce.

Couche de visualisation : assure la visualisation de résultat et l'administration de Hadoop.

Couche d'ingestion : elle contient les sources de données hétérogènes (BD, données multimédias, etc.).

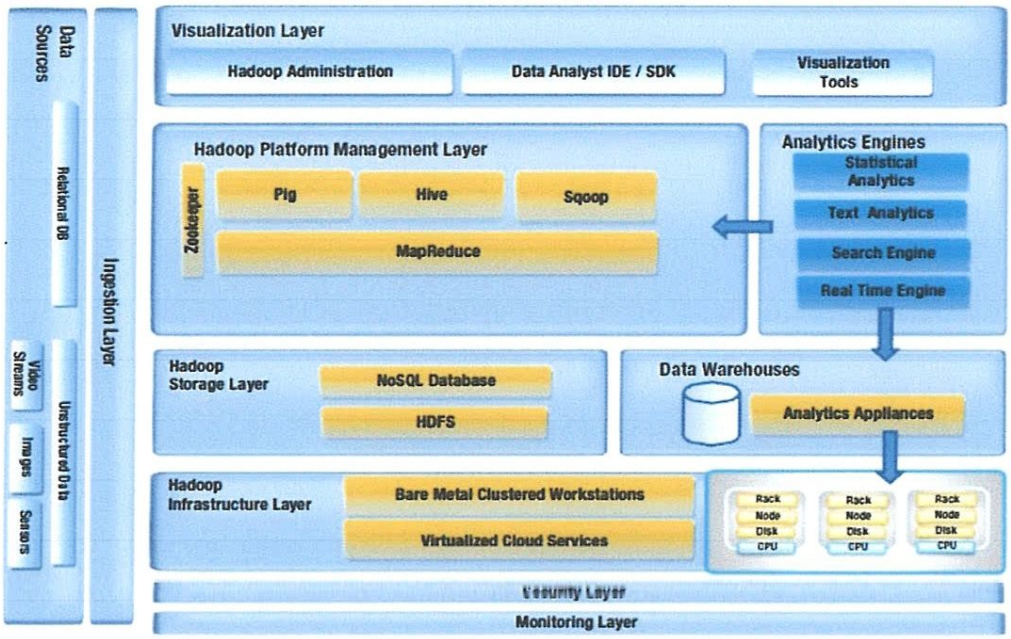
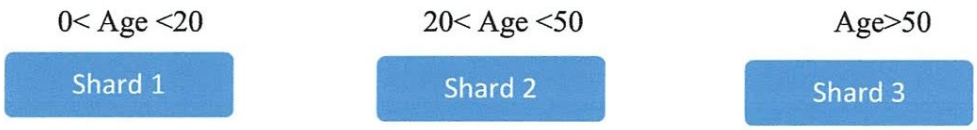


Figure 1.9: *Architecture de Big data [31]*

Le Sharding est une technique qui consiste à diviser des bases de données de taille importante en plusieurs fragments de taille réduite (Shard) afin d'accélérer leur traitement et de les gérer plus facilement. Ainsi les requêtes se feront sur de plus petites données, ce qui permet d'obtenir un temps de réponse plus rapides [31]. Un exemple illustrant le principe de Sharding sur la propriété Age.



L'architecture distribuée de données Nosql se fait selon deux types de distribution : la distribution sans maître et la distribution avec maître.

5.2.1. La distribution sans maître :

Dans la distribution sans maître, tous les nœuds du cluster jouent le rôle d'un maître, ils envoient des requêtes aux d'autres nœuds maîtres et ils attendent les réponses. Dans cette architecture les connexions utilisateurs peuvent s'effectuer sur n'importe quel nœud, qui se chargera de rediriger le client vers le nœud contenant les données souhaitées [16].

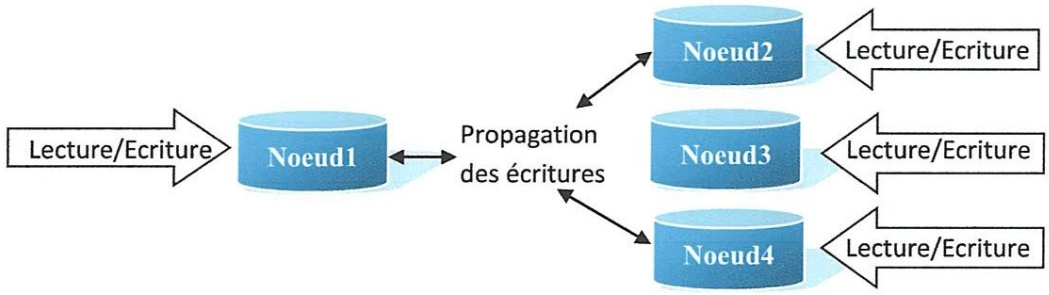


Figure 1.10: *Mode de distribution sans maître [16]*

5.2.2. Distribution avec maître :

La distribution de données sur un schéma avec maître consiste à avoir un serveur dit maître, les autres serveurs des esclaves. La requête du client arrive automatiquement sur le serveur maître, pour ensuite être redirigées sur les serveurs esclaves. Le serveur maître planifie l'exécution de la requête puis il sélectionne le serveur esclave qui contient la réponse. De plus, le serveur esclave est présent pour prendre la place du maître on cas d'une panne. Cette structure permet de garantir une forte cohérence des données, car seul le maître s'occupe des clients [31].

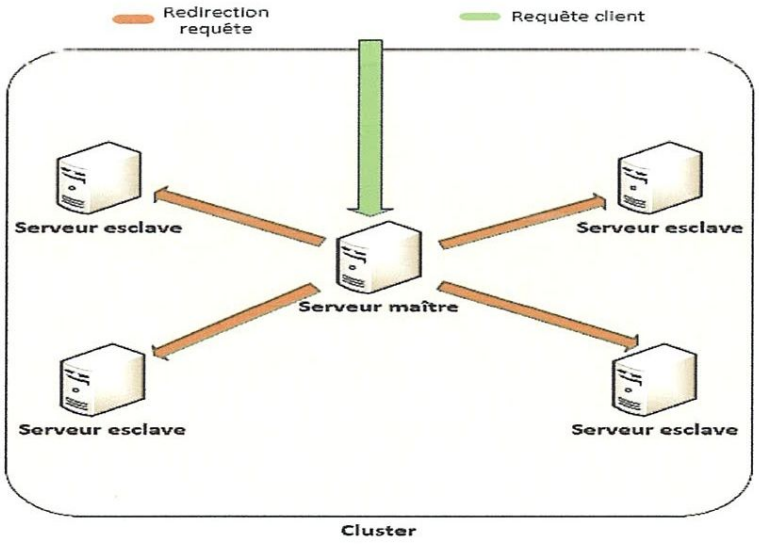


Figure 1 .11: *Mode de distribution avec maître [31]*

5.3. Le paradigme MapReduce

Le MapReduce est un paradigme de programmation parallèle inventé par Google et inspiré de Google MapReduce [11]. Il est utilisé pour le traitement de données massives en les distribuant aux seins d'un cluster de nœuds et produisant des requêtes distribuées. Ce

paradigme connaît un succès auprès de sociétés possédant une quantité de données très importante telles Amazon et Facebook.

Cette technique se décompose en deux grandes étapes [11]:

Etape de Map : prend en entrée un ensemble de données à traiter puis une fonction Map qui découpe ces données en plusieurs sous-ensembles. Ces derniers sont ensuite traités par des différents nœuds, et produit une liste intermédiaire de couples < clé, valeur >, Map (clé1, valeur 1) → List (clé2, valeur2).

A la fin de l'étape Map, les résultats intermédiaires sont regroupés et triés par clé. C'est l'étape de SHUFFLE and SORT.

Etape de Reduce : A partir des nœuds les plus bas on fait remonter leurs résultats au nœud parent qui les avait sollicités. Ce qui permet de calculer et fusionner les résultats intermédiaire ayant la même clé. Reduce (key2, List (valeur2)) → valeur2.

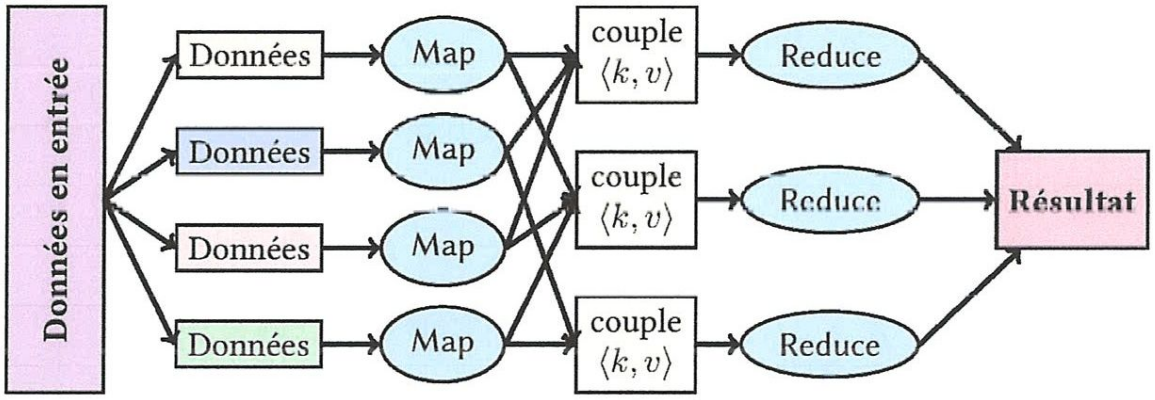


Figure 1.12 : *Schéma de fonctionnement du MapReduce [11]*

L'architecture de MapReduce est de type Maître-esclave où un nœud maître dirige tous les nœuds esclaves. Le maître choisit des nœuds esclaves libres et leur attribue des tâches Map ou Reduce à exécuter. Si un nœud est suspecté d'être défaillant, le maître relance ses tâches sur un nouveau nœud. En revanche, les défaillances du maître ne sont pas prises en compte [11].

5.3.1. Caractéristiques de MapReduce

MapReduce possède quelques caractéristiques [35]:

- Le modèle de programmation du MapReduce est simple mais très expressif. Bien qu'il ne possède que deux fonctions, Map() et Reduce() , elles peuvent être utilisées pour de nombreux types de traitement des données comme les fouilles de données et les graphes…
- Le système découpe automatiquement les données en bloc de même taille. Puis, il planifie l'exécution des tâches sur les nœuds disponibles. Ce qui assure des traitements parallèles.
- Il fournit une tolérance aux fautes grâce à laquelle il peut redémarrer les nœuds ayant rencontré une erreur on affecter la tâche à un autre nœud.
- La parallélisassions est invisible à l'utilisateur afin de lui permettre de se concentrer sur le traitement des données.

5.3.2. Inconvénient de MapReduce

MapReduce souffre de quelques points faibles [34] :

- Utilise les langages de bas niveau.
- Elle ne gère pas les index, ce qui provoque des baisses de performance.
- Certains algorithmes complexes sont difficiles à implémenter avec seulement deux méthodes map() et reduce().
- MapReduce est prévu pour lire un seul élément en entrée et génère une seule donnée en sortie, les algorithmes a multiples éléments en entrée ne sont pas bien supportés.
- Avec la tolérance aux pannes, les opérations ne sont pas toujours optimisées pour les entrées/sorties.

Les outils émergents du Big Data d'aujourd'hui comme Apache Storm ou Apache Spark tournent sur le framework Hadoop Yarn. Nous présentons brièvement cette plateforme (plus de détail voir le chapitre 2).

Hadoop est un Framework open source développé en Java, il a vu le jour on s'inspire de deux produits, le premier est le «Google File System» GFS. Le deuxième est bien évidemment le modèle de Programmation MapReduce. Hadoop est destiné à faciliter le développement d'applications distribuées et scalables, permettant la gestion de milliers de nœuds ainsi que des pétaoctets de données.

6. BigTable de Google

Nous présentons dans cette section un exemple de base de données NoSQL orientées colonne de Google appelée BigTable qui est basée sur une architecture maitre/esclave.
Le BigTable est une table de données multidimensionnelle, distribuée et triée [22]. Cette table est indexée par un identifiant de ligne, dit RowKey, un identifiant de colonne, dit ColumnKey et un horodatage, dit Timestamp, dont chaque valeur est un tableau d'octets non interprétés [22]. BigTable est utilisé par plusieurs applications de Google et répond à la nécessité d'un système de stockage de données structurées hautement évolutif, puisqu'il fournit un accès aléatoire aux données et en temps-réel [16]. BigTable n'est pas une base de données relationnelle, vu qu'il structure les données dans des enregistrements agrégés dans de gigantesques fichiers indexés [31].

Les enregistrements de données de Bigtable sont composés de colonnes groupées dans des familles. Les enregistrements de données sont identifiés par des RowKeys triés d'une manière lexicographique. Les valeurs dans chaque colonne disposent d'un Timestamp pour sauvegarder les valeurs antérieures [21].

Figure 1.13 : *Architecture de googleFS*

7. Conclusion

L'objectif de ce chapitre était de présenter les données NoSQL et ces différentes familles qui permettent d'offrir une gestion optimale et efficace de données volumineuses. Ces données NoSQL ont été conçues sous l'impulsion d'entreprises qui voulaient répondre à leurs propres besoins, le plus connu est Google. Nous avons présenté également le paradigme de programmation parallèle appelé Map reduce pour la manipulation de données massives, avec ses caractéristiques intéressantes. Le framework Hadoop est un très bon exemple de composant logiciel à base de Map reduce pour l'utilisation et la manipulation de données NoSQL. Le détail de ce framework avec ses fonctions sera l'objet du chapitre suivant.

Chapitre 02

Le Framework Hadoop

1. Introduction

Nous avons vu dans le chapitre précédent les notions fondamentales sur les données NoSQL. Des outils émergents aujourd'hui comme Apache tournent sur le framework Hadoop Yarn. Hadoop est l'un des outils les plus performants pour répondre aux problèmes du Big Data et de données NoSQL. Il permet aux applications de travailler avec des milliers de nœuds (distribution du traitement) et de centaines de téraoctets de données ce qui facilite la manipulation **et le traitement d'immenses volumes de données. Ce** Framework **et ses différents composants ont fait leurs épreuves d'utilisation par des nombreuses entreprises pour répondre au problème de grosses données, on trouve** Facebook pour l'analyse des logs, Google pour l'analyse des requêtes, etc.

Dans ce chapitre, nous présentons ce framework Hadoop ainsi que ses deux composants fondamentaux: le système HDFS et le programme MapReduce. Le chapitre termine par la description des SGBD NoSQL choisi pour atteindre les objectifs de ce projet de fin d'étude.

2. Historique

Hadoop (**H**igh-**a**vailability **d**istributed **o**bject-**o**riented **p**latform) a été développe par Doug Cutting en 2004, à l'origine du projet open source Nutch (un moteur de recherche open source de la fondation Apache). Le nom "Hadoop" était initialement celui d'un éléphant en peluche, jouet préféré du fils de Doug Cutting [37].

Doug Cutting a été un des principaux développeurs de la bibliothèque d'indexation Lucene [37]. Sa recherche est concentrée sur la distribution du traitement de Lucene afin de bâtir le moteur d'indexation web Nutch. De plus, Doug propose une solution pour la distribution de données et des calculs sur différentes machines, mais il a rencontré des problèmes. Il décidait donc de s'inspirer de la publication de Google sur leur système de fichier distribué GFS (Google File System). Il développe le composant HDFS pour Hadoop qui gère le stockage efficace de données volumineuses et le MapReduce pour le traitement distribué de ces données.

En 2008, Yahoo propose Hadoop sous la forme d'un projet Open Source. Aujourd'hui, le framework et son écosystème de technologies sont gérés et maintenus par l'association

Apache Software Foundation, une communauté mondiale de développeurs de logiciels et de contributeurs [38].

Après quatre ans de développement au sein de la communauté Open Source, deux versions d'Hadoop ont été proposée : Hadoop 1.0 fut proposé au public à partir de novembre 2012 et Hadoop 2.0 [39].

La principale différence entre la première et la deuxième version d'Hadoop est la séparation entre la gestion des ressources du cluster et le modèle de traitement des données [39]. La figure suivante présente les deux versions d'Hadoop avec ses composants.

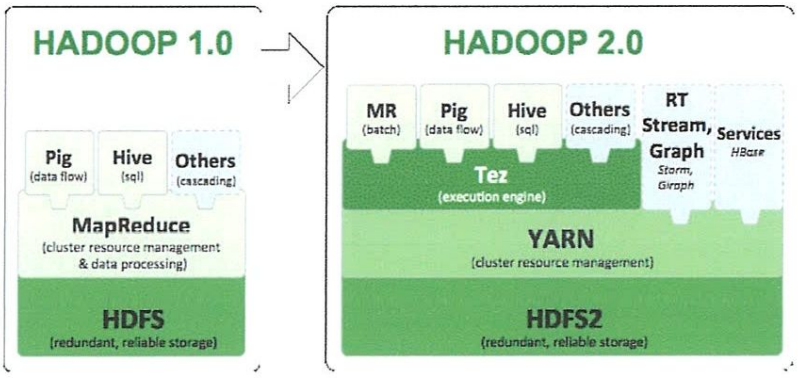


Figure 2.1: *L'évolution d'Hadoop [39]*

Hormis le système HDFS et le programme MapReduce, d'autres parties essentiales dans la constitution d'Hadoop qui sont [39]:

1. **Hive :** il présente un langage HQL (Hive Query Language) proche de SQL développé par Facebook pour interroger une base Hadoop.
2. **Hbase :** est une base de données NoSQL orientée "colonne" et utilisant HDFS.
3. **Pig :** il définit un langage pig latin développé par Yahoo similaire dans sa syntaxe à Perl et visant les objectifs de Hive.
4. **Sqoop :** Sq désigne SQL et oop désigne Hadoop, il assure le mapping entre bases SQL et Hadoop.
5. **D'autres parties** telles ZooKeeper pour assurer la coordination et Oozie pour l'ordonnancement.

3. Présentation d'Hadoop

Nous présentons dans cette section le framework Hadoop et ses composants.

3.1. Définition

Hadoop est un framework open source développé en Java, destiné pour le stockage et le traitement des données, dans un environnement distribué via des clusters qui se composent de plusieurs nœuds équipés de disques durs banalisés (système embarqué) [37]. C'est l'addition de la capacité de stockage et de traitement de chacun des nœuds qui permet d'offrir un espace de stockage et une puissance de calcul. De plus, augmenter le nombre de serveurs de calcul dans le réseau améliore significativement le traitement des données et réduire le temps de réponse. Hadoop permet de faire des traitements sur les données sans avoir besoin d'une connaissance de fond de son architecture. Il est conçus dans l'idée fondamentale d'une haute disponibilité et que les pannes matérielles ne sont pas tolère [37].

Le noyau d'Hadoop est constitué de deux parties importantes [37] [38] :

La partie de stockage: qui assure le stockage et l'intégrité des données à l'aide du HDFS (Hadoop Distributed File System).

La partie de traitement : qui assure le traitement des gros volumes de données via le paradigme MapReduce. Hadoop découpe les fichiers en blocs de taille fixe et les distribue à travers les nœuds du cluster, chaque nœud traite les données dont il dispose, ce qui garantit un traitement plus rapide et plus efficace.

Les SGBD traditionnels sont conçus pour fonctionner en mode transactionnel, ils ne sont en aucun cas capables de traiter de manière séquentielle des volumes de données. Le framework Hadoop permet d'assurer le traitement parallèle de grosses données notamment les données NoSQL. Il est caractérisé par les propriétés suivantes [37] [38] :

- **Robuste** : si un nœud de calcul tombe en panne, le travail est repris automatiquement par un autre nœud.
- **Coût** : il optimise les coûts grâce à sa distribution de stockage et traitement sur plusieurs nœuds.
- **Économique** . Hadoop permet aux entreprises de contrôle et de gestion aisés de données volumineuses par des serveurs moins couteux.
- **Flexible** : il est capable de traiter différents types de données.
- **Virtualisation** : il ne plus se repose sur l'infrastructure physique (stockage coûteuse), mais de virtualiser ses clusters.

Le Hadoop a été développé pour accéder à des gros volumes de données en des temps raisonnables à l'aide de traitement distribué du paradigme MapReduce. D'autre produit de la fondation Apache appelé Mahout qui est une collection des algorithmes programmées en java qui s'exécute sur Hadoop mais Hadoop n'est pas indispensable [38]. Les principaux algorithmes de Mahout sont les algorithmes de système de recommandation (SVD, kmeans, régression logistique, classifieur bayésien, etc.), l'apprentissage distribué, etc.

La section suivante présente les composants de la plateforme Hadoop.

3.2. Les composants Hadoop

Comme on a dit dans la section 1 que le framework Hadoop se compose de plusieurs parties, les plus importantes qui assurent le stockage et le traitement de données NoSQL sont : le système HDFS et le programme MapReduce Hadoop. Dans cette section nous allons détailler ces deux composants d'Hadoop.

3.2.1. Hadoop Distributed File System (HDFS)

Le HDFS (Hadoop Distributed File System) est un système de gestion de fichiers distribué, extensible et portable, écrit en java, développé par Hadoop à partir du GoogleFS [28]. Il a été conçu pour stocker des données structurées ou non de taille volumineuse sur un grand nombre de machines (sharding automatique en plusieurs blocs de données). Les données sont répliquées sur plusieurs nœuds différents afin d'une part d'assurer la disponibilité et la performance et d'autre part d'éviter les pertes d'information en cas d'une panne.

Deux grandes stratégies existent pour éviter la perte de données stockées et donc être tolérant aux pannes. L'une consiste à utiliser du matériel haut de gamme réalisant un stockage redondant au plus bas niveau (disques multiples, RAID, etc.) ou bien à utiliser plusieurs nœuds de stockage et à répliquer les données sur ces différentes machines. Le HDFS est plutôt orienté sur cette deuxième solution. Par défaut, HDFS fait des réplications sur 3 machines différentes, si possible sur différents sites.

De plus, le HDFS utilise l'abstraction afin de manipuler un système distribué comme s'il s'agissait d'une unique machine. Il est une architecture maître-esclave : le maître (le Namenode) et les esclaves (les Datanodes). La figure suivante illustre l'architecture HDFS.

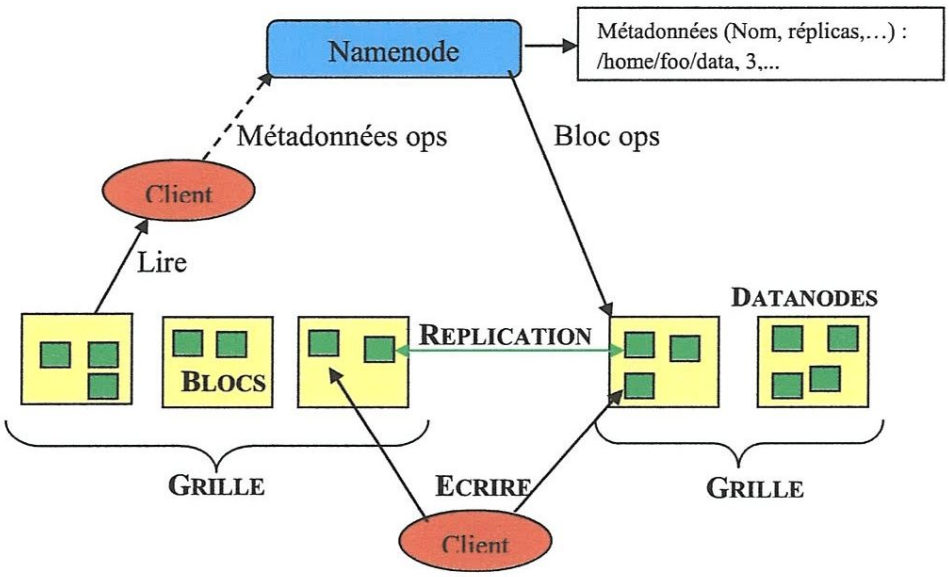


Figure 2.2 : *Architecture d'HDFS [40]*

L'architecture HDFS repose sur deux composants majeurs qui sont [40]:

1. **NameNode (nœud de noms) :** Ce composant est responsable à la gestion de l'espace de noms et la reconstitution des fichiers à partir des différents blocs répartis. Il assure la distribution et la réplication des blocs ainsi que le stockage et la gestion des métadonnées et les données log de toutes les transactions.
2. **DataNode** (nœud de données) : Ce composant est responsable au stockage et à la restitution des blocs de données. Il communique avec le NameNode d'une manière périodique selon la liste des blocs de données qu'il héberge.

L'objectif de la politique de placer les réplicas des blocs en grille (rack en anglais) est d'améliorer la fiabilité et la disponibilité de données ce qui empêche la perte de données lorsqu'un rack tombe en panne. Cependant, cette politique augmente le coût des écritures pour transférer les blocs à plusieurs racks, et réduit la bande passante globale du réseau lors de lecture des données placé dans deux racks différents [43].

Le NameNode prend toutes les décisions concernant la réplication des blocs. Il reçoit périodiquement de chaque DataNodes, un rapport du bloc (Block report) qui contient une liste de tous les blocs des DataNodes du cluster. La réception de ce rapport implique le bon fonctionnement de DataNode [43].

La persistance des métadonnées du système de fichiers : Le NameNode utilise un journal des transactions appelé EditLog qui contient tous les changements produits dans le HDFS. Par

exemple, la création d'un nouveau fichier dans HDFS provoque la NameNode d'insérer un enregistrement dans la EditLog. [43]

Le HDFS assure la tolérance aux fautes par la détection des erreurs et les récupérer rapidement et automatiquement selon deux cas distincts [40] :

i. **Dans le cas d'une panne du DataNode** : Si le NameNode ne voit plus de « Heartbeat » (un message décrit la mise à jour de plusieurs serveurs, pour effectuer entre eux un processus de tolérance de panne) provenant du DataNode, il utilisera les réplicas des autres DataNode pour les prochaines lectures. Un nouveau lot de réplicas sera alors mis en place sur les DataNode restant.

ii. **Dans le cas d'une panne du NameNode** : Sauvegarde des logs de transaction sur un support stable, et un redémarrage de la dernière image du HDFS.

Une fonction très importante de HDFS est la lecture et l'écriture de flux des fichiers [40] [41].

- **Lecture d'un fichier HDFS** : Le client indique au NameNode qu'il souhaite lire le fichier. Le NameNode lui renvoi le nombre de blocs qui constitue le fichier, et les différents Data Node qui hébergeant ces blocs. Enfin le client récupère chacun des blocs. En cas de panne de Datanode, le NameNode renvoie l'adresse du DataNode le plus accessible qui dispose de la plus grande bande passante).

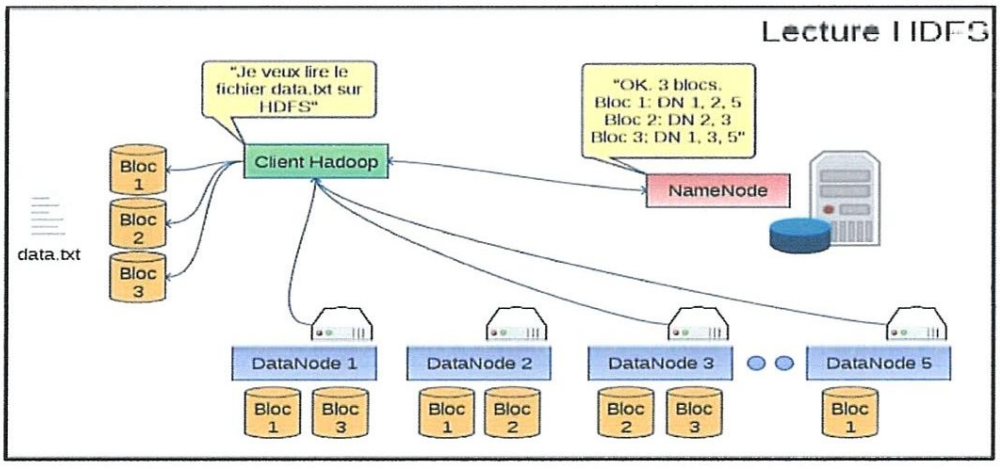


Figure 2.3 : *Lecture d'un fichier HDFS [41]*

- **Ecriture dans un fichier HDFS :** le fichier à écrire doit être divisé en blocs de 64Mo par défaut et qui possible de monter à 128 Mo, 256 Mo, 512 Mo voire 1 Go (selon la configuration effectuée). Par la suit le NameNode lui indique les DataNodes à contacter.

Le client contacte directement le DataNode concerné et lui demande de stocker le bloc. Les DataNodes s'occuperont en informant le NameNode de répliquer les données entre eux pour éviter toute perte de données [41].

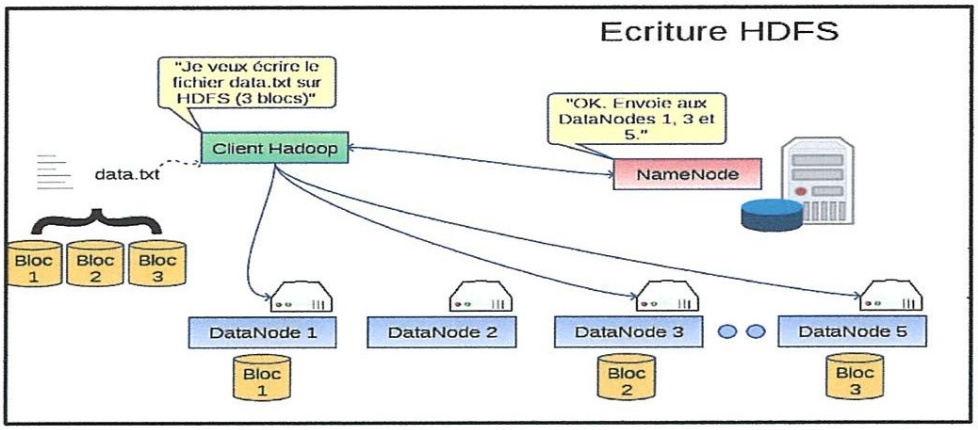


***Figure 2.4** : Ecriture d'un fichier HDFS [41]*

3.2.2. MapReduce dans Hadoop

Le second composant d'Hadoop est le module MapReduce, qui gère la répartition et l'exécution des requêtes sur les données stockées par le cluster. Une architecture de type maître-esclave dont le maitre réalise la tâche Job tracker (le gestionnaire de tâches), et les esclaves effectuent les Task trackers (le moniteur de tâches). Le module MapReduce repose sur cinq composants fondamentaux [34]:

1. **Le gestionnaire de tâches (Job Tracker) :** il est unique sur le *cluster*. Il vise à assurer la coordination des tâches sur les différents clusters. Il reçoit les tâches MapReduce à exécuter (sous la forme d'une archive .jar), et organise leur exécution sur le cluster (attribuer les fonctions de MapReduce aux différents TaskTrackers).
2. **Le moniteur de tâches (Task Tracker) :** on trouve plusieurs tasktrackers par cluster. Le moniteur de tâches permet d'exécuter des tâches données par le Jobtracker, ainsi que la lecture des blocs de données en accédant aux différents DataNodes. Par ailleurs, le Task Tracker notifie de façon périodique au JobTracker la progression des tâches qu'il exécute.
3. **Name node :** c'est le nœud principal dans l'architecture d'Hadoop (plus de détail voir la section 2.2.1).
4. **Data node :** c'est le nœud de données, il assure le stockage des blocs de données (plus de détail voir la section 2.2.1).

5. **SecondaryNode** (noeud secondaire) : si le namenode s'arrête, il n'y a pas moyen de pouvoir extraire les blocs d'un fichier donné. Pour résoudre ce problème, un nœud de noms secondaire a été mis en place dans l'architecture Hadoop. Son rôle consiste à faire une copie des données du « NameNode » afin de pouvoir prendre la relève en cas de panne de ce dernier.

La figure suivante illustre l'architecture de MapReduce d'Hadoop.

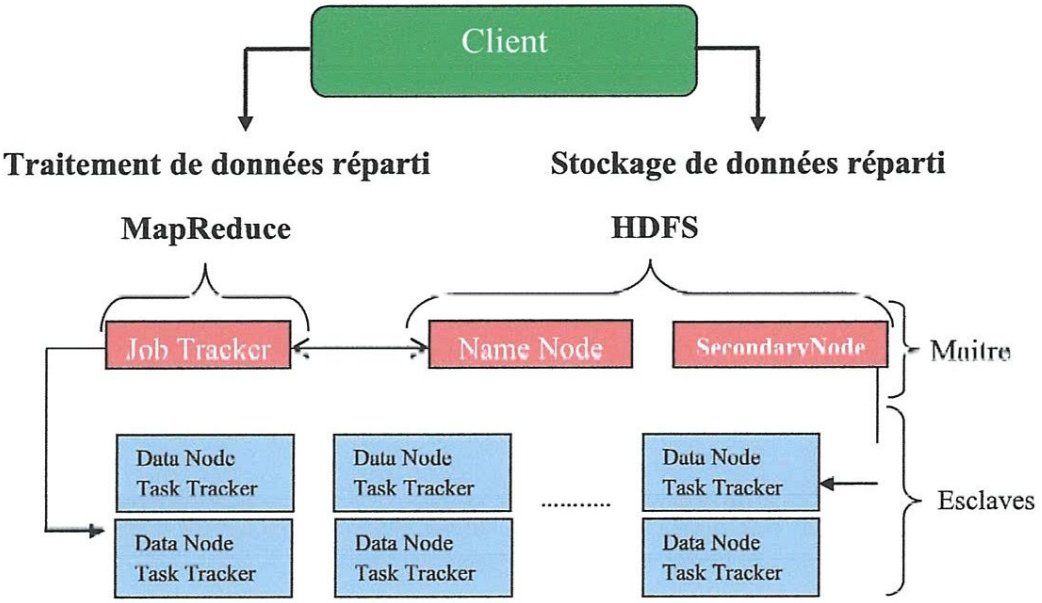


Figure 2.5 : *Architecture MapReduce d'Hadoop [45]*

Chacun des TaskTrackers constitue une unité de calcul du cluster. Le composant JobTracker est en communication avec HDFS. Il sait où sont les données d'entrée du programme MapReduce et où doivent être stockées les données de sortie. Il peut ainsi optimiser la distribution des tâches selon les données associées. La section suivante présente l'exécution du programme MapReduce.

3.3. Exécution du programme MapReduce

D'une manière sommaire, le programme MapReduce, prend en entrée les données à écrire sur HDFS (la fonction MAP) sous forme de fichier et envoie la requête cliente au JobTracker du cluster NameNode, puis il récupère les données de sortie depuis HDFS (la fonction Reduce). Le traitement MapReduce écrit par l'utilisateur s'appelle un job MapReduce sous Hadoop et il s'exécute en sept étapes suivantes [34] [35] :

Etape 1 : La configuration de Job MapReduce par l'utilisateur : consiste à écrire la fonction Map, et la fonction Reduce et de spécifie le nombre de tâches Reduce (r), le format de lecture du fichier d'entrée, le format de sortie des r fichiers Reduce, et la taille des blocs du fichier d'entrée et le facteur de réplication. Une fois que ces différentes tâches sont faites et qu'il déclenche l'exécution du job, le jobtracker démarre les r tasktrackers qui vont effectuer les r tâches Reduce que l'utilisateur a spécifié.

Etape 2 : le découpage et réplication : le HDFS découpe le fichier d'entrée en N blocs de taille fixe (Par défaut 64 Mo, ou selon la configuration). Ensuite, le HDFS réplique ces blocs selon le facteur de réplication définie par l'utilisateur (trois par défaut) et les distribue de façon redondante dans des nœuds différents dans le cluster. Le fait de diviser le fichier d'entrée en blocs de taille fixe permet d'équilibrée la charge de traitement parallèle.

Etape 3 : le déclenchement des tasktrackers (exécution de Map) : Le jobtracker déclenche N tasktrackers (sur les N nœuds dans lesquels ont été répartis), pour exécuter les tâches Map, soit un tasktracker-Map pour chaque bloc de fichier. Chaque tasktracker lit le contenu du bloc de fichier, le transforme par le processus de hachage défini dans la fonction Map en paires de clés/valeurs (Ce processus de hachage s'effectue en mémoire locale du nœud)

Etape 4 : la localisation (Préparation pour l'exécution de Reduce) : Dans chaque nœud, les paires de clés/valeurs sont sauvegardées sur la mémoire local du nœud. Ensuite ce fichier est partitionné en r régions (correspondant aux r tâches Reduce) par une fonction de hachage qui va assigner à chaque région une clé qui correspond à la tâche Reduce à laquelle elle a été assignée. Les informations sur la localisation de ces régions sont transmises au jobtracker, qui fait suivre ces informations aux r tasktrackers qui vont effectuer les tâches Reduce.

Etape 5 : Le Trie les données (Préparation pour l'exécution de Reduce) : Lorsque les r tasktrackers-Reduce sont notifiés des informations de localisation, ils utilisent des appels de procédures distantes (protocole RPC) pour lire depuis la mémoire des nœuds sur lesquels les tâches Map se sont exécutées, les régions des fichiers Map leur correspondant. Ensuite, ils les trient par clé (Notez au passage que le tri s'effectue en mode batch dans la mémoire du tasktracker-Reduce), si les données sont trop volumineuses, alors cette étape peut augmenter le temps total d'exécution du job.

Etape 6 : L'exécution de Reduce : Les tasktrackers-Reduce itèrent à travers toutes les données triées et pour chaque clé unique rencontrée, ils la passent avec sa valeur à la fonction Reduce écrite par l'utilisateur. Les résultats du traitement de la fonction Reduce sont alors enregistrés dans le fichier r_i (avec i l'indice de la tâche Reduce). Cette fois-ci, les fichiers ne sont pas sauvegardés dans la mémoire locale du nœud tasktracker, mais dans le HDFS (tolérance aux pannes).

Etape 7 : Le job s'achève : À ce stade, les r fichiers Reduce sont disponibles et Hadoop applique en fonction de la demande de l'utilisateur, soit un Print, soit leur chargement dans un SGBD, soit comme fichiers d'entrée à un autre job MapReduce.

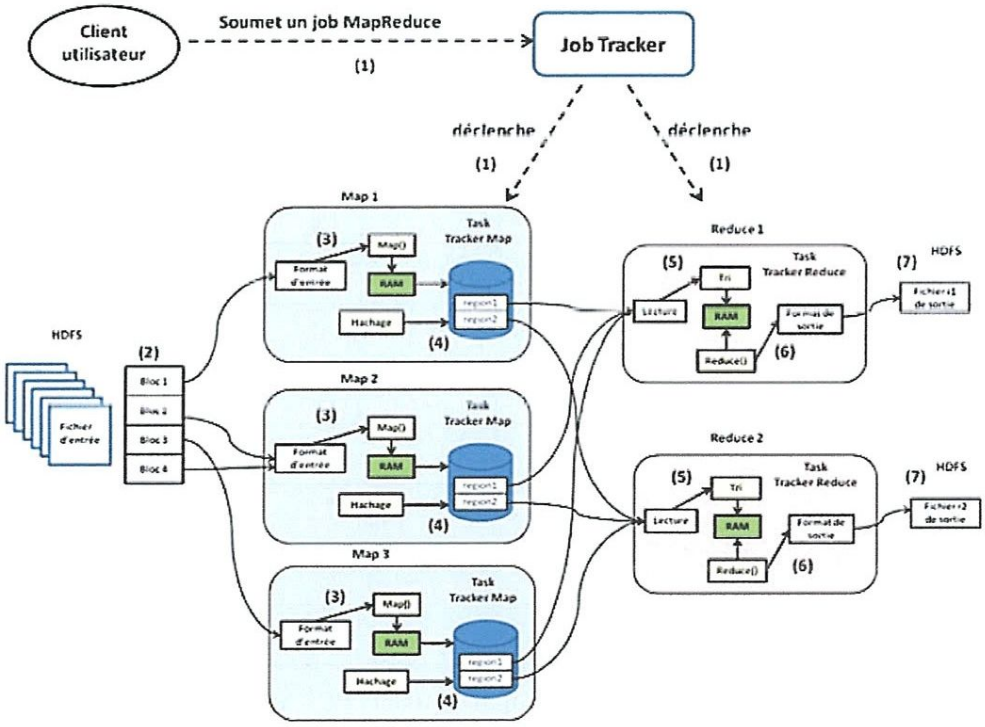


Figure 2.6 : *Étapes d'exécution d'un job MapReduce dans un cluster Hadoop [40]*

Tous les TaskTrackers signalent leur état continuellement par l'envoi d'un paquet Heartbeat. En cas de défaillance d'un TaskTracker (heartbeat manquant ou tâche échouée), le JobTracker considère le nœud défiant (redistribution de la tâche à un autre nœud). Au cours de l'exécution d'une tâche, on peut obtenir des statistiques détaillées sur son évolution (étape actuelle, avancement, temps estimé avant l'exécution,…etc) [41].

4. Architecture d'Hadoop

Dans un écosystème d'Hadoop multi nœuds, on peut avoir sur une même machine l'exécution du JobTracker et du NameNode, aussi l'exécution d'un TaskTracker avec un DataNode [40]. La figure suivante montre l'architecture générale d'Hadoop avec ses différents composants.

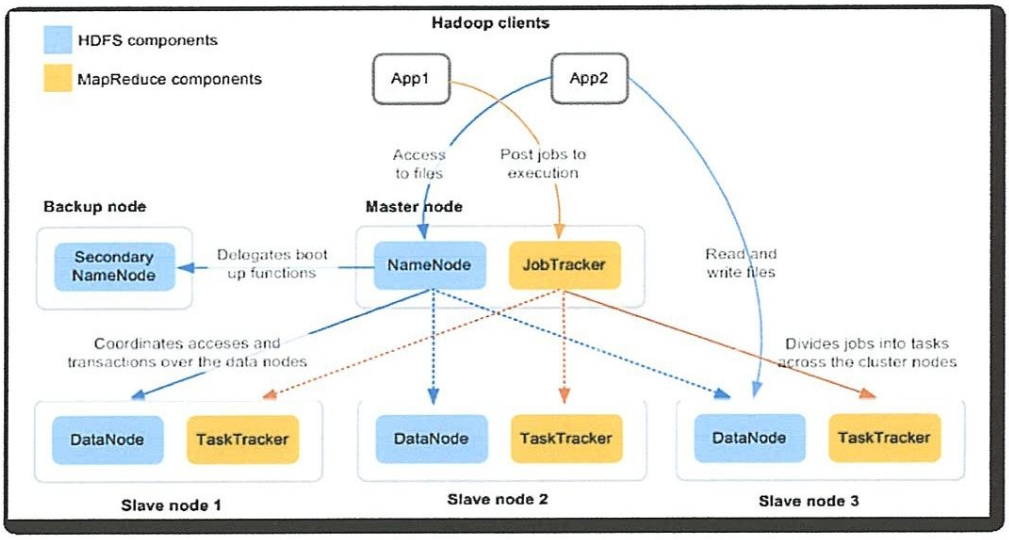


Figure 2.7 : *Architecture générale Hadoop [41]*

Le Framework Hadoop se réfère non seulement aux modules de base HDFS et MapReduce, mais aussi à son écosystème et à l'ensemble des logiciels qui le complète, pour permettre: Le stockage et l'extraction des données, la simplification de traitement distribué, l'entrepôt de données, le workflow, la programmation, sans oublier la gestion et coordination de la plateforme. Dans la figure suivante, nous présentons un schéma graphique sur l'écosystème Hadoop avec l'utilisation des logos propres pour chaque composant.

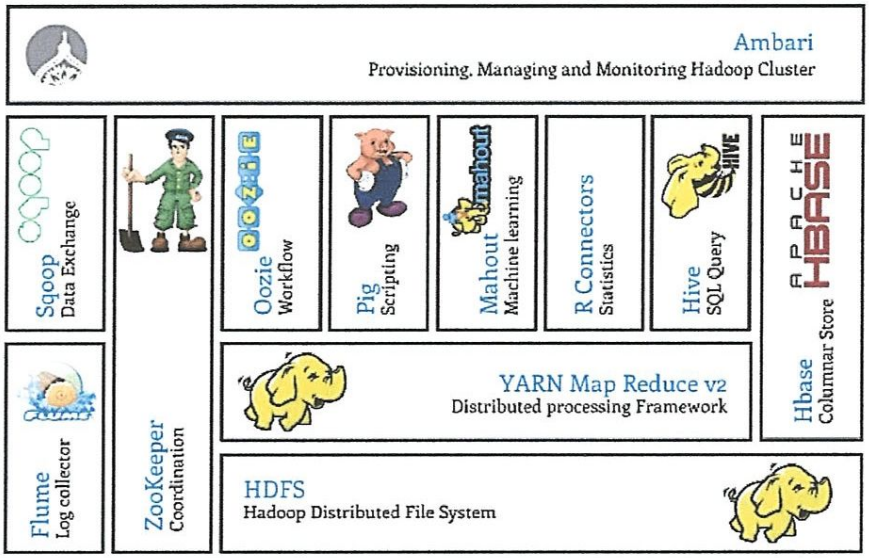


Figure 2.8 : *Ecosystème Hadoop [44]*

Hadoop est destiné à faciliter le développement d'applications distribuées et scalables, permettant la gestion de milliers de nœuds ainsi que des pétaoctets de données. C'est un ensemble de classes écrites en Java pour la programmation des tâches MapReduce et HDFS dont les implémentations sont disponibles en plusieurs langages de programmation. Ces classes permettent aux développeurs d'écrire des fonctions Map et des fonctions Reduce qui vont traiter les données sans que le développeur ait à savoir comment ces fonctions sont distribuées et parallélisées dans le cluster Hadoop [36].

Un Job MapReduce est une unité de travail que le client veut exécuter. Il consiste en trois choses :

- le fichier des données à traiter (Input file),
- le programme MapReduce,
- et les informations de configuration (Métadonnées).

Le cluster exécute le job MapReduce en divisant le programme MapReduce en deux tâches (les tâches Map et les tâches Reduce). Dans Hadoop, il y a deux types de processus qui sont responsable de l'exécution du job (un jobtracker et un ensemble de tasktrackers).

Le jobtracker : c'est le processus central qui est démarré sur le nœud de référence (le Name Node), il coordonne tous les jobs qui s'exécutent sur le cluster et gère les ressources du cluster et planifie les tâches à exécuter sur les tasktrackers.

Les tasktrackers : ce sont les processus qui traitent le programme MapReduce de l'analyste, ils sont démarrés au niveau des nœuds de données (Data nodes), exécutent les tâches Map ou Reduce et envoient des rapports d'avancement au jobtracker, qui garde une copie du progrès général de chaque job.

Si une tâche échoue, le jobtracker peut le planifier sur un tasktracker différent. En fait, le jobtracker désigne le nœud de référence et le processus Master qui y est démarré, tandis que le Tasktracker désigne un nœud de données et le processus Worker qui y est démarré [36].

Finalement, pour atteindre notre objectif de ce travail qui vise à faire une étude de données NoSQL sur Hadoop/MapReduce, il est nécessaire de choisir les SGBD de données NoSQL à utiliser. Dans ce contexte la prochaine section est consacré à présenter les caractéristiques des SGBD NoSQL que nous avons choisi pour notre travail.

5. Les SGBD NoSQL choisi

Après avoir présenté le framework Hadoop avec son module MapReduce, le reste de ce chapitre décrit les SGBD NoSQL que nous utiliserons pour effectuer notre étude. De ce fait, nous avons basé sur le classement des SGBD effectuée en mois de Janvier 2018 (voir la figure 2.9), et nous avons choisi deux SGBD NoSQL qui sont: Hbase et MongoDB.

341 systems in ranking, January 2018

Rank			DBMS	Database Model	Score		
Jan 2018	Dec 2017	Jan 2017			Jan 2018	Dec 2017	Jan 2017
1.	1.	1.	Oracle	Relational DBMS	1341.94	+0.40	-74.78
2.	2.	2.	MySQL	Relational DBMS	1299.71	-18.36	-66.58
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1148.07	-24.42	-72.89
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	386.18	+0.75	+55.81
5.	5.	↓ 4.	MongoDB	Document store	330.95	+0.18	-0.96
6.	6.	6.	DB2	Relational DBMS	190.28	+0.70	+7.78
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	126.70	+0.82	-0.75
8.	↑ 9.	↓ 7.	Cassandra	Wide column store	123.88	+0.67	-12.57
9.	↓ 8.	9.	Redis	Key-value store	123.14	-0.10	+4.44
10.	10.	↑ 11.	Elasticsearch	Search engine	122.55	+2.77	+16.38

Figure 2.9 : *Statistiques du meilleurs SGBD (Janvier 2018) [47]*

5.1. HBase

HBase est un sous-projet d'Hadoop écrit en Java, inspiré de Google BigTable. Il s'agit d'un système de gestion de données orienté colonnes destiné à manipuler de très larges volumes de données sur une architecture totalement distribuée. Il est utilisé par des acteurs majeurs du Web, comme Ebay, Yahoo! et Twitter, de même que par des sociétés telles qu'Adobe.

Le HBase mémorise des n-uplets constitués de colonnes (champs) qui sont identifiés par une **clé**. À l'affichage, les colonnes d'un même n-uplet sont affichées successivement. Contrairement aux SGBD orienté lignes (SGBD relationnel), les données sont sérialisées sous forme de colonne plutôt que de ligne [41].

Exemple:

Clés	**(Colonnes, Valeurs)**
001	(auteur, "Victor Hugo")
001	(titre, "les misérables")

5.1.1. Structure interne

Pour obtenir une grande efficacité, HBase sépare les données des tables en régions. Une région contient un certain nombre de n-uplets (un intervalle de clés successives). Initialement une nouvelle table est mise dans une seule région. Lorsqu'elle dépasse une taille supérieure, elle se fait couper en deux régions. Et ainsi de suite si les régions deviennent trop grosses. [41]

Chaque région est gérée par un Serveur de région (RegionServer). Et le même serveur peut gérer plusieurs régions. Ces serveurs sont distribués sur le cluster. Au final, les données sont stockées sur HDFS. La figure suivante montre la séparation d'une table en régions

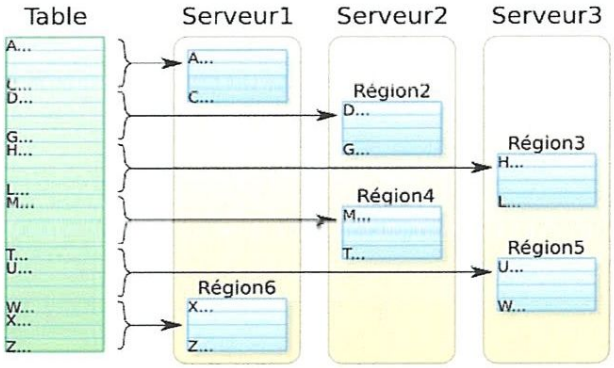


Figure 2.10. *Régions HBase [42]*

5.1.2. Structure des données

Une table HBase est un couple des <clé, n-uplet> triée par **clés**, Chaque **n-uplet** est une liste de familles, Une **famille de colonne** (Column family) est un couple des <nomcolonne, cellule> trié par noms de colonnes. Une **cellule** (Cell) est une liste des paires <valeur, date>. La date (timestamp) permet d'identifier la version de la valeur. Dans chaque table les données sont organisées dans des lignes. Une ligne est identifiée par une clé unique (RowKey).

Donc finalement, pour obtenir une valeur isolée, il faut fournir un quadruplet:
(Clé, NomFamille, NomColonne, Date)

Exemple : On veut enregistrer les commandes d'achats des clients sur trois familles (Personne, Adresse, achats) [42] :

- La famille Personne :
  - colonnes personne: nom, personne: prénom
- La famille Adresse
  - colonnes Adresse: rue, Adresse: ville, Adresse: cp, Adresse: pays
- La famille achats :
  - colonnes achats: date, achats: montant, achats: idfacture

Cet exemple est illustre par la représentation graphique suivante :

Column family

Row key	Personne		Adresse				Achats		
	Nom	Prénom	Rue	Ville	Cp	Pays	Date	Montant	idfacture
1	Aggoune	Aicha	18 Février	Guelma	24000	Algérie	25/02/2018	30.000	3
2	Belhaoues	Hocine	19 Juin	Guelma	24000	Algérie	13/03/2018	1.500	4

Table 1.1 : *Exemple d'une table de la base de données HBase*

Le HBASE comprend un composant appelé BlockCache qui est active par défaut et charge toute opération de lecture. Par ailleurs, le composant WAL de HBase (Write Ahead Log) qui gère les opérations d'ajout (Appending) et des mises à jour (updating) dans un RegionServer.

Toutes modifications dans les régions de tables seront écrites par WAL dans le fichier Hlog de HDFS avant d'être répliquer dans le MemStore (mémoire de stockage de données, il comporte plusieurs fichiers de stockage Storefile).

Hbase possède son propre ensemble d'API utilisé pour manipuler ses données qui peuvent aller jusqu'à milliards d'enregistrements distribués sur plusieurs serveurs tout en ayant un temps de réponse performant. Il dispose des méthodes la modification de contenu des tables HBase telles que la méthode put( ) pour l'insertion d'un élément, scan( ) pour l'affichage de contenu de la base, etc.

5.2. MongoDB

MongoDB est un SGBD NoSQL orienté document qui reste une solution fortement utilisée par les développeurs. Il occupe la 5ème place dans le classement des SGBD les plus performants et la 1ère place pour les SGBD NoSQL [47]. Il a été développé en 2007 par une entreprise travaillant sur des outils de Cloud computing appelée 10gen écrit en langage C++ [40]. Il est très adapté aux applications web à cause de sa simplicité d'utilisation ainsi que ces performances remarquables.

Le MongoDB manipule les documents au format BSON un dérivé de JSON (JavaScript Object Notation). Il réplique les données sur plusieurs serveurs avec le principe de maître-esclave, permettant ainsi une plus grande tolérance aux pannes. La répartition et la duplication de document est faite de sorte que les documents les plus demandés soient sur le même serveur et que celui-ci soit dupliqué un nombre de fois suffisant [46].

5.2.1. Structure de données

Les données sont structurées et stockées sous la forme d'un **document.** Les documents sont enregistrés dans des **collections** (un nombre important de documents). Les collections sont comparables aux tables, et aux enregistrements des bases de données relationnelles. Contrairement aux bases de données relationnelles, les champs d'un enregistrement sont libres et peuvent être différents d'un enregistrement à un autre au sein d'une même collection. Le seul champ commun et obligatoire est le champ de **clé** principale ("id").

Les clés peuvent être ajoutées à tout moment « à la volée », sans reconfiguration de la base MongoDB. Chaque document dans une collection pourrait avoir des données différentes [46].

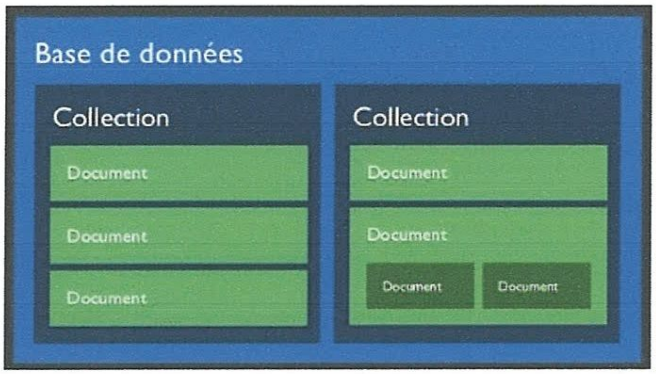


Figure 2.11 : *Une structure des données MongoDB*

5.2.2. Fonctionnement de MongoDB

MongoDB permet de distribuer les morceaux de données appelé Chunks sur plusieurs serveurs via la technique de Sharding. Il s'agit de créer un cluster MongoDB (sharded cluster) composé de plusieurs machines sur lesquelles les données seront répliquées par le mécanisme de ReplicaSet qui consiste d'effectuer un sauvegarde de données et les restituer en cas de défaillance d'un nœud [48]. Le cluster MongoDB se compose de trois principaux éléments [49]:

- Le serveur de configuration sert à stocker les paramètres de configuration du cluster et localiser les données sur différents nœuds.
- Les shards qui stockent un sous ensemble de données pour assurer la scalabilité.
- Un ou plusieurs routeurs communiquent avec le serveur de configuration pour connaitre la répartition des données et donc choisir le bon shard.

La figure suivante illustre un exemple d'une architecture composée de 5 instances : un serveur de configuration, un routeur et 3 shards pour la répartition des données [48].

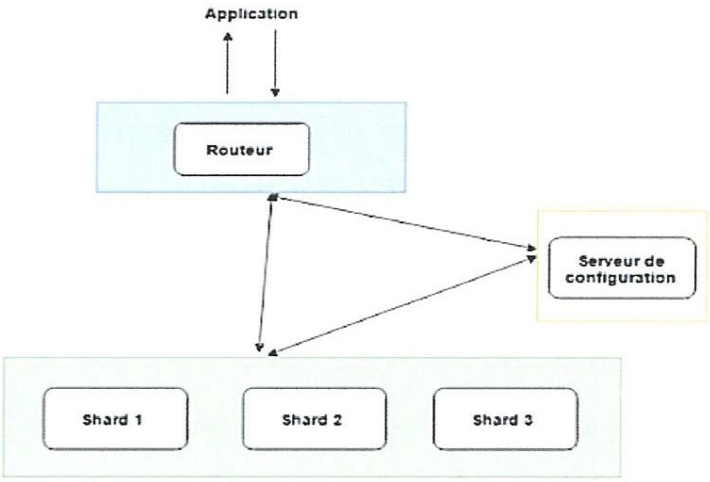


Figure 2.12 : *Cluster MongoDB [48]*

Un jeu de réplicas obtenu par le mécanisme ReplicaSet est un groupe de processus mongodb qui conservent le même ensemble de données en fournissant une redondance et une haute disponibilité de données (documents). Il consiste à ajouter N serveur dont l'un est considéré comme le nœud primaire (serveur maître) et en cas de défaillance de ce nœud, le ReplicaSet choisi un autre nœud qui devient le nœud primaire [49].

La recherche de données de mongoDB est basée sur le mécanisme d'indexation pour accéder directement à l'information recherchée. Le SGBD Mongo repose sur l'indexation Arbre-B non dense (BTree non-dense), dont le serveur Maître est la racine de l'arbre et les feuilles qui contiennent les données sont prises en charge par les nœuds du cluster [50].

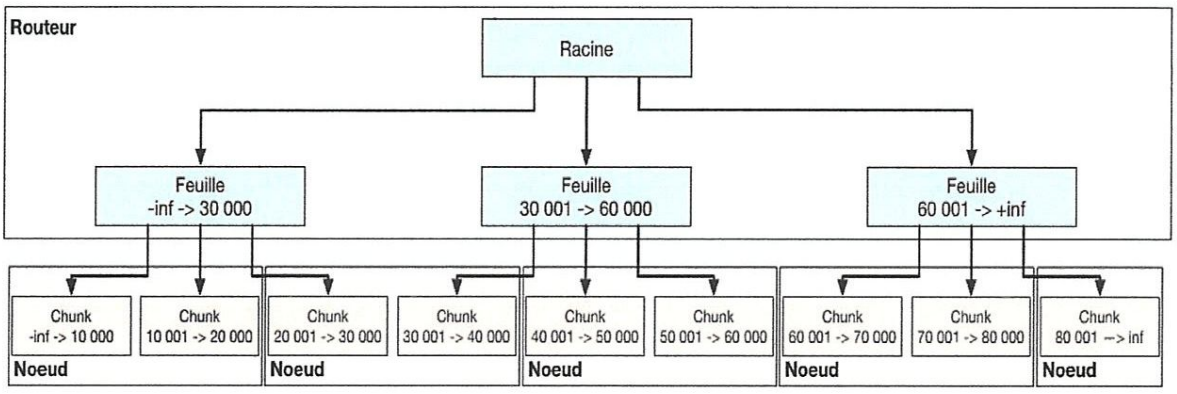


Figure 2.13 : *Exemple de l'indexation de données de MongoDB [50]*

Pour que la répartition de charges soit efficace, il faut que les données soient dispersées uniformément entre les différents shards. C'est le rôle du routeur de répartir les données. Ce

partitionnement se fait au niveau des collections et s'appuie sur une clé de sharding, par exemple, prenons un document Personne qui contient le nom, le prénom et l'âge [50]. L'attribut âge est considéré comme clé de sharding et le MongoDB va définir automatiquement des intervalles d'âges comme il montre cette illustration.

Quand un nouveau document sera ajouté dans la base, il sera directement dirigé vers le shard qui lui correspond.

La manipulation de données orientées documents se fait à travers des requêtes propre pour chaque SGBD. Nous citons dans ce qui suit les requêtes MongoDB les plus utilisées :

- db.COLLECTION_NAME.insert(doc) : Insertion de document doc dans une collection de la base de données db.
- db.COLLECTION_NAME.find().pretty() : recherche et affiche tous les documents dans une collection de la base de données db. Cette requêtes est équivalente à la requete SQL select *.
- db. COLLECTION_NAME.find({key1:value1, key2:value2}).pretty(): est équivalente à la requête select from where, où les conditions sont représentées par des pairs (key, value) comme paramètres de la fonction find. La conjonction entre les conditions (l'opérateur AND) est interpretée par la virgule tandis que la disjonction se fait par $or.

Par exemple : dans la base de données des livres, afficher tous les livres de Rudi Bruchez ou leur titre est 'MongoDB' :

```
>db.livre.find({$or:[{"Auteur":" Rudi Bruchez "},{"titre": "MongoDB"}]}).pretty()
```

Plus de détail sur la manipulation de données de MongoDB via les requêtes NoSQL sera présentée dans le chapitre suivant.

6. Conclusion

L'objectif de ce chapitre était de présenter le framework Hadoop ainsi que ses deux composants fondamentaux: le système HDFS et le programme MapReduce. Le chapitre

termine par la description de deux SGBD NoSQL choisi pour effectuer une étude comparative de données NoSQL : Hbase et MongoDB.

Ces deux SGBD NoSQL ont vécu de très importants changements, aussi bien en termes de performance que d'architecture. Ils ne sont pas sûrs que le même test aujourd'hui produise les mêmes résultats. Une étude comparative de ces deux SGBD sera le but du chapitre suivant.

Chapitre 03

Mise en œuvre et étude de données NoSQL

1. Introduction

Dans ce chapitre, nous allons présenter notre contribution qui se déroule en trois phases principales :

i. Etude de données NoSQL sur le Framework Hadoop/Mapreduce selon deux modes : un cluster Hadoop en single node et un cluster Hadoop en multi node. Une comparaison de résultat devra être établie entre ces deux modes.
ii. Etude de données NoSQL sur le Framework Hadoop en utilisant le SGBD NoSQL HBase qui joue le rôle très important pour le stockage efficace de données volumineuses.
iii. Etude de données NoSQL sur MongoDB qui représente le meilleur SGBD NoSQL. Ce dernier stocke ses données orientées documents en format JSON. Dans cette étude nous allons analyser le temps d'exécution de requêtes.

Nous terminerons ce chapitre par une discussion de résultat obtenu dans ces trois phases d'étude de données NoSQL.

2. Les logiciels utilisés pour la manipulation de données NoSQL

Avant de présenter notre étude sur les données NoSQL via l'environnement distribué Hadoop et les SGBD NoSQL, nous devons présenter en premier lieu les logiciels et les langages nécessaires pour cette étude.

Toutes nos installations et nos tests seront réalisés sur des machines physiques (un micro-ordinateur) disposant de :

- M1 : CPU Intel Core i5 et d'un processeur de 2.50 GHz et ayant 4 Go de RAM avec un disque dur de 750 Go.
- M2 : CPU Intel Core i5 et d'un processeur de 2.40 GHz et ayant 4 Go de RAM avec un disque dur de 500 Go.
- M3 : CPU Intel Atom N260 et d'un processeur de 1.60 GHz et ayant 2 Go de RAM avec un disque dur de 300 Go.

Ces machines sont dotées du système d'exploitation Linux avec la distribution Ubuntu[1] 16.04.3 LTS. Hormis que linux est un logiciel open source, gratuit et offre une sécurité supérieure, ce choix est justifié par son robustesse et son efficacité de gestion des données massives.

[1]https://www.ubuntu.com/download/server

De plus, nous avons besoins de mettre en œuvre l'environnement distribué Hadoop Apache[2] version 2.9.0 et l'utilisation du langage java via IDE Eclipse Oxygene[3] pour l'implémentation de deux principaux programmes d'Hadoop qui sont : le Map et le Reduce. A travers ces deux programmes, nous pouvons manipuler les données NoSQL qui vont être stockées dans le système de gestion de fichiers distribué d'Hadoop (HDFS) sous forme de fichiers.

Nous avons cité dans le chapitre précédent, que nous avons choisi deux principaux SGBD NoSQL: Hbase et MongoDB. Deux stratégies existantes pour assurer la distribution de données volumineuses : l'une consiste à utiliser des machines physiques (appelées nœuds) forment un réseau et la deuxième stratégie sert à utiliser un logiciel de virtualisation des machines sur la même machine physique.

Dans ce travail, nous avons essayé d'utiliser la deuxième stratégie pour utiliser plusieurs machines virtuelles via l'outil VMware Workstation 11, mais nous avons rencontré un problème de blocage de la machine physique puisque cette stratégie consomme beaucoup de mémoire ce qui implique le besoin d'utilisation d'une machine très puissante en termes de processeur installé, la taille de RAM et de disque dur. Manipuler les données NoSQL nécessite l'utilisation des serveurs très performants. Dans ce contexte, nous avons utilisé trois PC dont l'un est considéré comme Maître 'Master node' et deux autres sont des esclaves (leurs caractéristiques ont été mentionnées au début de cette section).

Nous allons présenter en détail l'utilisation de ces différents logiciels dans les prochaines sections. Le tableau suivant résume les différents logiciels nécessaires pour la manipulation et la gestion de données NoSQL.

Logiciels utilisés	Description
	Ubuntu est la distribution GNU/Linux la plus populaire, basée sur Debian et sponsorisée par la société Canonical. Ubuntu 16.04 LTS c'est une nouvelle version en 2017 basée sur le kernel Linux 3.8.8. Elle améliore l'ergonomie à l'aide de "quicklists".
	Hadoop est un projet open source (Fondation Apache) dédié au calcul distribué. Il assure la gestion et la manipulation distribuées de données volumineuses. La version utilisée est Hadoop 2.9.0

[2] http://hadoop.apache.org/

[3] http://www.eclipse.org/downloads/packages/release/Oxygen/M7

HBASE	HBase est un SGBD open source d'Apache destiné à manipuler un grand nombre de tables de données. Ses données sont orientées colonnes proches du modèle relationnel. La version HBase utilisée est 1.2.6.
mongoDB	MongoDB est un SGBD de données NoSQL orientées documents, composée d'un serveur MongoDB et une base de données formée d'un ensemble de collections de documents.
Welcome to Robo 3T 1.2 	Robo3T verstion 1.2, est un outil de gestion le plus populaire, facilitant la manipulation de données de MongoDB.
	Eclipse Oxygene est une nouvelle version d'Eclipse qui porte de nombreuses améliorations dans les fonctionnalités et les performances. Il inclut de nouveaux outils pour l'analyse de code Java et peut être étendu pour prendre en charge le développement Java 9.

Table 3.1. *Les logiciels utilisés*

Dans le reste de ce chapitre, nous allons présenter notre étude de données NoSQL sur trois types de stockage : le HDFS du framework Hadoop qui manipule les fichiers, le SGBD Hbase avec ses données orientées colonnes et le SGBD MongoDB qu'il occupe la 1ère place des SGBD NoSQL, dont ses données sont orientées documents.

Nous avons planifie de faire une étude supplémentaire qui consiste à comparer la performance entre les quatre types de données NoSQL (données orientées clé-valeur via le SGBD Radis, données orientées colonnes, données orientées documents et données orientées graphe via SGBD Neo4j) mais malheureusement nous ne sommes pas arrivé à faire cette étude parce qu'elle prend beaucoup temps à apprendre et chacun ayant sa propre architecture avec des fonctionnalités différentes. En outre, nous avons rencontré des difficultés dans leur installation sur un seul PC qui doit être très puissant, et la distribution de données sur plusieurs nœuds.

En effet, nous focalisons notre étude sur deux types de données NoSQL : les données orientées colonnes sur HBase et les données orientées documents sur MongoDB. Avant tout, nous étudions dans un premier temps la manipulation de données sur Hadoop via son système de stockage de données HDFS.

3. Installation et configuration de cluster Hadoop

Avant d'entamer à l'étude de données NoSQL sur Hadoop, nous allons présenter l'installation et la configuration de cluster Hadoop selon deux modes : single node et multinode. Un cluster Hadoop se compose de plusieurs machines virtuelles (nœuds) utilisées pour le traitement distribué de tâches (datanode, namenode, secondarynode, etc). L'installation et la configuration des nœuds individuels dépendent de type d'application (Hadoop, Hive, HBase, etc) et le type d'architecture de cluster (single node, multinode).

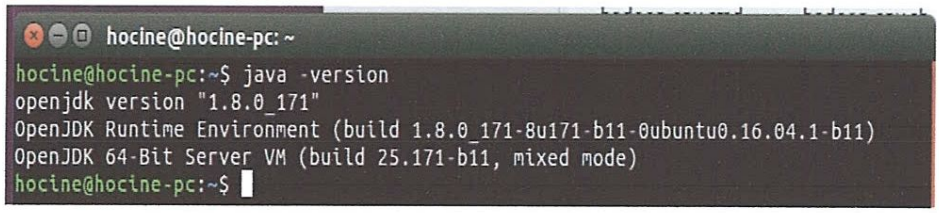
Pour installer le framework Hadoop, des pré-requis doivent être prise en compte:

I. Installation de mise à jour du système Ubuntu 16.04 LTS.
II. Installation de java.
III. Installation de protocole SSH.
IV. Installation de Resync.

I. Installation de mise à jour du système Ubuntu : La mise à jour du système Ubuntu est faite par la commande suivante :

II. Installation de java : L'installation de la dernière version de java via la commande suivante :

sudo apt-get install default-jdk. Une fois l'installation terminée, nous pouvons vérifier la version de java (1.8.0_171) par la commande java –version



III. Installation de SSH : le SSH (Secure Shell), est un protocole utilisé pour se connecter de manière sécurisée sur des systèmes distants. C'est le moyen le plus courant d'accéder à des serveurs distants Linux et Unix (cf. Annexe A).

Pour fonctionner de manière transparente (connexion sans mot de passe), nous devons générer la clé RSA de SSH (cf. Annexe A).

Après la génération de cette clé, nous devons activer l'accès au SSH par l'insertion de sa clé dans la liste des clés autorisées en utilisant les deux commandes suivantes :

```
hocine@hocine-pc:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
hocine@hocine-pc:~$ chmod 0600 ~/.ssh/authorized_keys
```

On peut tester le bon fonctionnement de SSH par la commande ssh localhost. La figure suivante illustre le résultat de ce test.

```
hocine@hocine-v: ~
hocine@hocine-v:~$ ssh localhost
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.13.0-43-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 paquet peut être mis à jour.
0 mise à jour de sécurité.

Last login: Thu Jun  7 04:35:39 2018 from 127.0.0.1
hocine@hocine-v:~$
```

Figure 3.1 : *Vérification d'installation de SSH*

IV. Installation de Resync : le Resync est un outil de copie de fichiers vers ou depuis un autre hôte sur n'importe quel shell distant. Il a pour but de réduire la quantité de données envoyées sur le réseau en envoyant seulement les différences entre les fichiers sources et les fichiers existants dans la destination (cf. Annexe A).

Après avoir appliqué les pré-requis nécessaires pour Hadoop, l'installation de ce framework consiste à télécharger son fichier compressé (hadoop-2.9.0.tat.gz) depuis le site Apache et extraire son contenu dans le répertoire /usr/local/hadoop comme le montre la figure suivante.

```
hocine@hocine-pc:~$ tar xzf '/home/hocine/Téléchargements/hadoop-2.9.0.tar.gz'
hocine@hocine-pc:~$ ls
Bureau              examples.desktop          index.html  studio3t
Documents           hadoop-2.9.0              Modèles     Téléchargements
eclipse             hadoop-2.9.0.tar.gz.mds   Musique     Vidéos
eclipse-workspace   Images                    Public      vmware
hocine@hocine-pc:~$ mv hadoop-2.9.0 hadoop
```

Figure 3.2 : *Installation de Hadoop 2.9.0*

Finalement, nous devons ajuster les paramètres des fichiers de configuration du cluster Hadoop selon deux modes : single node et multi nodes.

3.1. Configuration de cluster Hadoop en Single Node

Dans le mode d'utilisation de cluster Hadoop en single node, une seule machine joue le rôle du maître et d'esclave. La mise en place du cluster Hadoop implique d'ajuster 05 fichiers de configuration :

1).bashrc : définir le chemin d'Hadoop comme suit :

```
Export HADOOP_HOME=/home/hocine /hadoop
Export PATH= $PATH:$HADOOP_HOME/bin
```

2) hadoop-env.sh : pour exporter le chemin de JDK.

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

3) core-site.xml : Ajuster le port de local host à 9000.

```
<configuration>
    <property>
       <name>fs.defaultFS</name>
       <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

4) hdfs-site.xml: Ajuster les paramètres de réplication. Dans ce mode la valeur vaut 01.

```
<configuration>
 <property>
      <name>dfs.replication</name>
      <value>1</value>
</property>
</configuration>
```

5) mapred-site.xml: Ajuster les paramètres d'exécution de mapreduce sur hadoop yarn.

```
<configuration>
   <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
   </property>
</configuration>
```

6) yarn-site.xml: Ajuster les propriétés de Yarn.

```
<configuration>
<!-- Site specific YARN configuration properties -->
 <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
</property>
</configuration>
```

L'architecture en couches d'Hadoop en single node est illustrée dans la figure suivante.

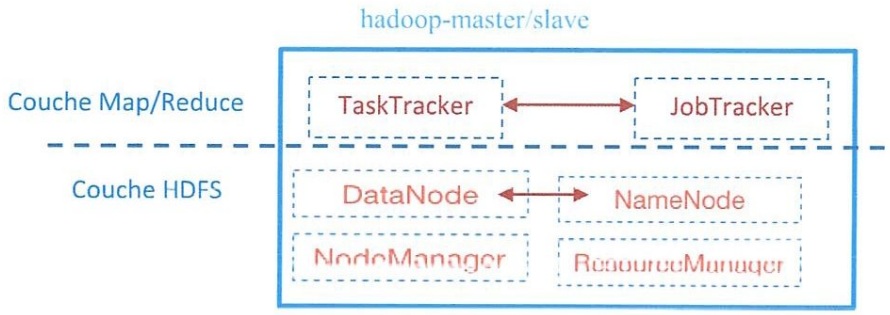


Figure 3.3 : *Architecture en couches d'Hadoop en single node*

L'architecture d'Hadoop en single node est divisée en deux couches : la couche Map/Reduce assure l'exécution de JobTracker de façon pseudo distribuée et déclenche des TaskTracker. La couche HDFS pour le stockage de données dans un seul nœud de données (DataNode) qui est associé avec son gestionnaire de nœud (NodeManager). Le NameNode de la couche HDFS sert à assurer la distribution et la réplication des blocs.

3.2. Configuration de cluster Hadoop en Multi Node

Le but principal du framework Hadoop est d'assurer la distribution de données volumineuses et la distribution de traitement afin d'en optimiser la performance de la gestion de ce type de données. Dans ce contexte, nous devons présenter dans un premier lieu l'architecture distribuée de cluster Hadoop en Multi nœuds (multi node), plus précisement cluster Hadoop en 3 nodes (un master node et 2 slaves). Pour cela, nous avons utilisé trois machines relient entre eux via un routeur comme point d'accès.

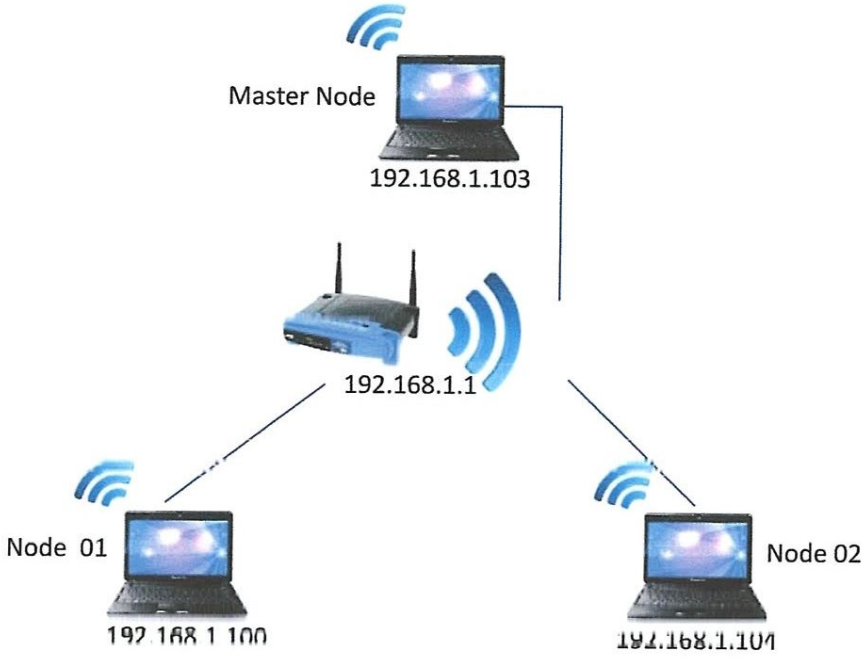


Figure 3.4 : *Architecture physique d'Hadoop en trois nœuds*

La configuration de cluster Hadoop en trois noeuds nécessite l'adaption de fichiers de configurations selon le type de chaque nœud (Master node et 2 slaves nodes). Nous appliquons des modifications sur les mêmes fichiers de configuration présentés dans le cas de cluster hadoop en single node .

Le nœud maître (master node) agira aussi comme un esclave, et les deux autres nœuds deviendront seulement des esclaves. La configuration de cluster multi node consiste à configurer le nœud maître et les nœuds esclaves.

Dans la configuration du nœud maitre, nous devons modifier le fichier */etc/hosts* pour ajouter l'adresse IP des trois nœuds.

```
192.168.1.103 node-master
192.168.1.100 node1
192.168.1.104 node2

127.0.0.1       localhost
127.0.1.1       hocine-V
```

Puis, nous devons générer la clé d'authentification RSA de SSH par la commande :

```
hocine@hocine-V:~$ ssh-keygen -b 4096
```

L'étape suivante sert à copier la clé générée sur les trois nœuds. En utilisant la commande suivante pour copier la clé RSA du noeud-maître sur lui-même (hadoop@node-master) de sorte que nous puissions également l'utiliser en tant que DataNode si nécessaire.

```
hocine@hocine-V:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hocine@node-master
```

Cette clé a été également copier dans les deux nœuds esclaves (hadoop@node1 et hadoop@node2) par les deux commandes suivantes :

```
hocine@hocine-V:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hocine@node1
```

```
hocine@hocine-V:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hocine@node2
```

Nous présentons deux exemples de fichiers de configurations à modifier:

1) core-site.xml : Ajuster le port de nœud maitre à 9000.

```
<configuration>
        <property>
            <name>fs.default.name</name>
            <value>hdfs://node-master:9000</value>
        </property>
</configuration>
```

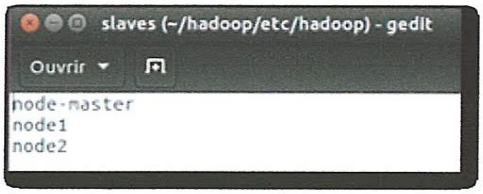
2) hdfs-site.xml: Ajuster les paramètres de réplication sur les différents nœuds.

```
<configuration>
<property>
    <name>dfs.replication</name>
    <value>2</value>
</property>
<property>
     <name>dfs.namenode.name.dir</name>
     <value>file:///home/hocine/hadoop/yarn_data/hdfs/namenode</value>
</property>

<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///home/hocine/hadoop/yarn_data/hdfs/datanode</value>
</property>

</configuration>
```

Pour l'ajustement de fichiers de configuration au niveau de slave node, ses fichiers sont utilisés par les scripts de démarrage pour démarrer les nœuds requis.



Après avoir installé et configuré le cluster Hadoop en single node et en multi node, la section suivante présente l'étude de données sur Hadoop.

4. Etude de données sur Hadoop/MapReduce

Le framework Hadoop permet d'assurer le traitement distribué de données volumineuses sur des nœuds via son paradigme de programmation MapReduce. Afin de réaliser notre étude de données sur Hadoop, nous avons choisi les données textuelles pour calculer la fréquence d'apparition de chaque mot dans un texte en s'appuyant sur le Job WordCount. Cette étude se déroule selon deux modes : single node et multi nodes.

Les données textuelles d'entrée sont stockées dans le HDFS sous forme de fichier texte nommé Nom.txt qui contient des noms des personnes. Nous avons utilisé un fichier de nom existant sur le web et nous avons dupliqué son contenu plusieurs fois pour augmenter sa taille qui devient 142 Mo avec plus de 20 millions de noms. La figure suivante illustre l'insertion du fichier nom.txt selon deux modes : single node (partie A) et multi node (partie B).

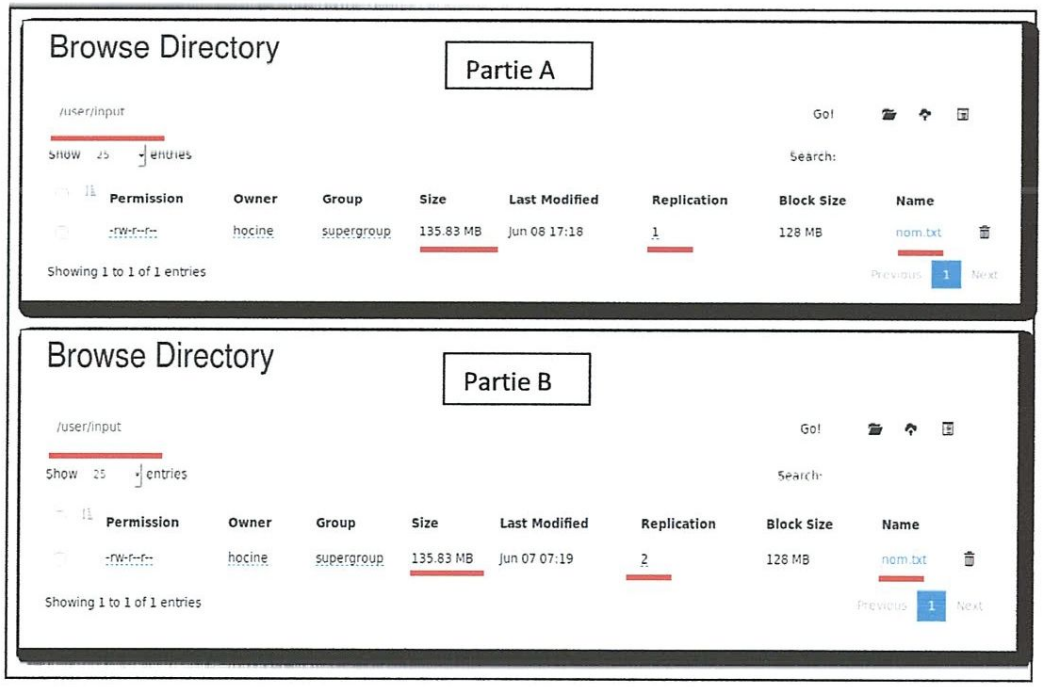


Figure 3.5 : *Insertion de données dans le HDFS selon le mode single node et multi node*

D'après la figure ci-dessus, nous constatons que la taille de bloc de données est 128 MB, alors que sur des systèmes classiques, la taille est généralement de 64 Ko. L'intérêt de fournir aux

blocs des tailles plus grandes permet de réduire le temps d'accès par rapport les blocs de petites tailles.

4.1. Etude de données sur cluster Hadoop single node

Dans cette étude, nous voulons exécuter le Job Wordcount écrit sous forme d'un fichier .jar via éclipse. Ce fichier comprend trois classes java : TokenizerMapper qui hérite la classe Mapper d'Hadoop, IntSumReducer hérite la classe Reducer et la classe WordCount qui est la classe principale (inclus la méthode main) (cf. Annexe A).

Comme nous avons mentionné dans les chapitres précédents que Hadoop est basé sur deux fonctions principales :

- Map prend un ensemble de données en entrée et le diviser en un autre ensemble de données des paires (clé-valeur) permettant d'exécuter le même traitement en parallèle : Map (clé, valeur) -> liste (iclé, ivaleur)
- Reduce permet d'agréger les données traitées par Map selon les clés similaires : Reducc(iclé, list(ivaleur)) –> list(iclé, fvaleur). Le résultat final est le regroupement de tous les résultats de Reduce.

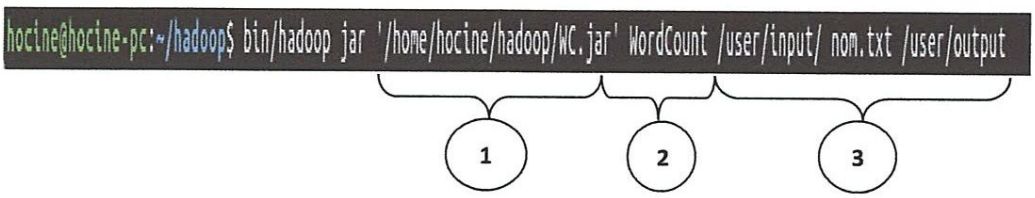
Avant d'exécuter le JobTracker sur un seul nœud (localhost), nous devons formater le name node du système HDFS par la commande : sudo hdfs Hadoop namenode –format.

Le démarrage de cluster Hadoop se fait à travers les fichiers d'hadoop tels que dfs.sh (concerne les name node, data node) et yarn.sh (pour fonctionner le système Yarn d'Hadoop). La figure suivante présente le démarrage du cluster Hadoop en single node.

```
hocine@hocine-pc:~/hadoop$ sbin/start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hocine/hadoop/logs/hadoop-hocine-namenode-hocine-pc.out
localhost: starting datanode, logging to /home/hocine/hadoop/logs/hadoop-hocine-datanode-hocine-pc.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/hocine/hadoop/logs/hadoop-hocine-secondarynamenode-hocine-pc.out
starting yarn daemons
starting resourcemanager, logging to /home/hocine/hadoop/logs/yarn-hocine-resourcemanager-hocine-pc.out
localhost: starting nodemanager, logging to /home/hocine/hadoop/logs/yarn-hocine-nodemanager-hocine-pc.out
```

Figure 3.6 : *Démarrage de cluster Hadoop sur un seul nœud*

Après le démarrage de cluster Hadoop, l'étape suivante sert à lancer le processus JobTracker qui va se charger de l'ordonnancement des traitements et de la gestion de l'ensemble des ressources du système. Il reçoit les tâches MapReduce à exécuter sous forme de fichier nommé WC.jar (le chemin est représenté par (1)) ainsi que la classe principale WordCount (2) et le répertoire où stocker les données de sorties (3).



Le JobTracker du cluster Hadoop est en relation avec le namenode d'HDFS afin de distribuer les traitements sous forme de tâches appelées TaskTracker. Dans le cas de cluster Hadoop en single node, le jobtracker et le tasktracker s'exécuter sur le même nœud (LocalHost).

La figure suivante illustre une partie de résultat de l'exécution en single node.

```
18/06/10 04:19:42 INFO mapreduce.Job: Job job_1528600680585_0001 running in uber mode : false
18/06/10 04:19:42 INFO mapreduce.Job:  map 0% reduce 0%
18/06/10 04:20:12 INFO mapreduce.Job:  map 12% reduce 0%
18/06/10 04:20:18 INFO mapreduce.Job:  map 23% reduce 0%
18/06/10 04:20:24 INFO mapreduce.Job:  map 33% reduce 0%
18/06/10 04:20:30 INFO mapreduce.Job:  map 45% reduce 0%
18/06/10 04:20:36 INFO mapreduce.Job:  map 64% reduce 0%
18/06/10 04:20:41 INFO mapreduce.Job:  map 100% reduce 0%
18/06/10 04:20:56 INFO mapreduce.Job:  map 100% reduce 100%
18/06/10 04:20:56 INFO mapreduce.Job: Job job_1528600680585_0001 completed successfully
18/06/10 04:20:57 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=518352
                FILE: Number of bytes written=986105
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=142425706
                HDFS: Number of bytes written=59870
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=55197
                Total time spent by all reduces in occupied slots (ms)=12255
                Total time spent by all map tasks (ms)=55197
                Total time spent by all reduce tasks (ms)=12255
                Total vcore-milliseconds taken by all map tasks=55197
```

Figure 3.7 : *Résultat de l'exécution de Job WordCount sur un seul nœud*

D'après la figure ci-dessus, nous remarquons que l'exécution de la fonction Map se fait d'une façon progressive par contre la fonction reduce s'exécute directement quand la proportion de l'exécution de Map atteint 100%. Dans un cluster en single node, le nombre de tasktracker soit 1 (map tasks=1, reduce tasks=1). Le temps d'exécution de tous les map est 55.197ms (ou 0.055s). Ainsi que le temps d'exécution de tous les reduce est 12.255ms.

L'un des critères de performance d'un cluster est le temps d'exécution de map et de reduce. Utiliser Hadoop sur une seule machine donne un meilleur résultat pour la manipulation de données volumineuses par rapport à l'utilisation d'un SGBD. De plus, le Hadoop sur un seul

nœud permet de diviser le JobTracker sur plusieurs fonctions de Map et Reduce. La figure suivante illustre un exemple de fonctionnement d'Hadoop en single node sur les données textuelles utilisées dans ce travail.

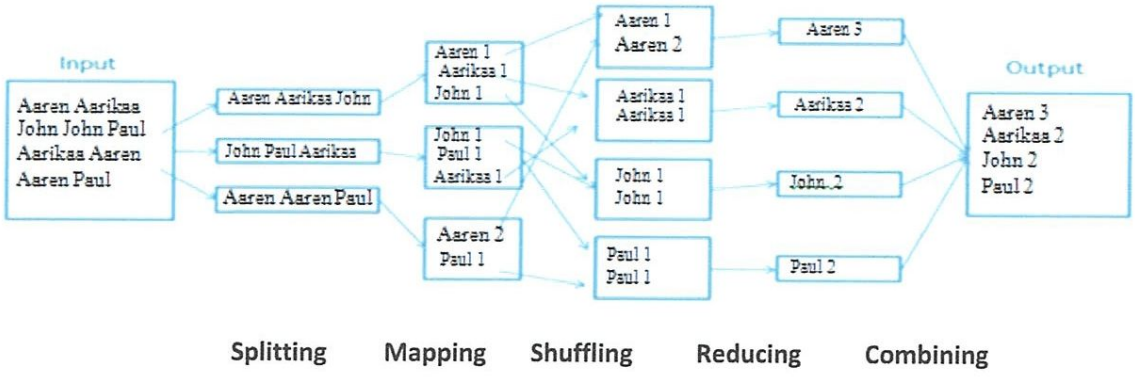


Figure 3.8: *Exemple de fonctionnement du cluster Hadoop*

Splitting: est un paramètre de division (il peut être n'importe quoi, espace, virgule, point-virgule.) pour la préparation de l'exécution de la tâche map.

Shuffling : l'ensemble du processus en parallèle sur différents clusters. Afin de les regrouper en "Phase de réduction", les données KEY similaires doivent être sur le même cluster.

La figure suivante présente le fichier de résultat part-r-00000 via l'interface graphique du HDFS.

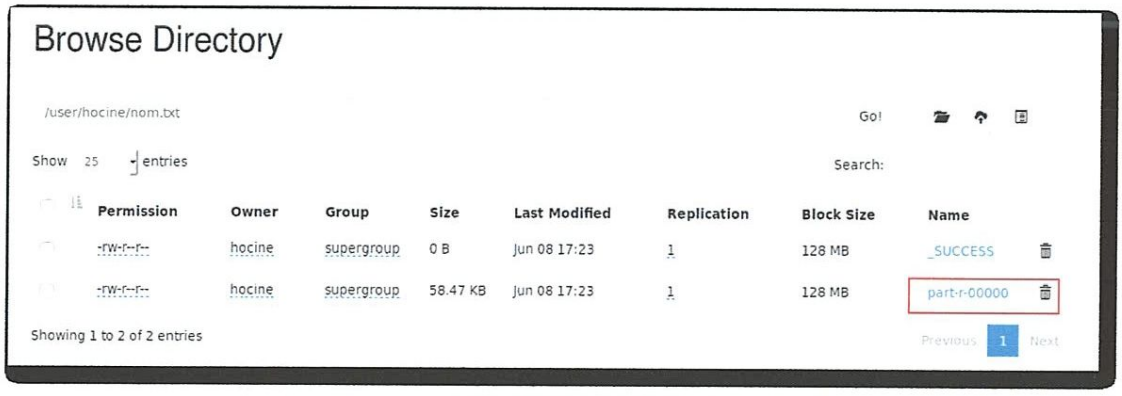


Figure 3.9 : *Représentation de fichier résultat via l'interface graphique*

Nous pouvons visualiser le contenu du fichier de sortie part-r-00000.txt en Annexe A.

4.2. Etude de données sur cluster Hadoop multi node

Dans cette section, nous voulons étudier les données sur un cluster Hadoop multi node qui se compose de trois nœuds : 1 master node et deux slaves nommés node1 et node2. Le démarrage de ce cluster se fait de la même façon qu'un cluster en single node, sauf que l'exécution des tâches Jobtracker se fait dans le master node et les slaves node 1 et node 2.

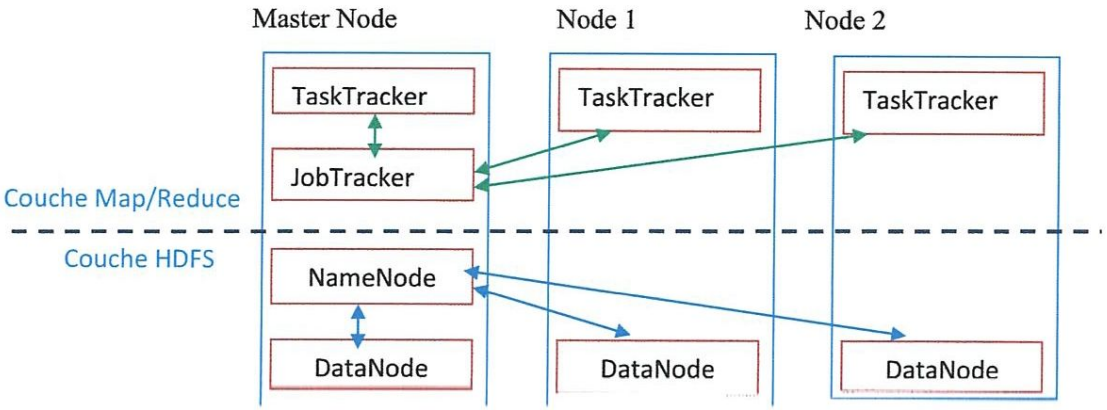


Figure 3.10 : *Architecture en couche de Cluster Hadoop sur 3 nodes*

Le déroulement de démarrage de cluster à trois nœuds est en Annexe A.

La figure suivante présente le résultat de l'exécution de job Wordcount via le navigateur web.

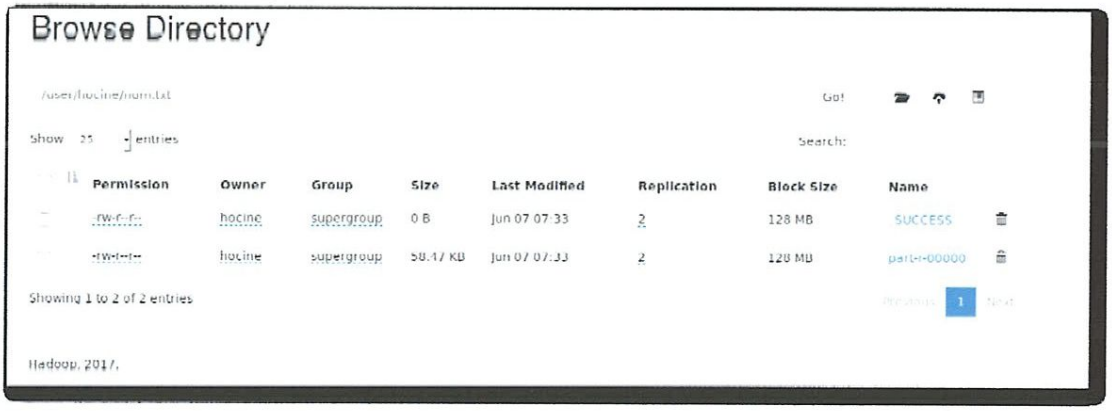


Figure 3.11 : *Résultat de l'exécution de Job WordCount sur trois nœuds*

Pour bien étudier les données sur Hadoop/mapreduce, nous avons effectué une comparaison entre le cluster en single node, 2 nodes et 3 nodes en termes de temps d'exécution de map, de reduce et de temps d'exécution total qui égale au temps de splitings + mapping + shuffling + reducer (Une partie de résultat d'exécution de Job wordcount sur deux nœuds est en Annexe A). Le résultat de l'exécution se résume dans la table suivante :

Nombre de nœuds	Temps d'exécution Map (ms)	Temps d'exécution Reduce (ms)	Temps d'exécution total (ms)
1 Nœud	55.197	12.255	104.000
2 nœuds	29.524	10.072	56.000
3 nœuds	28.045	3.129	48.000

Table 3.2. *Comparaison entre les cluster Hadoop*

A partir de la table ci-dessus, nous observons que le temps d'exécution de job donne un meilleur résultat dans le cluster à 3 nœuds (48.000 ms) par rapport les clusters à 2 nœuds (56.000ms) et single node (104.000). De plus, le temps d'exécution de job sur 2 nœuds est meilleur que dans un seul nœud. Par ailleurs, le temps d'exécution de map sur un cluster à 2 nodes et un cluster à 3 nodes sont très proches (l'écart de 1.479 ms) par contre le temps d'exécution de reduce sur 3 nodes est meilleur que dans le cluster à 2 nodes. Selon ces deux remarques nous pouvons conclure que le temps d'exécution d'un job est lié non seulement aux temps de map et reduce mais aussi au temps d'exécution des opérations entre le Map et le reduce représentées par le processus Shuffling qui regroupe les sous résultats de Map.

Utiliser un nombre important de nœuds pour le traitement de données volumineuses sur Hadoop améliore significativement le traitement des données et réduire le temps de réponse avec moins de complexité contrairement aux bases de données réparties. Le Hadoop a été développé pour accéder à des gros volumes de données en des temps raisonnables à l'aide de traitement distribué du paradigme MapReduce.

Les données sont répliquées sur plusieurs nœuds différents afin d'une part d'assurer la disponibilité et la performance et d'autre part d'éviter les pertes d'information en cas d'une panne. Dans cette optique, nous avons réalisé une étude de temps de réponse en cas de défaillance d'un ou de plusieurs nœuds. La table suivante présente le résultat de cette étude.

Nombre de nœuds en panne	Temps d'exécution Map (ms)	Temps d'exécution Reduce (ms)	Temps d'exécution total (ms)
0 nœud/3	28.045	3.129	48.000
1 nœud / 3	29.688	9.283	55.000
2 nœuds / 3	45.747	13.056	74.000

Table 3.3. *Etude de tolérance aux pannes d'un cluster à 3 nœuds*

D'après la table ci-dessus, nous constatons que lorsqu'un nœud tombe en panne dans une architecture à trois nœuds, le temps d'exécution total s'augmente avec 14,53% par rapport

l'état initial de cluster (3 nodes présentées dans la 1ère ligne). Dans le cas où 2 nœuds tombent en panne le temps d'exécution s'accroit à 54, 16%.

Selon les deux résultats obtenus dans les deux tables 3.2 et 3.3, nous pouvons déduire le temps de reprise après la panne par la comparaison de temps d'exécution Maps selon la table suivante.

Nombre de nœuds en panne	Temps de reprise après la panne (ms)
1 nœud / 3	0.164
2 nœuds / 3	9.45

Table 3.4. *Temps de reprise après la panne*

D'après la table en haut, nous remarquons que le temps de reprise après la panne d'un seul nœud est très court (0.164 ms) contrairement au temps de reprise après la panne de deux nœuds. Nous déduisons que le temps de reprise après la panne augmente proportionnellement par rapport au nombre de nœuds en panne.

Le framework Hadoop ne peut effectuer qu'un traitement distribué (par blocs) de données volumineuses qui ne seront accessibles que de manière séquentielle. Cela signifie que l'on doit utiliser un SGBD NoSQL assurant le stockage efficace de ce type de données. Dans ce contexte, nous allons présenter dans la section suivante comment stocker les données NoSQL sur Hadoop en utilisant le SGBD NoSQL HBase.

5. Etude de données NoSQL sur Hadoop/ HBase

Le Hbase est un SGBD NoSQL et un sous projet non intégré du framework Hadoop permettant d'assurer l'accès aléatoire en temps réel aux données du HDFS. Il permet de gérer les données volumineuses NoSQL de type colonnes inspiré de la base BigTable de Google. Le démarrage de HBase demande la modification de quelques fichiers de configuration de Hadoop. Nous présentons dans la sous-section suivante les étapes d'installation de Hbase sous Ubuntu puis nous étudierons les données NoSQL orientées colonnes avec une comparaison de performance entre les données stockées directement sur HDFS et les données stockées sur HBase.

5.1. Installation de HBase

L'installation de HBase nécessite des préinstallassions de Java et d'Hadoop. Après avoir téléchargé le fichier compressé HBase depuis le site Apache[4], nous devons l'extraire par la commande suivante:

```
hocine@hocine-pc:~$ tar -zxvf '/home/hocine/Téléchargements/hbase-1.2.6-bin.tar.gz'
```

On déplace le contenu de ce fichier dans le répertoire /usr/local et on effectue des modifications sur les fichiers de configuration de Hadoop et de Hbase.

Nous devons ajouter les variables d'environnement d'HBase dans le fichier d'Hadoop .bashrc (cf. Annexe B). Les modifications des fichiers de configuration de HBase sont présentées comme suit :

1. hbase-env.sh : Ajouter la variable d'environnement de Java par l'instruction

   ```
   export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
   ```

2. hbase-site.xml : Ajouter les propriétés hbase.rootdir et hbase.zookeeper.property. dataDir (cf. Annexe B).

Le démarrage de HBase se fait par la commande : bin/start-hbase.sh. La figure suivante illustre les composants d'HBase via la commande jps.

```
hocine@hocine-pc:~/hbase$ jps
30707 HMaster
29908 ResourceManager
30052 NodeManager
31157 Jps
29737 SecondaryNameNode
30842 HRegionServer
29532 DataNode
30637 HQuorumPeer
29373 NameNode
```

Figure 3.12: *Visualisation de composants de HBase*

L'interface Hbase est exécutée par le serveur sur l'adresse http://localhost:16010 comme le montre la figure suivante.

[4] http://www.interior-dsgn.com/apache/hbase/stable/

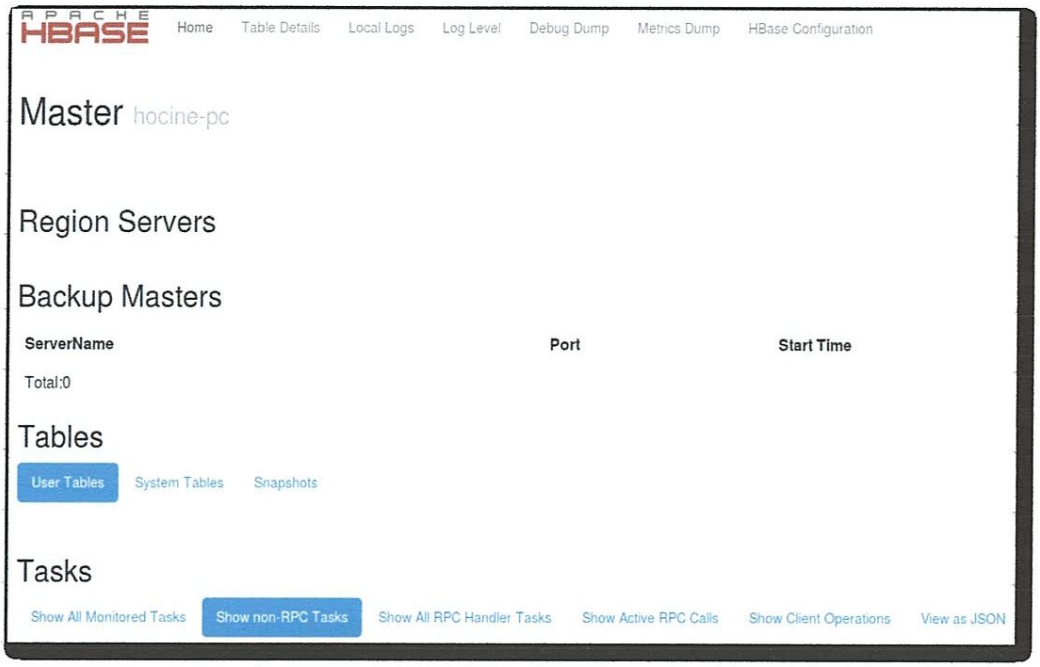


***Figure 3.13:** Interface de HBase*

5.2. Stockage de données sur HBase

Après avoir installé le cluster Hadoop et le SGBD HBase, nous présentons dans cette sous-section comment créer une base de données HBase, insérer et afficher les données.

Dans ce cadre, nous voulons utiliser les mêmes données textuelles de Job Wordcount pour créer une table HBase qui se compose d'une seule column family (famille de colonne) baptisée : Personal. Le fichier texte de prénom comprend plus de 20 millions prénoms. Pour cela, nous devons proposer une méthode de génération automatique d'une base de données HBase à partir d'un texte. Ce travail ce n'ai pas notre objectif d'une part, et il demande plus de temps qui peut être vu comme un autre sujet de Master d'autre part. Dans ce cadre, nous avons créé manuellement 31 enregistrements via la méthode put de HBase. La figure suivante illustre la création de la base de données orientée colonnes.

```
hbase(main):003:0> create 'personal', 'id' , 'nam'
0 row(s) in 12.8830 seconds

=> Hbase::Table - personal
hbase(main):004:0> scan 'personal'
ROW                    COLUMN+CELL
0 row(s) in 14.7890 seconds

hbase(main):005:0> put 'personal' , '1' , 'nam' , 'hocine'
0 row(s) in 2.3250 seconds
```

***Figure 3.14:** Exemple de création de HBase et insertion de données*

Nous remarquons que le temps de création de la base données Hbase est assez long 12.8830 seconde. Cela est revient à la puissance minimale de la machine physique utilisé pour manipuler ces données orientées colonnes. D'après cette remarque, nous évaluons le temps de lecture de données de la base de données à travers la commande scan.

La figure suivante présente le contenu de la base de données ainsi que le temps de lecture.

```
hocine@hocine-pc: ~/hbase
hbase(main):033:0> scan 'Hocine'
ROW                    COLUMN+CELL
 1                     column=Nom:, timestamp=1529501759821, value=amine
 10                    column=Nom:, timestamp=1529527843481, value=addy
 11                    column=Nom:, timestamp=1529527857842, value=ada
 12                    column=Nom:, timestamp=1529527879234, value=adele
 13                    column=Nom:, timestamp=1529527922832, value=aggi
 14                    column=Nom:, timestamp=1529527936817, value=agna
 15                    column=Nom:, timestamp=1529527951043, value=aida
 16                    column=Nom:, timestamp=1529527969435, value=aidan
 17                    column=Nom:, timestamp=1529527981194, value=aila
 18                    column=Nom:, timestamp=1529528009118, value=alana
 19                    column=Nom:, timestamp=1529528024237, value=alane
 2                     column=Nom:, timestamp=1529501793035, value=ali
 20                    column=Nom:, timestamp=1529528036485, value=alecia
 21                    column=Nom:, timestamp=1529528077074, value=alena
 22                    column=Nom:, timestamp=1529528098873, value=alexis
 23                    column=Nom:, timestamp=1529528119001, value=alexina
 24                    column=Nom:, timestamp=1529528132005, value=alia
 25                    column=Nom:, timestamp=1529528144137, value=imade
 26                    column=Nom:, timestamp=1529528157548, value=alis
 27                    column=Nom:, timestamp=1529528179566, value=allx
 28                    column=Nom:, timestamp=1529528193719, value=alma
 29                    column=Nom:, timestamp=1529528264188, value=alma
 3                     column=Nom:, timestamp=1529527704728, value=Aaren
 30                    column=Nom:, timestamp=1529528271719, value=alma
 31                    column=Nom:, timestamp=1529528277755, value=alma
 4                     column=Nom:, timestamp=1529527736445, value=Aarika
 5                     column=Nom:, timestamp=1529527755990, value=Abaqael
 6                     column=Nom:, timestamp=1529527774197, value=Abagail
 7                     column=Nom:, timestamp=1529527792821, value=Abbe
 8                     column=Nom:, timestamp=1529527814595, value=Abbey
 9                     column=Nom:, timestamp=1529527826672, value=abbi
31 row(s) in 0.0580 seconds
```

Figure 3.15: *Lecture de données de HBase*

D'après le résultat ci-dessus, nous constatons que le temps de lecture est 0.0580 s. Ce temps d'exécution est assez long contrairement aux points forts de HBASE. En réalité, le HBase permet d'améliorer le stockage de données NoSQL et fournit des capacités aléatoires de lecture/écriture en temps réel. Un point négatif de l'utilisation de données NoSQL est qu'il exige des machines très puissantes permettant de manipuler des milliards de données avec un temps court par rapport les bases de données réparties. Dans ce contexte, notre étude de données NoSQL sur HBase s'arrête ici.

D'autres SGBD NoSQL ont été proposés et qui ne sont pas basés sur le Hadoop pour le traitement distribué de données. Un exemple de ces SGBD est MongoDB qui stocke et traite ses données sous forme de documents en format binaire de JSON appelé BSON.

Dans la section suivante, nous voulons étudier les données de MongoDB qui est le meilleur SGBD NoSQL d'après les statistiques des SGBD présentés sur le site https://db-engines.com.

6. Etude de données NoSQL sur MongoDB

Nous étudions dans cette section un deuxième type de données volumineuses. Il s'agit, de données NoSQL orientées documents en utilisant le SGBD MongoDB (parfois appelé base de données mongodb). Ce dernier est gratuit, distribué et open-source, dont ses documents sont similaires à des objets JSON. Un enregistrement (Records) dans MongoDB est un document, qui est une structure de données du pair champ-valeur, dont le champ désigne une propriété d'une collection et la valeur du champ peut inclure des documents, des tableaux, et même des tableaux de documents. Le MongoDB est comme un SGBD, il comporte un langage de définition de données et un langage de manipulation de données. Avant de présenter comment créer et manipuler les données orientées documents, nous présentons dans ce qui suit l'installation de MongoDB.

6.1. Installation de MongoDB

Pour installer MongoDB sous Ubuntu, il suffit de télécharger des packages *.deb propres pour chaque distribution de Linux. Pour utiliser ces packages, nous devons importer la clé publique GPG utilisée par le système de gestion des packages qui garantit la cohérence et l'authenticité des paquets. La commande suivante permet d'importer la clé GPG:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
```

Le résultat d'importation de clé se trouve dans Annexe C. La deuxième étape sert à créer un nouveau dépôt aux sources mongoDB via la commande echo suivante :

```
hocine@hocine-pc:~$    echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/a
pt/ubuntu xenial/mongodb-org/3.6 multiverse" | sudo tee /etc/apt/sources.list.d/
mongodb-org-3.6.list
deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/
3.6 multiverse
hocine@hocine-pc:~$
```

L'étape suivante permet de rafraichir la liste des sources de mongoDB (cf. Annexe C). Par la suite, nous installons les packages MongoDB par la commande : sudo apt-get install –y

mongodb-org. La figure suivante illustre le résultat de l'installation de packages MongoDB.

```
hocine@hocine-pc:~$ sudo apt-get install -y mongodb-org
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  libllvm4.0 linux-headers-4.10.0-28 linux-headers-4.10.0-28-generic
  linux-headers-4.13.0-37 linux-headers-4.13.0-37-generic linux-image-4.10.0-28-generic
  linux-image-4.13.0-37-generic linux-image-extra-4.10.0-28-generic
  linux-image-extra-4.13.0-37-generic
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
Les paquets supplémentaires suivants vont être installés :
  mongodb-org-mongos mongodb-org-server mongodb-org-shell mongodb-org-tools
Les NOUVEAUX paquets suivants seront installés :
  mongodb-org mongodb-org-mongos mongodb-org-server mongodb-org-shell mongodb-org-tools
0 mis à jour, 5 nouvellement installés, 0 à enlever et 35 non mis à jour.
Il est nécessaire de prendre 67.0 Mo dans les archives.
Après cette opération, 277 Mo d'espace disque supplémentaires seront utilisés.
Réception de:1 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6/multiverse amd64 mo
ngodb-org-shell amd64 3.6.4 [8,481 kB]
2% [1 mongodb-org-shell 1,723 kB/8,481 kB 20%]                    189 kB/s 5min 44s
```

Figure 3.16 : *Résultat de l'installation de packages MongoDB*

L'installation du MongoDB est achevé par l'installation de ses packages. La figure suivante présente le résultat de démarrage du serveur mongodb sur le port 27017.

```
hocine@hocine-pc: ~
hocine@hocine-pc:~$ sudo service mongod start
hocine@hocine-pc:~$ mongod
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] MongoDB starting : pid=3832 port=27017 dbpath=/data/db 64-bit host=hocine-pc
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] db version v3.6.3
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] git version: 9586e557d54ef70f9ca4b43c26892cd55257e1a5
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] allocator: tcmalloc
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] modules: none
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] build environment:
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten]     distmod: ubuntu1604
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten]     distarch: x86_64
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten]     target_arch: x86_64
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] options: {}
2018-04-13T11:37:24.364+0100 I STORAGE  [initandlisten] exception in initAndListen: IllegalOperation: Attempted to create a lock file o
n a read-only directory: /data/db, terminating
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] now exiting
2018-04-13T11:37:24.364+0100 I CONTROL  [initandlisten] shutting down with code:100
```

Figure 3.17 : *Résultat de démarrage du serveur MongoDB*

6.2. Manipulation de données NoSQL sur MongoDB

Pour la manipulation et l'étude de données orientées documents de MongoDB à travers une interface graphique, nous avons utilisé l'outil GUI le plus populaire appelé **Robo3T**[5], la version gratuite.

L'interface principale de Robo3T offre des opérations telles que la création d'une nouvelle connexion avec une base de données MongDB, la manipulation de données via des requêtes

[5] https://robomongo.org/

mongoDB, la visualisation de résultats, indexation, etc. Pour manipuler les données sur Robo3T, nous devons créer une connexion avec le serveur MongoDB sur le port 27017 (cf. Annexe C).

De plus, nous avons importé la base de données DOHMH New York City Restaurant qui contient 25357 collections sur les restaurants de New York. Le contenu de cette base de données se trouve dans le fichier compressé restaurants.json-.ZIP que nous avons le télécharger depuis le site OpenData[6]. Pour manipuler les données de restaurant, nous avons créé la base de données "**HocineDB**" et une collection "**restaurants**" et nous exécutons la commande suivante :

```
hocine@hocine-pc:~$ mongoimport --db HocineDB --collection restaurants
--file /home/hocine/Documents/restaurants.json
2018-05-16T15:05:24.122+0100    connected to: localhost
2018-05-16T15:05:24.917+0100    imported 25357 documents
```

Après avoir créé la base de données MongoDB baptisée HocineDB, nous allons présenter comment manipuler les données NoSQL à travers des requêtes mongo.
Comme nous avons mentionné que les documents de MongoDB sont représentés sous format binaire JSON (ou BSON), un exemple de données de la base HocineDB est représenté dans la figure suivante.

```
{
    'Restaurant_id': '40399126',
    'Borough': 'Manhattan',
    'Name':'Cafe',
    'address':
        {
        'Street':'5 Avenue',
        'City':'New York',
        'country':'America',
        'Zip code': '75009'
        }
    'Note': '8'
}
```

Figure 3.18 : *Exemple de documents de la base de données de Restaurant*

Le MongoDB crée automatiquement un champ « _id » avec une valeur unique pour chacun des documents avant d'insérer les documents dans la collection. Pour interroger une base de données MongoDB, nous utilisons la méthode find( ) et à partir de cette méthode on peut filtrer les données selon un certain nombre de conditions.

[6] https://opendata.cityofnewyork.us/

Exemple : soit la requête Mongo suivante : db.restaurants.find({"borough" : "Manhattan"}) qui permet de récupérer tous les restaurants dans l'arrondissement (en anglais borough) est Manhattan. Le résultat de cette requête est présenté dans cette figure.

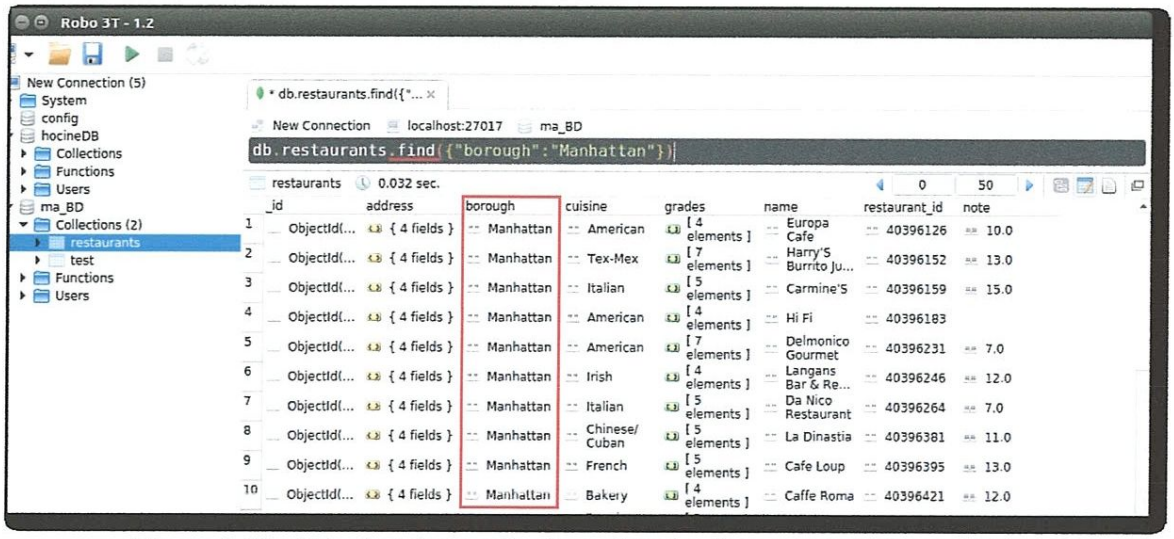


Figure 3.19 : *Résultat de requête de recherche dans une base MongoDB*

D'autres exemples de requête de recherche sont en Annexe C. Nous pouvons réaliser une requête de projection sur quelques champs (ou colonnes) en mettant 1 comme valeur aux champs que nous voulons afficher.

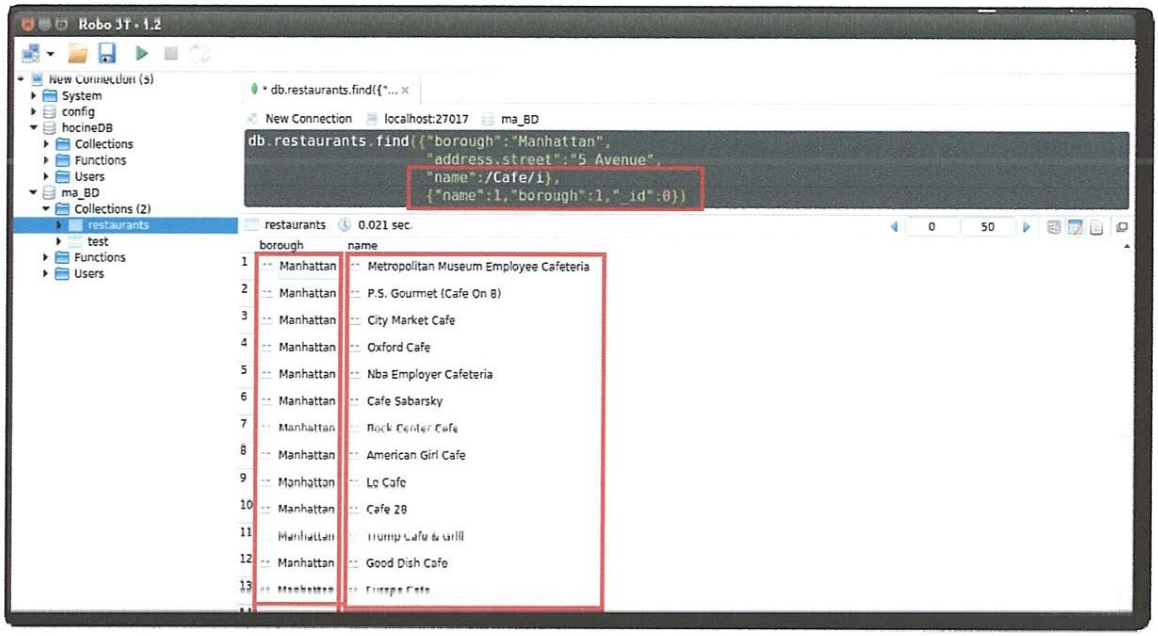


Figure 3.20 : *Résultat de requête de projection sur des champs de MongoDB*

La figure ci-dessus permet d'afficher tous les restaurants dont l'arrondissement est Manhattan, la rue est 5 Avenue et le nom de restaurant comprend le mot cafe. Par défaut,

MongoDB affiche les valeurs du champ _id, pour éviter d'afficher ce champ, il faut explicitement le mettre à 0.

On prend les mêmes conditions de cette requête pour compter le nombre documents (restaurants) satisfont ces conditions en utilisant la méthode count( ) (cf. Annexe C). On peut rechercher les valeurs distinctes d'un champ donné (cf. Annexe C).

Concernant les requêtes de modification de contenu de la base de données sont présentées comme suit :

- **Insertion :** insérer une données dans une collection de MongoDB se fait à travers la méthode save( ) ou parfois insert( ). Ex. insérer le restaurant LaMega.

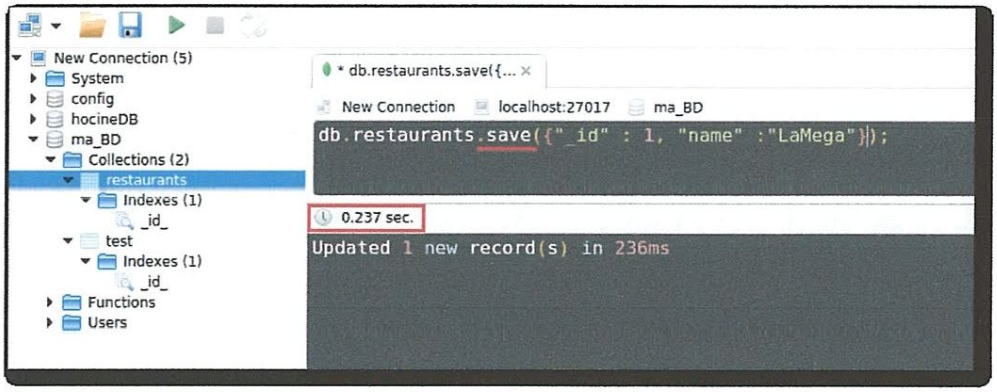


Figure 3.21 : *Insertion d'un document dans une collection de MongoDB*

Dans le cas où un document de la base ayant la même valeur de l'identifiant "_id" que le nouveau document. L'identifiant du document existant serait automatiquement écrasé par une nouvelle valeur.

- **Suppression :** la suppression d'un document se fait par la méthode remove. Ex. supprimer le restaurant dont l'identifiant soit 1.

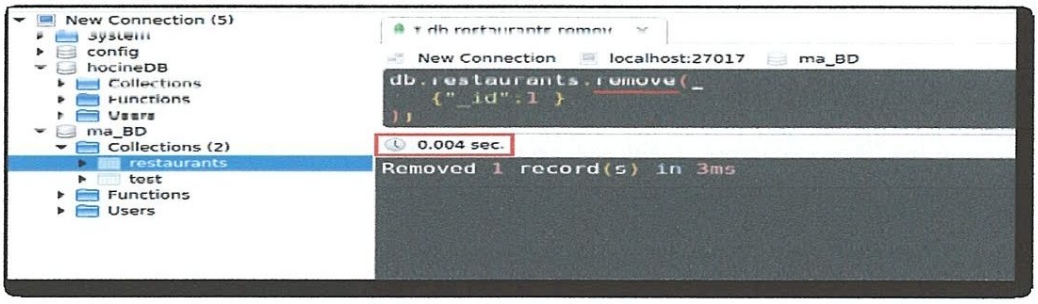


Figure 3.22 : *Suppression d'un document d'une collection MongoDB*

- **Mettre à jour :** la mise à jour d'une valeur d'un champ se fait par la méthode Update ( ) et la clause $set. L'exemple suivant consiste à modifier la valeur du champ "comment" du document dont l'identifiant est égale à 5ac36fce2b47344ca2535a0f.

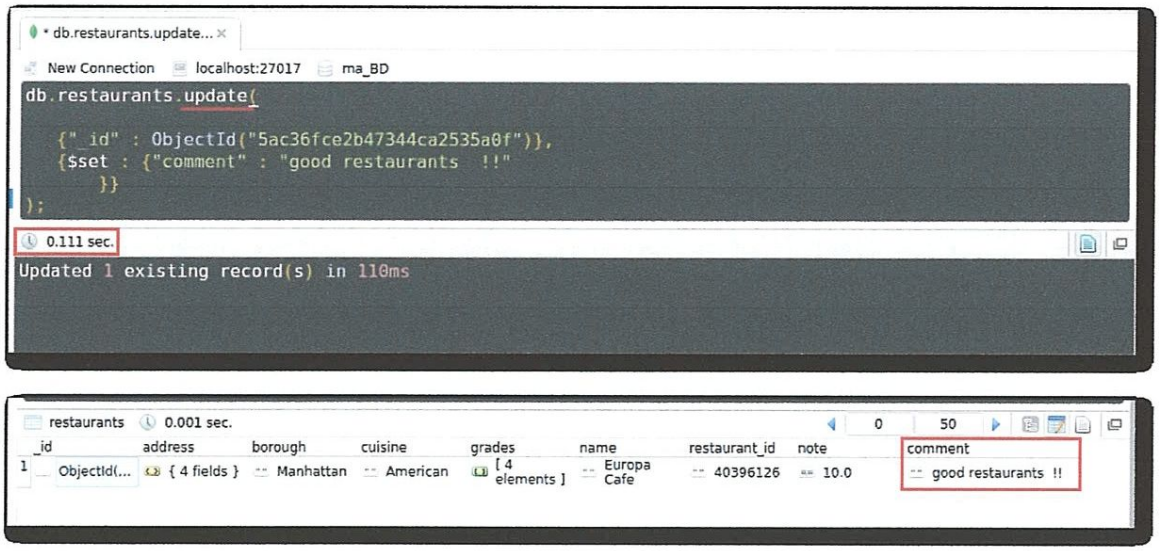


Figure 3.23: *Mise à jour d'un document d'une collection MongoDB*

D'après les manipulations faites sur les données NoSQL orientée documents, la première remarque qu'on peut constater que le SGBD MongoDB offre des fonctionnalités proches d'un SGBD relationnelles à savoir, langage de définition de données, langage de manipulation de données, indexation, trie, etc. Le MongoDB est basé sur l'utilisation de langage de requêtes à base de document pour définir et accéder aux données orientées documents. Par contre, une base de données mongodb n'impose pas de schéma stable de sorte que le nombre de champs du document peut différer d'un document à l'autre. Cette propriété permet d'éviter la complexité de jointure dans le fait que toutes les données peuvent être regroupées dans une seule collection.

Sur le plan d'évaluation de performance en termes de temps d'exécution, nous avons procédé l'étude suivante selon deux cas: étude de requêtes de recherche et étude de requête de modification de contenu de documents.

Dans le premier cas, nous avons réalisé 7 requêtes de recherche avec leur temps d'exécution.

N°	**Requête**	**Temps d'exécution (ms)**
Q1	Afficher tous les restaurants existants.	9
Q2	Afficher les scores inférieurs à 10 des restaurants de Manhattan.	16

Q3	Chercher les restaurants de Manhattan, situes dans la rue 5 Avenue et que leurs noms contiennent le mot cafe	20
Q4	La même requête Q3 sans que les résultats sont projetés sur le nom et l'arrondissement.	21
Q5	Compter le nombre de résultat de la requête Q3	18
Q6	Chercher les différents arrondissements	23
Q7	Chercher les restaurants de l'arrondissement est Manhattan	32

Table 3.5: *Temps d'exécutions de requêtes de recherche sur MongoDB*

La représentation graphique qui correspond à cette table est présentée dans la figure suivante.

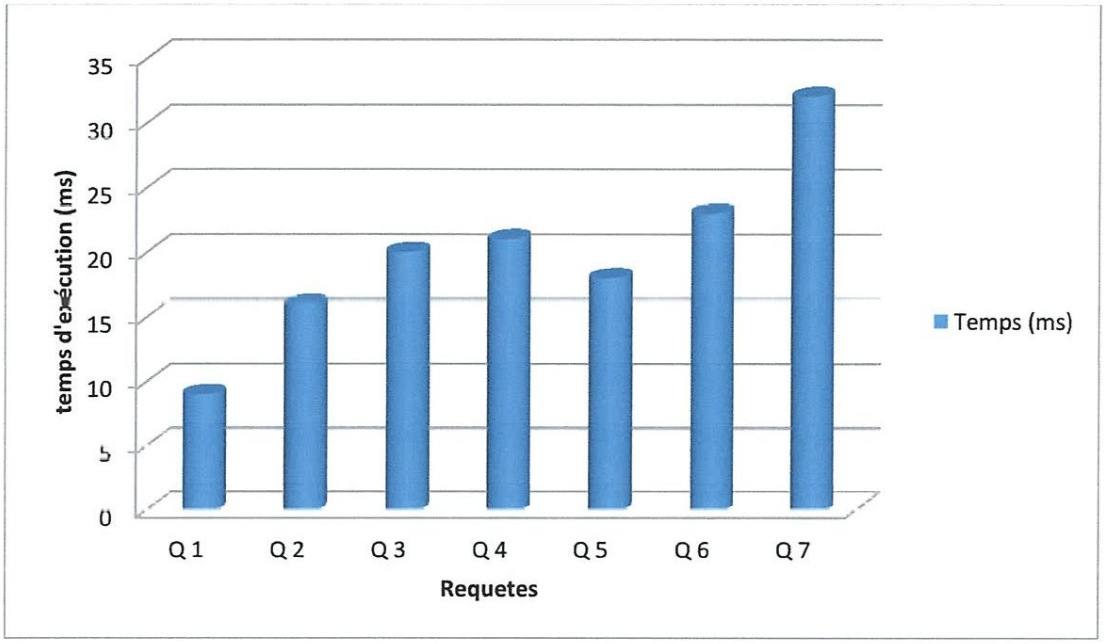


Figure 3.24 : *Histogramme de Temps d'exécutions de requêtes de recherche sur MongoDB*

D'après les résultats ci-dessus, nous remarquons que le temps d'exécution de requêtes ne dépend pas au nombre de champs à afficher (entre Q3 et Q4) par contre utiliser des conditions de recherche effectue une petite augmentation du temps comme dans le cas des requêtes Q1 (0 condition) et Q3 (3 conditions) (l'écart de 11ms) et entre Q3 et Q7 (1 condition) avec un écart de 12ms. Par ailleurs, le temps de calcul dans une base de données volumineuse a été optimisé par rapport les BDR. Le temps d'exécution de la requête Q5 (calcule le nombre de résultat de la requête Q3) est plus court que le temps d'exécution de requête d'affichage de données (Q3). Par conséquent, nous pouvons conclure que les SGBD NoSQL sont plus performants par rapport les SGBD classique en terme de temps de calcul et de la recherche par critères.

Dans le deuxième cas, nous étudions le temps d'exécution de 3 requêtes de modifications qui sont : l'insertion d'un document, suppression d'un document et la modification de valeur d'un document. Nous utilisons les mêmes requêtes de figures 21, 22 et 23. La table suivante résume le temps d'exécution de ces trois requêtes.

Type de requête	Temps d'exécution (ms)
Insertion (save)	236
Suppression (remove)	3
Modification (Update)	110

Table 3.6 : *Temps d'exécutions de requêtes de modification de données MongoDB*

A partir de cette table, nous pouvons conclure que le temps d'exécution s'augmente dans les requêtes d'insertion et de modification par rapport la requête de suppression. De ce fait, l'accès aux données NoSQL se fait rapidement contrairement à l'ajout d'un document ou la modification de contenu. Sachant que l'ajout d'un document dans une collection demande plus de 230 ms et la modification d'un document exige plus de 105 ms.

Finalement, la manipulation de données orientées documents par MongoDB est meilleur que dans un SGBD classique notamment dans les opérations de jointures (pas de jointures dans mongo), l'accès aux données et le calcul de données (opérations d'agrégation, de comparaison, etc).

7. Discussion de résultat

Nous avons effectué trois études de données NoSQL : étude de données sur Hadoop/MapReduce, étude de données orientées colonnes sur HBase via Hadoop et étude de données orientées documents sur MongoDB.

D'après les résultats obtenus de ces trois études, nous pouvons conclure que l'utilisation d'un nouveau type de données nommé NoSQL (données volumineuses) avec des SGBD dédiés aux stockage et traitement de ce type données donne un meilleur résultat en termes de temps d'exécution, de sécurité de données, de stockage de données, de tolérance aux pannes, etc.

Dans les SGBD classiques, un problème se pose quant au stockage et à l'analyse des données volumineuses et hétérogènes (des structures différentes). Nous voyons que le temps de lecture croît avec l'augmentation de la capacité de stockage des disques durs. Une solution existante qui consiste de paralléliser les traitements en stockant sur plusieurs unités de disques durs

avec la duplication des données (Bases de données réparties). Toutefois, cela soulève forcément le problème de fiabilité des disques durs qui engendre la panne matérielle.

Le framework Hadoop est un système distribué qui répond à ces problématiques. Il dispose d'un système de stockage distribué appelé HDFS et un système d'analyse des données appelé MapReduce dédié au traitement distribué sur des gros volumes de données.

Le framework Hadoop ne peut effectuer qu'un traitement distribué par blocs qui peuvent aller jusqu'à 1Go. Ses données volumineuses sont stockées dans les datanodes de HDFS et ne seront accessibles que de manière séquentielle. Distribuer les données sur plusieurs nœuds permet d'améliorer le temps d'exécution qui comprend le temps de Map, de splitting, de reduce, de shuffling et de combinaison de résultat. La tolérance aux pannes est une propriété très importante de Hadoop ainsi que le temps de reprise après la panne est très petit.

Par ailleurs, le framework Hadoop n'assure pas l'accès en temps réel de données stockées sur HDFS. Pour cela, nous avons étudié le SGBD HBase d'Hadoop qui stocke ses données sous forme de colonnes. Utiliser HBase permet d'améliorer le fonctionnement d'Hadoop en termes de stockage efficace de données. Le HBase dispose des méthodes pour manipuler ses données. D'après l'étude que nous avons réalisé, le HBase exige l'utilisation des machines très puissantes pour obtenir un meilleur résultat.

La dernière étude de données NoSQL est l'utilisation de meilleur SGBD NoSQL appelé « MongoDB » qui est très utile dans les applications de web et la gestion de documents en format JSON. Le MongoDB offre un langage de requêtes orienté documents proche de SQL pour la manipulation de documents. Le temps d'exécution de requêtes est plus court par rapport les SGBD relationnels notamment dans le cas de jointure. Dans une base de données MongoDB, on ne trouve pas la jointure, toutes les tables relationnelles sont représentées par une seule collection de documents ce qui diminue le temps d'exécution.

Finalement, nous avons planifié de faire une étude supplémentaire qui sert à faire une comparaison entre les quatre SGBD NoSQL mais nous avons rencontré des problèmes techniques (du matériel). Utiliser et traiter les données volumineuses NoSQL nécessite l'utilisation des machines physiques très puissantes.

8. Conclusion

Nous avons effectué une étude de données NoSQL sur trois types de stockage : le HDFS d'Hadoop, le HBase, et le MongoDB. Une discussion de différents résultats sont décrites. Les

résultats montrent que l'utilisation des machines physiques puissantes améliore la performance de ces systèmes.

CONCLUSION GENERALE

Conclusion générale

Le travail présenté dans ce mémoire se situe dans le contexte général de l'étude de données volumineuses "Big Data", et plus particulièrement les données NoSQL. Ces données nécessitent l'utilisation d'un système de gestion de données ayant la capacité de stockage de très grande quantité de données d'une part, et le traitement à grande échelle sur des clusters de serveurs, d'autre part. Hadoop Apache est le framework le plus performant et le plus utilisé pour le traitement distribués de données NoSQL via le paradigme MapReduce.

Dans ce travail, nous avons effectué trois grandes études sous Ubuntu 16.04 LTS:

- ***Etude de données sur le framework Hadoop :*** dans cette étude, la première étape sert à comprendre et apprendre le fonctionnement d'Hadoop puis le chargement de données dans son système de gestion distribuée de fichier HDFS. La deuxième étape vise à implémenter et exécuter les deux programmes d'Hadoop qui sont : Map et Reduce en utilisant le langage Java. Nous avons réalisé une étude de performance sur les données pour le décomptage de mots (Wordcount).
- ***Etude de données NoSQL sur le SGBD HBase :*** nous avons étudié les données NoSQL orientées colonnes qui sont proches de base de données relationnelles en utilisant le SGBD HBase d'Apache. Ce dernier est conçu pour s'exécuter sur le HDFS d'Hadoop.
- ***Etude de données NoSQL sur le SGBD MongoDB :*** le SGBD MongoDB est le meilleur SGBD NoSQL qui gère les données sous forme de documents au format BSON un dérivé de JSON. Le choix de SGBD est motivé par son adaptation aux applications web.

Dans l'étude de données sur Hadoop, nous avons réalisé un test de performance selon deux modes d'évaluation : single node (le maître est le même l'esclave) et multinodes (un maître et plusieurs esclaves). Dans le second mode, nous avons essayé d'utiliser plusieurs nœuds esclaves (machines virtuelles) sur une même machine physique mais malheureusement nous ne sommes pas arrivés à cause de contraintes techniques. La gestion distribuée et la réplication de données NoSQL nécessitent l'utilisation des machines très puissantes.

Le résultat de tests que nous avons effectué montre la nouveauté de données NoSQL par rapport les données classiques (données relationnelles, non structurés, documents, etc). Ainsi, l'utilisation de l'environnement distribué de données volumineuses sous Hadoop offre une

meilleure performance pour le traitement distribué de données via le paradigme MapReduce. Le choix d'un tel type de base de données NoSQL dépend le domaine d'application utilisé.

Toutefois, nous n'avons pas étudié le fonctionnement d'autres bases de données NoSQL. Par conséquent, nous désirons dans les travaux futurs d'utiliser une machine puissante pour effectuer des tests sur de vrai données volumineuses (données scientifiques du domaine de la biologie par exemple) sur plusieurs machines d'une part, et étudier d'autres types de données NoSQL à savoir les données orientées clé-valeur et données orientées graphe avec une étude comparative entre eux, d'autre part. L'utilisation de données NoSQL reste un domaine de recherche très à la mode, issues du terme big Data. Nous pensons d'utiliser ces données pour résoudre d'autres problèmes à titre d'exemple le mapping entre les différents types de bases de données NoSQL, le passage du modèle relationnel vers le NoSQL, etc.

Références Bibliographiques

Références Bibliographies

[1] E.F. Codd. « *A Relational Model of Data for Large Shared Data Banks* ». Communications of the ACM, vol. 13, n° 6, 1970, p. 377–387

[2] S. Sumathi, S. Esakkirajan. « Fundamentals of Relational Database Management Systems», Springer – 2007

[3] Thomas M. Connolly - Carolyn E. Begg. « Database systems: a practical approach to design, implementation, and management» ,Pearson Education – 2005

[4] Rudi Bruchez. « les base de donnes NoSQL et le Big Data» 2edition

[5] Big data et NoSQL, http://administration-systeme.blogspot.com/?view=classic, visité le 14/11/2017

[6] Sapia, C., Blaschka, M., Höfling, G., & Dinter, B. « Extending the E/R model for the multidimensional paradigm». In Advances in Database Technologies (pp. 105-116). Springer, Berlin, Heidelberg. 1999

[7] Mpinda, S. A. T., Maschietto, L. G., & Bungama, P. A. « From Relational Database to Column-Oriented NoSQL Database: Migration Process». International Journal of Engineering Research & Technology (IJERT), 4, 399-403. 2015

[8] http://davidmasclet.gisgraphy.com/post/2010/06/09/10-minutes-pour comprendre...NoSQL, visité le 01/12/2017

[9] Soufiane RITAL. « Not only SQL Introduction», cours pour TELECOM ParisTech.

[10] KOUEDI Emmanuel. « Approche de migration d'une base de données relationnelle vers une base de données NoSQL orientée colonne», Mémoire master informatique, UNIVERSITE DE YAOUNDE I, mai 2012 Institut TELECOM CNRS LTCI, paris,2012.

[11] Jeffrey Dean et Sanjay Ghemawat. « MapReduce: Simplified Data Processing on Large Clusters». 2004.

[12] N. Leavitt. Will. « NoSQL Databases Live Up to Their Promise? ». 2010

[13] Pierre-Jean Benghozi, Sylvain Bureau et Françoise Massit-Folléa « Étude L'Internet des objets». programme Vox Internet II.

[14] Internet des objets, https://fr.wikipedia.org/wiki/Internet_des_objets, visité le 16/12/2017

[15] CHEN, Hsinchun, CHIANG, Roger HL, et STOREY, Veda C. « Business intelligence and analytics: From big data to big impact». MIS quarterly, 2012, vol. 36, no 4.

[16] R. Bruchez. «Les bases de données NoSQL et le Big data». Livre 2ème édition, EYROLLES, Allemagne.

[17] Gardarin, G. «Bases de données», Livre, 5ème édition, editions eyrolles, 2003.

[18] http://www.lemagit.fr/conseil/Quel-SGBD-NoSQL-pour-vos-besoins-IT-Criteres-de-choix, visité le 19/12/2017

[19] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes et R. Gruber, Bigtable. « A Distributed Storage System for Structured Data», USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2006, Print ISBN 1-931971-47-1.

[20] Mr H.Hashem. « Modélisation intégratrice du traitement BigData», thèse de doctorat, université paris-Saclay.

[21] Aicha Aggoune. « Bases de données avancées ». Cours niveau master 2 informatique, département d'informatique, université de Guelma, Algérie.

[22] Aicha aggoune. « Traitement de l'Hétérogénéité Sémantique pour l'Exploration des Sources de Données Multimédias ». Thèse de doctorat, université de constantine2, Algérie, 2017.

[23] Sun, B., Luo, W. S., Du, L. B. et Lu, Q. « Storage Model Based on Oracle InterMedia for Surveillance Video», Applied Mechanics and Materials, Vol. 644, 2014, pp. 3318-3321.

[24] Laney, Douglas. «3D Data Management: Controlling Data Volume, Velocity and Variety» . Gartner. Retrieved 6 February 2001.

[25] The Big Data Long Tail. Blog fait par Jason Bloomber, 17 Janvier 2013. http://www.devx.com/blog/the-big-data-long-tail.html

[26] Dave Evans. «L'Internet des objets Comment l'évolution actuelle d'Internet transforme-t-elle le monde ? ». CISCO. Livre blanc, 2011.

[27] Grand dictionnaire terminologique, Office québécois de la langue française. Visité le 30/12/2017

[28] Brewer , Eric A. «Towards Robust Distributed Systems». Portland, Oregon, July 2000. –Keynote at the ACM Symposium on Principles of Distributed Computing (PODC) on 2000-07-19. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[29] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. « Bigtable: A Distributed Storage System for Structured Data». 2006

[30] A.G. PIAZZA. « NoSQL: Etat de l'art et benchmark», Haute École de Gestion de Genève (HEG-GE). 2013

[31] Adriaan De Jonge. «Essential App Engine: Building High-Performance Java Apps with Google App Engine», Addison-Wesley Professional, 2011

[32] Daniel A. Keim, Jörn Kohlhammer, Geoffrey Ellis, Florian Mansmann. «Mastering the Information Age - Solving Problems with Visual Analytics». 2010

[33] Nick Rozanski, Eoin Woods. «Software systems architecture: Working with Stakeholders using viewpoints and perspectives». 2012

[34] Lee, K. H., Lee, Y. J., Choi, H., Chung, Y. D., & Moon, B. «Parallel data processing with MapReduce: a survey ». AcM sIGMoD Record, 40(4), 11-20. 2012

[35] Jiang, D., Ooi, B. C., Shi, L., & Wu, S. «The performance of mapreduce: An in-depth study». Proceedings of the VLDB Endowment, 3(1-2), 472-483. 2010

[36] http://juvenal-chokogoue.developpez.com/tutoriels/hadoop-fonctionnement/. Visité le 12/01/2018

[37] CUTTING, Doug et CAFARELLA, M. «About hadoop». 2005.

[38] https://www.lebigdata.fr/hadoop. Visité le 12/01/2018

[39] http://www.tomsitpro.com/articles/hadoop-2-vs-1,2-718.html. Visité le 26/02/2018

[40] THUSOO, Ashish, SARMA, Joydeep Sen, JAIN, Namit, et al. «Hive-a petabyte scale data warehouse using hadoop». In : Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010. p. 996-1005.

[41] Benjamin Renaut. « Intoduction a Hadoop et MapReduce », Cours informatique, département d'informatique, université de Nice Sophia Antipolis, France.

[42] Pierre Nerzic, « Outils Hadoop pour le BigData», 2018, p. 85-102

[43] Dhruba Borthakur , https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html visité le 23/02/2018

[44] Fonctionnement de Hadoop, http://www.opentuto.com/fonctionnement-de-hadoop/, Visité le 01/03/2018

[45] LeframeworkApacheHadoop, http://igm.univmlv.fr/~dr/XPOSE2012/Le20framework % 20Apache%20 Hadoop/architecture.html, Visité le 01/03/2018

[46] Introduction to MongoDB, https://docs.mongodb.com/manual/introduction/. Visité le 10/03/2018

[47] Classement des SGBD, https://db-engines.com/en/ranking. Visité le 15/01/2018

[48] MongoDB et le sharding, https://www.supinfo.com/articles/single/4761-mongodb-sharding, Visité le 18/05/2018

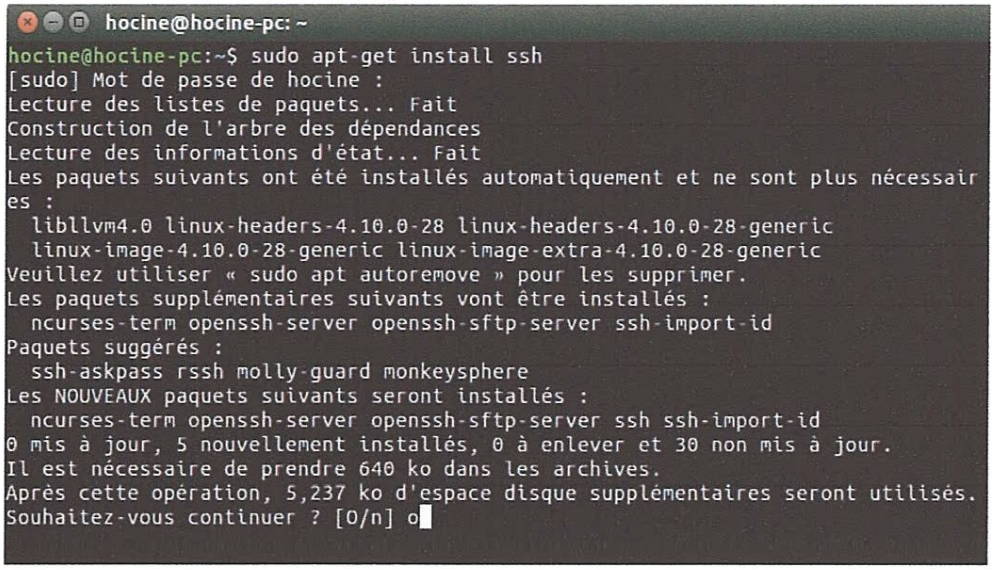
[49] Réplication, https://docs.mongodb.com/manual/replication/, Visité le 18/05/2018

[50] Passez à l'échelle, https://openclassrooms.com/courses/maitrisez-les-bases-de-donnees-nosql/pourquoi-pas-le-oriente-documents, Visité le 19/05/2018

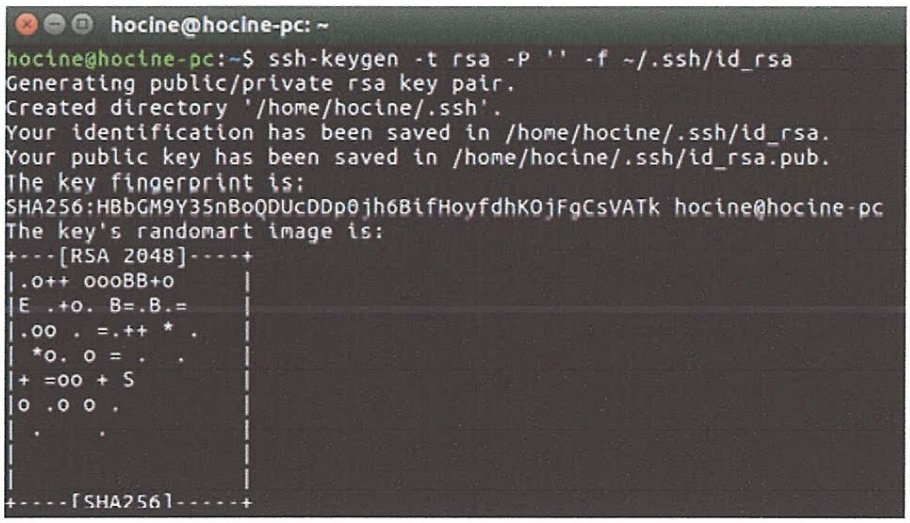
ANNEXES

Annexe A

La figure suivante présente le déroulement de l'installation de SSH.



La génération de clé RSA de SSH. Le symbole -P " " indique un mot de passe vide.



Installation de Resync

Code java de la fonction Map pour le job Wordcount

```
17-  public static class TokenizerMapper
18         extends Mapper<Object, Text, Text, IntWritable>{
19
20      private final static IntWritable one = new IntWritable(1);
21      private Text word = new Text();
22
23-     public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
24        StringTokenizer itr = new StringTokenizer(value.toString());
25        while (itr.hasMoreTokens()) {
26          word.set(itr.nextToken());
27          context.write(word, one);
28        }
29      }
30    }
```

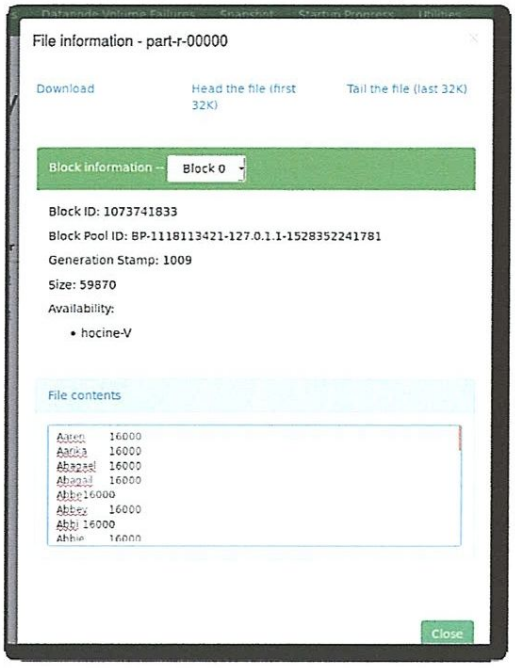
Code java de la fonction Reduce pour le job Wordcount

```
31
32-  public static class IntSumReducer
33         extends Reducer<Text,IntWritable,Text,IntWritable> {
34     private IntWritable result = new IntWritable();
35
36-    public void reduce(Text key, Iterable<IntWritable> values,
37                       Context context
38                       ) throws IOException, InterruptedException {
39       int sum = 0;
40       for (IntWritable val : values) {
41         sum += val.get();
42       }
43       result.set(sum);
44       context.write(key, result);
45     }
46   }
```

La classe WordCount contient le job que nous devrons l'exécuter sur Hadoop.

```
47
48-   public static void main(String[] args) throws Exception {
49      Configuration conf = new Configuration();
50      Job job = Job.getInstance(conf, "word count");
51      job.setJarByClass(WordCount.class);
52      job.setMapperClass(TokenizerMapper.class);
53      job.setCombinerClass(IntSumReducer.class);
54      job.setReducerClass(IntSumReducer.class);
55      job.setOutputKeyClass(Text.class);
56      job.setOutputValueClass(IntWritable.class);
57      FileInputFormat.addInputPath(job, new Path(args[0]));
58      FileOutputFormat.setOutputPath(job, new Path(args[1]));
59      System.exit(job.waitForCompletion(true) ? 0 : 1);
60    }
```

La Visualisation de contenu de fichier de sortie



Démarrage de cluster Hadoop sur trois nœuds :

```
hocine@hocine-V: ~/hadoop
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hocine-V/127.0.1.1
************************************************************/
hocine@hocine-V:~/hadoop$ sbin/start-dfs.sh
Starting namenodes on [node-master]
node-master: starting namenode, logging to /home/hocine/hadoop/logs/hadoop-hocine-namenode-hocine-V.out
node-master: starting datanode, logging to /home/hocine/hadoop/logs/hadoop-hocine-datanode-hocine-V.out
node1: starting datanode, logging to /home/hocine/hadoop/logs/hadoop-hocine-datanode-hocine-acer.out
node2: starting datanode, logging to /home/hocine/hadoop/logs/hadoop-hocine-datanode-hocine-HP.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: secondarynamenode running as process 9761. Stop it first.
hocine@hocine-V:~/hadoop$ sbin/start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/hocine/hadoop/logs/yarn-hocine-resourcemanager-hocine-V.out
node-master: starting nodemanager, logging to /home/hocine/hadoop/logs/yarn-hocine-nodemanager-hocine-V.out
node2: starting nodemanager, logging to /home/hocine/hadoop/logs/yarn-hocine-nodemanager-hocine-HP.out
node1: starting nodemanager, logging to /home/hocine/hadoop/logs/yarn-hocine-nodemanager-hocine-acer.out
hocine@hocine-V:~/hadoop$ jps
9761 SecondaryNameNode
11316 NodeManager
10695 NameNode
10859 DataNode
11628 Jps
11180 ResourceManager
hocine@hocine-V:~/hadoop$
```

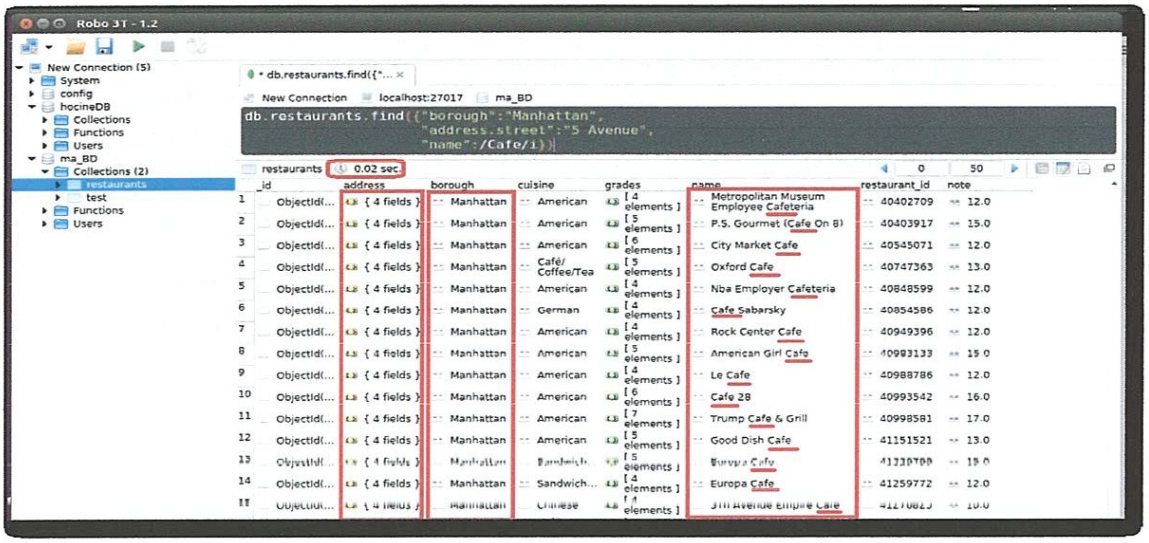
NB : La commande JPS permet d'afficher les informations sur le nombre de nœuds possible.

Résultat de l'exécution de Job wordcount sur trois nœuds via l'invite de commande d'Ubuntu.

```
18/06/10 01:51:49 INFO mapreduce.Job: Job job_1528591241243_0001 running in uber mode : false
18/06/10 01:51:49 INFO mapreduce.Job:  map 0% reduce 0%
18/06/10 01:52:06 INFO mapreduce.Job:  map 33% reduce 0%
18/06/10 01:52:12 INFO mapreduce.Job:  map 53% reduce 0%
18/06/10 01:52:18 INFO mapreduce.Job:  map 65% reduce 0%
18/06/10 01:52:19 INFO mapreduce.Job:  map 100% reduce 0%
18/06/10 01:52:25 INFO mapreduce.Job:  map 100% reduce 100%
18/06/10 01:52:26 INFO mapreduce.Job: Job job_1528591241243_0001 completed successfully
18/06/10 01:52:26 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=518352
                FILE: Number of bytes written=986107
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=142425708
                HDFS: Number of bytes written=59870
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=28045
                Total time spent by all reduces in occupied slots (ms)=3129
                Total time spent by all map tasks (ms)=28045
                Total time spent by all reduce tasks (ms)=3129
                Total vcore-milliseconds taken by all map tasks=28045
                Total vcore-milliseconds taken by all reduce tasks=3129
                Total megabyte-milliseconds taken by all map tasks=28718080
                Total megabyte-milliseconds taken by all reduce tasks=3204096
        Map-Reduce Framework
                Map input records=20101601
                Map output records=20100000
                Map output bytes=222804000
                Map output materialized bytes=64794
                Input split bytes=107
                Combine input records=20134608
                Map output records=20100000
                Map output bytes=222804000
                Map output materialized bytes=64794
                Input split bytes=107
                Combine input records=20134608
                Combine output records=20552
                Reduce input groups=4944
                Reduce shuffle bytes=64794
                Reduce input records=4944
                Reduce output records=4944
                Spilled Records=44496
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=339
                CPU time spent (ms)=33700
                Physical memory (bytes) snapshot=487915520
                Virtual memory (bytes) snapshot=3959726080
                Total committed heap usage (bytes)=303038464
```

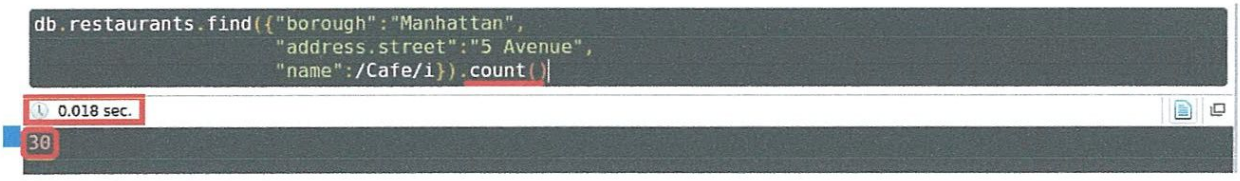
Résultat de l'exécution de Job WordCount sur 2 nœuds

Exemple 1 : chercher les restaurants de Manhattan qui se trouve dans la rue 5° Avenue et le nom du restaurant comprend le mot "cafe".

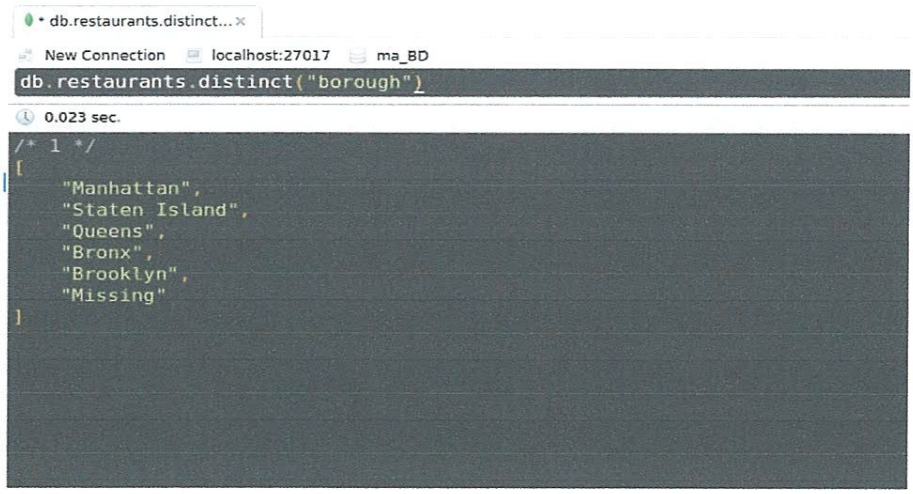


Exemple 2 : chercher le nom et le grade des restaurants de Manhattan dont le score du grade est inférieur à 10.

Exemple 3 : compter le nombre résultat de la requête de l'exemple 1.



Exemple 4 : chercher les différents arrondissements.


db.restaurants.distinct...
New Connection
localhost:27017
ma_BD
db.restaurants.distinct("borough")
0.023 sec.
/* 1 */
[
"Manhattan",
"Staten Island",
"Queens",
"Bronx",
"Brooklyn",
"Missing"
]