Table des matières

Chapitre II: Hadoop et ses composants

Chapitre III : Mise en œuvre, Test et Evaluation

Liste des Figures

Liste des Tableaux

Introduction générale

Contexte

Nous sommes confrontés actuellement à une explosion de données structurées ou non structurées produites massivement par les différentes sources de données numériques. D'une part les applications qui génèrent des données issues des logs, des réseaux de capteurs, des traces de GPS, etc., et d'autre part, les utilisateurs produisent beaucoup de données telles que des photographies, des vidéos et des musiques. Selon IBM, chaque heure 2.5 trillions d'octets de données sont générées. Selon les prévisions faites, d'ici 2020 cette croissance sera supérieure à 40 Zettaoctets, alors qu'un Zettaoctet de données numériques, seulement, ont été générées de 1940(début de l'informatique) à 2010. Beaucoup de concepts «inséparables» dominent actuellement le marché de l'IT : «Cloud Computing», «Big Data», «NoSQL» ou «MapReduce».

Problématique

De rudes contraintes opposent les différents chercheurs dans le domaine, quant au stockage et à l'analyse de ces masses de données. Les prévisions de taux de croissance des volumes de données traitées dépassent les limites des technologies traditionnelles à savoir les bases de données relationnelles ou les Datawarehouses. On parle de pétaoctet (billiard d'octets 10^{15}), voir d'exaoctet (trilliard d'octets 10^{18}) et encore le Zettaoctet(10^{21}) ou le Yottaoctet(10^{24}).

Notre problématique est comment prendre en charge l'accroissement rapide des volumes de données géographiquement éloignées et comment gérer cette puissante montée en charge. Quel sont les technologies et les modèles de programmation proposées pour pallier ces différents problèmes engendrés par ce déluge de données ?

Solution

Cette révolution scientifique qui envahisse le monde de l'information et l'Internet a imposé aux différents chercheurs depuis quelques années, de nouveaux défis et les a poussé à concevoir de nouvelles technologies pour contenir, traiter ces volumes énormes de données. Plusieurs modèles de programmation parallèle et systèmes de gestion de fichiers distribués ont émergé, principalement, le Framework Hadoop se place comme la solution la plus répandue dans le marché informatique.

Hadoop est un environnement d'exécution distribuée, performant et scalable, il propose un système de stockage distribué via son système de fichier HDFS (Hadoop Distributed File System) et un système d'analyse et traitement de données basé le modèle de

programmation MapReduce pour réaliser des traitements parallèles et distribués sur des gros volumes de données.

Notre projet de fin d'étude a pour but d'étudier les méthodes et les technologies du Big Data, Nous nous intéresserons particulièrement aux technologies Hadoop avec ses composantes HDFS et MapReduce.

La partie applicative, va consister à un déploiement d'Hadoop avec tous ses composants dans un environnement distribué, configuration d'un cluster de plusieurs machines, exécution de plusieurs jobs MapReduce sur des fichiers HDFS et rapprochement des différentes statistiques des opérations effectuées. L'installation d'Hadoop a été effectuée à partir de la distribution Cloudera CDH4 sur des machines virtuelles avec Linux centos 6.6 comme système d'exploitation.

Objectif

Mettre en œuvre une architecture complète d'un cloud computing et montrer l'influence du parallélisme sur les performances globales du système, d'un côté. D'un autre côté, la familiarisation avec l'environnement Hadoop et la maitrise de nouvelles technologies, MapReduce, HDFS et NoSQL sont les objectifs principaux ciblés par ce projet de fin d'études de Master en informatique.

Organisation du mémoire

Nous commencerons dans cette thèse par une introduction générale décrivant le contexte de travail, la problématique, la solution adoptée et l'objectif de ce sujet.

La deuxième partie s'articule autour de trois chapitres qui se présentent comme suit :

Dans le premier chapitre, on a introduit les concepts des Big Data et du Cloud Computing, on a également présenté le NoSQL, l'un des principales technologies du Big Data.

Le deuxième chapitre est consacré à la présentation du Framework Hadoop, la description de ses principaux composants HDFS et sa propre version de MapReduce.

Dans le troisième chapitre, on a abordé la partie technique qui consiste à déployer, configurer et tester l'ensemble des éléments logiciels nécessaires pour la mise en œuvre de la solution.

Nous achèverons avec une conclusion, dans laquelle on va synthétiser tous les résultats obtenus et les enseignements acquis durant la réalisation de ce projet.

I.1 Big Data

I.1.1 Introduction

De grandes quantités d'informations sont mises en ligne sur le web par des milliers d'entreprises, d'organisations et d'individus, la charge ainsi que le volume de données à gérer ont crû de façon exponentielle, pour cela les sociétés ont recouru au Datawarehouse ou entrepôt de données pour l'analyse et le stockage de données.

Généralement le Datawarehouse est centralisé dans un serveur connecté à une baie de stockage, cette solution est difficilement scalable (ajout de puissance à la demande) en plus du fait qu'elle ne gère que les données structurées dans des SGBD.

Pour faire face à l'explosion du volume des données, on parle actuellement de pétaoctet (billiard d'octets) voir de zettaoctet (trilliard d'octets) et aussi face à la grande variété des données (image, texte, web, etc.) un nouveau domaine technologique a vu le jour : le Big Data inventé par les géants du web, au premier rang comme Yahoo, Google et Facebook, qui ont été les tous premiers à déployer ce type de technologie.

Ce concept apporte une architecture distribuée et scalable pour le traitement et le stockage de données. Ce nouveau paradigme a pour principal objectif l'amélioration des performances et l'augmentation de la vitesse d'exécution des requêtes et des traitements.

I.1.2 Définition de Big Data

Le terme de Big Data a été évoquée la première fois par le cabinet d'études Gartner en 2008 mais la naissance de ce terme effective remonte à 2001 et a été évoquée par le cabinet Meta Group.

Il fait référence à l'explosion du volume des données (de par leur nombre, la vitesse à laquelle elles sont produites et leur variété) et aux nouvelles solutions proposées pour gérer cette volumétrie tant par la capacité à stocker et explorer, et récemment par la capacité à analyser et exploiter ces données dans une approche temps réel. [12]

Big data, littérairement les grosses données, est une expression anglophone utilisée pour désigner des ensembles de données qui deviennent tellement volumineux qu'ils en deviennent difficiles à travailler avec des outils classiques de gestion de base de données. Il s'agit donc d'un ensemble de technologies, d'architecture, d'outils et de procédures permettant à une organisation très rapidement de capter, traiter et analyser de larges quantités et contenus hétérogènes et changeants, et d'en extraire les informations pertinentes à un coût accessible. [16]

I.1.3 Caractéristiques du Big Data

Le Big Data (en français "Grandes données") regroupe une famille d'outils qui répondent à une triple problématiques : C'est la règle dite des **3V**. [10]

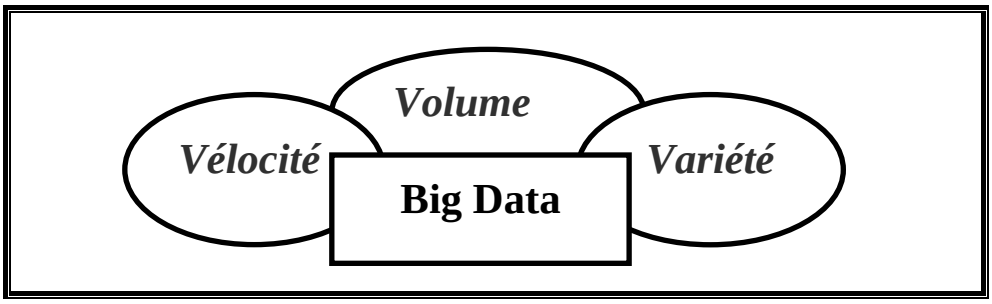

Figure I.1: Le Big Data, les 3 V [16]

I.1.3.1 Volume

Le Big Data est associé à un volume de données vertigineux, se situant actuellement entre quelques dizaines de téraoctets et plusieurs péta-octets en un seul jeu de données. Les entreprises et tous les secteurs d'activités confondus, devront trouver des moyens pour gérer le volume de données en constante augmentation qui est créé quotidiennement. Les catalogues de plus de 10 millions de produits sont devenus la règle plutôt que l'exception.

Voici quelques chiffres pour illustrer ce phénomène :

- 90% des données actuelles ont été créées dans les deux dernières années seulement ;
- Twitter comme exemple, génère 7 To de données chaque jour.

I.1.3.2 Vitesse

La vitesse décrit la fréquence à laquelle les données sont générées, capturées et partagées. Les entreprises doivent appréhender la vitesse non seulement en termes de création de données, mais aussi sur le plan de leur traitement, de leur analyse et de leur restitution à l'utilisateur en respectant les exigences des applications en temps réel.

I.1.3.3 Variété

La croissance de la variété des données est la conséquence des nouvelles données multi structurelles et de l'expansion des types de données provenant de différentes sources hétérogènes. Aujourd'hui, on trouve des capteurs d'informations aussi bien dans les appareils électroménagers, les trains, les automobiles ou les avions, qui produisent des informations très variées.

Ces nouvelles données dites **non-structurées** sont variées :

- Des photos ;
- Des mails (avec l'analyse sémantique de leur contenu) ;
- Les données issues des réseaux sociaux (commentaires et avis des internautes sur Facebook ou Twitter par exemple) ;

Ces trois caractéristiques illustrées par les trois « V », sont les principes définissant le Big Data. Avant tout, il s'agit d'un changement d'orientation sur l'utilisation de la donnée. En somme, le point clé du Big Data est de donner un sens à ces grosses données et pour cela, il faut les analyser.

I.1.4 Processus de chargement et de collecte de données dans Big Data

La couche responsable du chargement de données dans Big Data, devrait être capable de gérer d'énorme volume de données, avec une haute vitesse, et une grande variété de données. Cette couche devrait avoir la capacité de valider, nettoyer, transformer, réduire (compression), et d'intégrer les données dans la grande pile de données en vue de son traitement. La Figure illustre le processus et les composants qui doivent être présent dans la couche de chargement de données. [2][5]

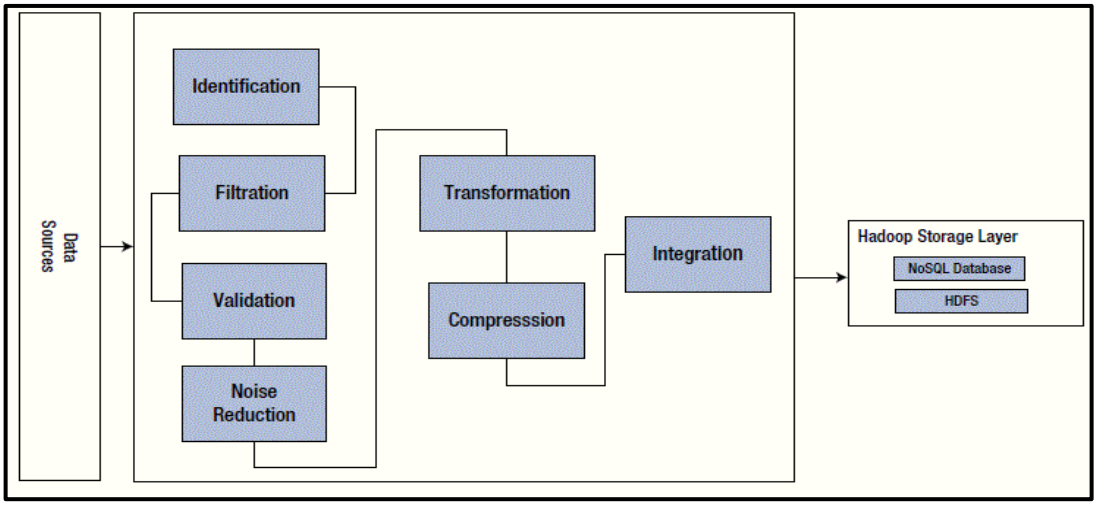

Figure I.2 : Couche de chargement des données dans le Big Data [2]

La couche de chargement de données de Big Data collecte les informations pertinentes finales, sans bruit, et les charge dans la couche de stockage de Big Data (HDFS ou NoSQL base).Elle doit inclure les composants suivants :

- **Identification** des différents formats de données connues, par défaut Big Data cible les données non structurées ;
- **Filtration et sélection** de l'information entrante pertinente pour l'entreprise ;

- **Validation** et analyse des données en permanence ;
- **Réduction** de bruit implique le nettoyage des données en supprimant le bruit ;
- **La transformation** peut entrainer le découpage, la convergence, la normalisation ou la synthèse des données ;
- **Compression** consiste à réduire la taille des données, mais sans perdre de la pertinence des données ;
- **Intégration** consiste à intégrer l'ensemble des données dans le stockage de données de Big Data (HDFS ou NoSQL base).

I.1.5 Différence entre BI (Business intelligence) et Big Data

La méthodologie BI traditionnel fonctionne sur le principe de regrouper toutes les données de l'entreprise dans un serveur central (Datawarehouse ou entrepôt de données). Les données sont généralement analyse en mode déconnecté.

Les données sont généralement structurées en SGBDR avec très peu de données non structurées. [2][5]

Une solution Big Data, est différente d'une BI traditionnel dans les aspects suivants :

- Les données sont conservées dans un système de fichiers distribué et scalable plutôt que sur un serveur central ;
- Les données sont de formats différents, à la fois structurées ainsi que non structurées ;
- Les données sont analysées en temps réel ;
- La technologie Big Data s'appuie sur un traitement massivement parallèle (concept MPP).

I.1.6 Architecture Big Data

On distingue principalement les couches suivantes :

- **Couche matériel** (infrastructure Layer) :peut-être des serveurs virtuels VMware, ou des serveurs lame blade ;
- **Couche stockage** (Storage layer) : les données seront stockées soit dans une base NoSQL, ou bien directement dans le système de fichier distribué ou les Datawarehouse ;
- **Couche management et traitement** : on trouve dans cette couche les outils de traitement et analyse des données comme MapReduce ou Pig ; [2]

- **Couche visualisation** : pour la visualisation du résultat du traitement.

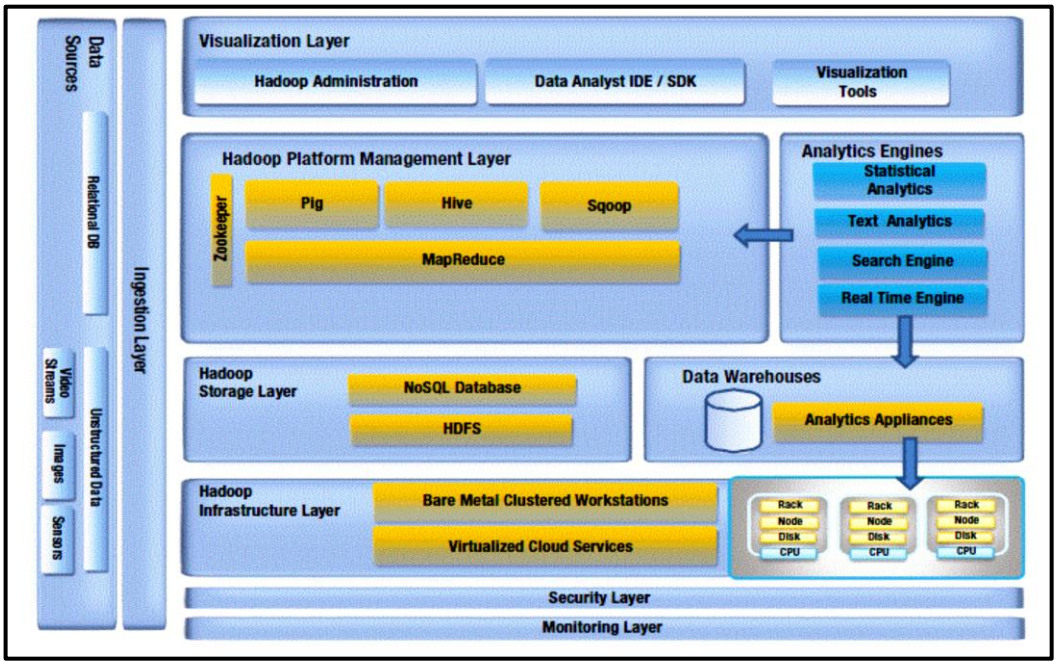

Figure I.3: Architecture de Big Data [2]

I.1.7 Avantages de l'architecture Big Data

Plusieurs avantages peuvent être associés à une architecture Big Data, nous pouvons citer par exemple :

- **Evolutivité (scalabilité)** : Quelle est la taille que devra avoir votre infrastructure ? Combien d'espace disque est nécessaire aujourd'hui et à l'avenir ? le concept Big Data nous permet de s'affranchir de ces questions, car il apporte une architecture scalable.
- **Performance** : Grâce au traitement parallèle des données et à son système de fichiers distribué, le concept Big Data est hautement performant en diminuant la latence des requêtes.
- **Coût faible** : Le principal outil Big Data à savoir Hadoop est en Open Source, en plus on n'aura plus besoin de centraliser les données dans des baies de stockage souvent excessivement chère, avec le Big Data et grâce au système de fichiers distribués les disques internes des serveurs suffiront.
- **Disponibilité** : On a plus besoin des RAID disques, souvent coûteux. L'architecture Big Data apporte ses propres mécanismes de haute disponibilité.

I.1.8 L'analyse : le point clé du Big Data

Le Big Data répond à de nombreux objectifs précis parmi lesquels on trouve l'**extraction** d'informations utiles des données stockées, l'**analyse** de ces données, la **restitution** efficace des résultats d'analyse ou encore, l'accroissement de **l'interactivité** entre utilisateurs et données.

La combinaison de ce déluge d'informations et d'algorithmes logiciels intelligents ouvre la voie à de nouvelles opportunités de business. Prenons, par exemple, Google et Facebook qui sont des « entreprises Big Data », mais aussi IBM ou JDA Software.

L'analyse est le point clé de l'utilisation du Big Data. Elle permet de mieux connaître sa clientèle, d'optimiser son marketing, de détecter et prévenir des fraudes, d'analyser son image sur les réseaux sociaux et d'optimiser ses processus métiers. [10]

I.1.9 Quelques domaines d'utilisation du Big Data

Avant de conclure, citons rapidement quelques domaines d'utilisation du Big Data. Le Big Data trouve sa place dans de nombreux domaines :

Dans la première catégorie, on retrouve des secteurs qui manipulent quotidiennement des volumes de données très important, avec des problématiques de vitesse associées. On y trouve :

- Les **Banques** : la sanctuarisation de données anciennes dues à des contraintes réglementaires ;
- La **Télécommunication** : l'analyse de l'état du réseau en temps réel ;
- Les **Médias Numériques** : le ciblage publicitaire et l'analyse de sites web ;
- Les **Marchés Financier** : l'analyse des transactions pour la gestion des risques et la gestion des fraudes, ainsi que pour l'analyse des clients.

La deuxième catégorie de secteur est plus hétérogène, les besoins, mais aussi l'utilisation qui est faite du Big Data, peuvent être très différents. On y trouve :

- **Les Services Publics** : l'analyse des compteurs (gaz, électricité, etc.) et la gestion des équipements ;
- **Le Marketing** : le ciblage publicitaire et l'analyse de tendance ;
- **La Santé** : l'analyse des dossiers médicaux et l'analyse génomique. [10]

I.1.10 Big Data et Datawarehouse

Les entrepôts de données sont traditionnellement des supports des données structurées et ont été étroitement lié aux systèmes opérationnels et transactionnels de l'entreprise

(SGBDR). Ces systèmes qui sont soigneusement construits sont maintenant au milieu d'importants changements après l'émergence de Big Data. [3][5]

Datawarehouse est une base de données (données structurées) regroupant une partie ou l'ensemble des données fonctionnelles d'une entreprise. Il entre dans le cadre de l'informatique décisionnelle ; son but est de fournir un ensemble de données servant de référence unique, utilisées pour la prise de décisions dans l'entreprise par les baies de statistiques et de rapports réalisés via des outils de reporting. D'un point de vue technique, il sert surtout à 'délester' les bases de données opérationnelles des requêtes pouvant nuire à leurs performances. [17]

Les organisations continueront inévitablement à utiliser des entrepôts de données pour gérer le type de données structurées et opérationnelles qui caractérise les systèmes relationnels(SGBDR). Ces entrepôts seront toujours fournis aux analystes avec la capacité pour analyser les données clés, les tendances, et ainsi de suite.

Cependant, avec l'avènement du Big Data, le défi pour les entrepôts de données est de réfléchir à une approche complémentaire avec le Big Data, on pourrait concevoir un modèle hybride. Dans ce modèle les restes de données optimisées opérationnelles très structurées seront stockées et analysées dans l'entrepôt de données, tandis que les données qui sont fortement distribuées et non structurées seront contrôlées par Big Data (Hadoop ou NoSQL). [3][5]

La tendance serait de stocker la grande masse de données non structurées dans une vaste gamme de serveurs Big Data (Hadoop/MapReduce) pour tirer profit de la scalabilité et la rapidité d'analyse de Big Data, ensuite à l'aide d'outils, ces données seront déplacées dans le modèle relationnel de sorte qu'elles peuvent être interrogées avec le langage SQL traditionnel (SGBDR et Datawarehouse).

On peut donc interfacer Big Data avec le Datawarehouse(DW), effectivement les données non structurées provenant de différentes sources peuvent être regroupées dans un HDFS avant d'être transformées et chargées à l'aide d'outils spécifiques dans le Datawarehouse et les outils traditionnels de BI. [2][5]

Comme les DW traditionnels ne gèrent pas les données non structurées, Big Data peut servir comme un moyen de stockage et d'analyse des données non structurées qui seront chargées dans les DW.

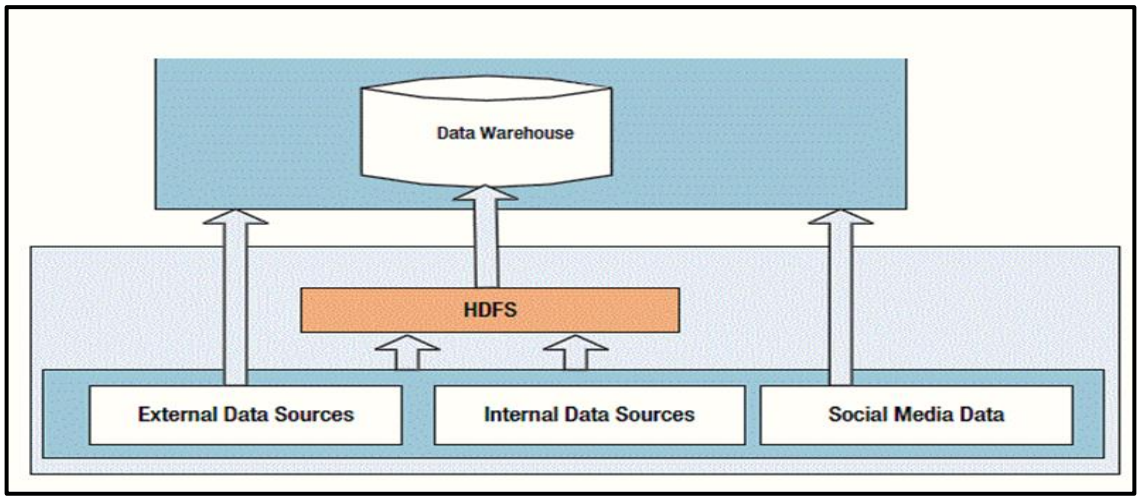

Figure I.4 : Lien entre Big Data et DW [2]

I.1.11 Big Data et les ETL (extraction, transformation et chargement)

Certains outils ETL traditionnels comme Talent commence à s'adapter avec le monde Big Data. Les outils ETL sont utilisés pour transformer les données dans le format requis par l'entrepôt de données(Datawarehouse). La transformation est effectivement faite dans un endroit intermédiaire avant que les données ne seront chargées dans l'entrepôt de données.

Pour le Big Data des outils ETL comme Informatica ont été utilisés pour permettre une solution d'ingestion rapide et flexible des données non structurées (supérieure à 150 Go/jour). [2]

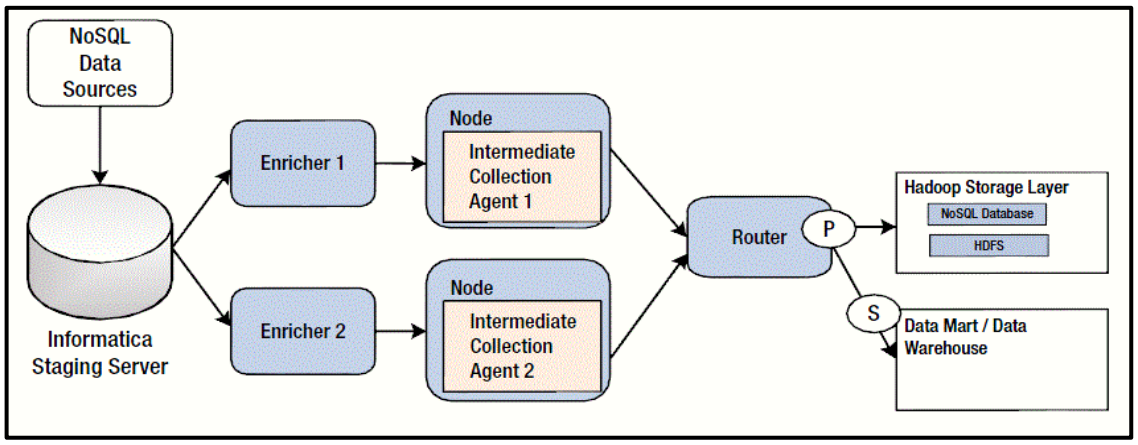

Figure I.5 : Utilisation d'ETL Informatica pour Big Data [2]

I.1.12 Les principales technologies de Big Data

Elles sont nombreuses. Pour optimiser les temps de traitement sur des bases de données géantes, plusieurs solutions peuvent entrer en jeu :

- **Des bases de données NoSQL** (comme MongoDB, Cassandra ou Redis) qui implémentent des systèmes de stockage considérés comme plus performants que

le traditionnel SQL pour l'analyse de données en masse (orienté clé/valeur, document, colonne ou graphe).

- **Des infrastructures de serveurs pour distribuer les traitements** sur des dizaines, centaines, voire milliers de nœuds. C'est ce qu'on appelle le traitement massivement parallèle. Le Framework Hadoop est sans doute le plus connu d'entre eux. Il combine le système de fichiers distribué HDFS, la base NoSQL HBase et l'algorithme MapReduce. (Hadoop, HDFS et MapReduce seront présentés dans le chapitre II).

- **Le stockage des données en mémoire :** On parle de traitement in-memory pour évoquer les traitements qui sont effectués dans la mémoire vive de l'équipement informatique, plutôt que sur des serveurs externes.

  L'avantage du traitement in-memory est celui de la vitesse puisque les données sont immédiatement accessibles. En revanche, ces données ne sont pas stockées sur le long terme, ce qui peut poser des problèmes d'historisation. [11]

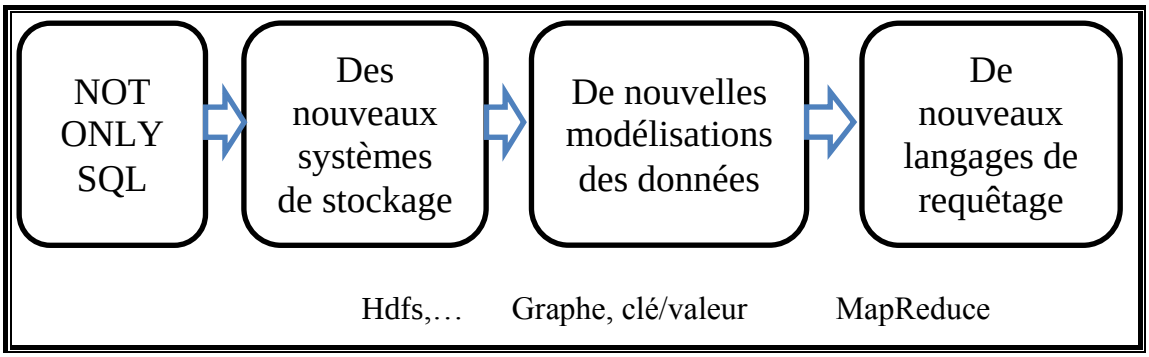

Figure I.6: Les solutions de stockage

I.1.13 Bases données NoSQL

Les bases de données NoSQL (No-SQL ou Not Only SQL) sont un sujet très à la mode en ce moment. Le terme NoSQL désigne une catégorie de systèmes de gestion de base de données destinés à manipuler des bases de données volumineuses pour des sites de grande audience. Les bases données NoSQL sont scalables, elles permettent de traiter les données d'une façon distribuée. Parmi les avantages du NoSQL on trouve :

- Leurs performances ne s'écroulent jamais quel que soit le volume traité. Leur temps de réponse est proportionnel au volume ;

- Elles se migrent facilement. En effet, contrairement aux SGBDR classiques, il n'est pas nécessaire de procéder à une interruption de service pour effectuer le déploiement d'une fonctionnalité impactant les modèles des données ;
- Elles sont facilement scalable. A titre d'exemple, le plus gros cluster de NoSQL fait 400 To, tandis qu'Oracle sait traiter jusqu'à une vingtaine de Téraoctet (pour des temps de réponse raisonnable).

I.1.13.1Caractéristiques NoSQL

- Gros volume de données ;
- Réplication scalable et distribution ;
  – Des centaines de machines voire des milliers;
  – Distribuées partout dans le monde.
- Des requêtes qui exigent une réponse rapide;
- Asynchronicité des insertions et updates ;
- Acidité non respectée dans la plupart du temps ;
- Des performances en lectures/écritures ;
  - Centaines de milliers de lectures/secondes ;
  - Centaines de milliers d'écritures/secondes ;

I.1.13.2 Les types des bases NoSQL

Il en existe 4 types distincts qui s'utilisent différemment et qui se prêtent mieux selon le type données que l'on souhaite y stocker. [7]

- **Clé-Valeur**

Les BD NoSQL fonctionnant sur le principe Clé-Valeur sont les plus basiques que l'on peut trouver.

- Elles fonctionnent comme un grand tableau associatif et retourne une valeur dont elle ne connaît pas la structure ;
- Leur modèle peut être assimilé à une table de hachage (hashmap) distribuée **;**
- Les données sont simplement représentées par un couple clé/valeur ;
- La valeur peut être une simple chaîne de caractères, ou un objet sérialisé.

- **Document**

Elles sont basées sur le modèle « clé-valeur » mais la valeur est un document en format semi-structuré hiérarchique de type JSON ou XML (possible aussi de stocker

n'importe quel objet, via une sérialisation). Elles stockent une collection de "documents"

- **Colonnes**

Les données sont stockées par colonne, non par ligne, on peut facilement ajouter des colonnes aux tables, par contre l'insertion d'une ligne est plus coûteuse quand les données d'une colonne se ressemblent, on peut facilement compresser la colonne.

C'est un modèle proche d'une table dans un SGBDR mais ici le nombre de colonnes:

- est **dynamique ;**
- peut **varier d'un enregistrement à un autre,** ce qui évite de retrouver des colonnes ayant des valeurs NULL.

- **Graphe**

Elles permettent la modélisation, le stockage et la manipulation de données complexes liées par des relations non-triviales ou variables

• modèle de représentation des données basé sur la théorie des graphes

• s'appuie sur les notions de nœuds, de relations et de propriétés qui leur sont rattachées.

I.1.13.3 Les principales bases de données NoSQL

- **MongoDB**

La plus populaire des bases NoSQL documentaires est écrite en C et n'utilise pas de machine virtuelle JAVA. Cette base est idéale pour débuter car elle est à la fois polyvalente et simple. Aussi à l'aise pour le stockage massif de données que pour le développement rapide orienté Web. Elle possède également une documentation de premier ordre. [18]

- **Cassandra**

Cassandra est le projet open source qui découlent de la technologie de stockage Facebook. À l'origine, elle a été écrite spécifiquement pour répondre à la croissance explosive de cette entreprise. Elle est assez complexe à configurer, mais elle permet d'adresser toutes les situations ou la performance et le traitement de la volumétrie est critique. Cassandra est une base de données en colonnes écrite en JAVA. [18]

- **HBase**

Hbase est inspirée des publications de Google sur BigTable. Comme BigTable, elle est une base de données oriente colonne. Basée sur une architecture maitre/esclave, les

bases de données HBase sont capables de gérer d'énormes quantités d'informations (plusieurs milliards de lignes par table). [5]

I.2 Le Cloud Computing

Depuis sa création, la technologie de l'Internet se développe d'une manière exponentielle. Actuellement, une nouvelle « tendance » est dominante, il s'agit du Cloud Computing. Cette technologie, s'appuie sur le WEB 2.0, offre des occasions aux sociétés de réduire les coûts d'exploitation des logiciels par leurs utilisations directement en ligne. [1]

Dans la section suivante, nous allons présenter les notions fondamentales du Cloud Computing, ses enjeux, ses évolutions et son utilité.

I.2.1. Définition

Le Cloud Computing, « littéralement l'informatique dans les nuages », est un concept qui consiste à « rendre accessible et exploitable des données et des applications à travers un réseau. Ce terme désigne à la fois les applications en tant que services sur Internet et le matériel et logiciels qui permettent de fournir ces services ». [1]

I.2.2. Les différents services

Le concept « Cloud Computing » est utilisé pour désigner un ensemble de services, on distingue trois catégories de services fournis :

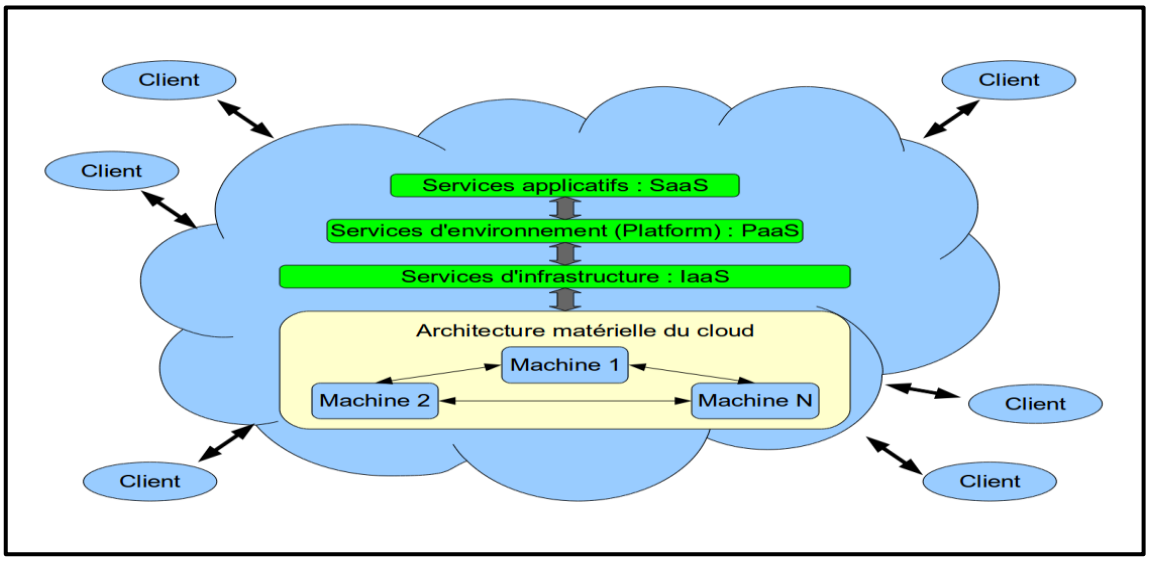

Figure I.7: Services Cloud Computing [1]

- **Infrastructure as a service (IaaS) :** il s'agit de la mise à disposition, à la demande, de ressources d'infrastructures dont la plus grande partie est localisée à distance dans des Data Center.

- **Platform as a service (PaaS) :** désigne les services visant à proposer un environnement complet permettant de développer et déployer des applications.
- **Software as a service (SaaS) :** il s'agit des plateformes du nuage, regroupant principalement les serveurs mutualisés et leurs systèmes d'exploitation. En plus de pouvoir délivrer des logiciels.

I.2.3. Les formes de déploiement du Cloud Computing

Nous distinguons trois formes de Cloud Computing: Le Cloud publique, également le premier apparu, le Cloud privé et le Cloud hybride qui est en fait la combinaison des deux premiers.

- **Le Cloud public**

Le principe est d'héberger des applications, en général des applications Web, sur un environnement partagé avec un nombre illimité d'utilisateurs. La mise en place de ce type de Cloud est gérée par des entreprises tierces (exemple Amazon, Google, etc.). Les fournisseurs du Cloud publique, les plus connus sont Google et Amazon. Ce modèle est caractérisé par :

- Mise en place de lourds investissements pour le fournisseur de services.
- Offre un maximum de flexibilité.
- N’est pas sécurisé.

- **Le Cloud privé**

C'est un environnement déployé au sein d'une entreprise en utilisant les infrastructures internes de l’entreprise, et en utilisant des technologies telles que la virtualisation. Les ressources sont détenues et contrôlées par le département informatique de l’entreprise. Eucalyptus, OpenNebula et OpenStack sont des exemples de solution pour la mise en place du Cloud privé.

- **Le Cloud hybride**

En général, on entend par Cloud hybride la cohabitation et la communication entre un Cloud privé et un Cloud publique dans une organisation partageant des données et des applications.

I.3 Conclusion

Nous avons abordé dans ce premier chapitre les principes des Big Data, ces caractéristiques, son fonctionnement ainsi que les différents domaines dans lesquels elles sont utilisées.

On a aussi recensé les différents modèles de bases de données NoSQL qui existent, actuellement, dans le marché en accentuant sur les solutions les plus populaires.

Nous avons présenté brièvement en fin du chapitre les différents concepts liés au Cloud Computing, à savoir les types de services Cloud connus jusqu'à présent et ses différents modes de déploiement.

II.1 Introduction

Le Big data regroupe plusieurs nouvelles technologies et d'outils pour répondre à une triple problématique : un **V**olume de données important à traiter, une grande **V**ariété d'informations (structurées ou non structurées), et un certain niveau de **V**élocité à atteindre. Pour répondre à ces besoins, il s'avère que l'écosystème Hadoop serait la solution open source par excellence.

Apache Hadoop (High-availability distributed object-oriented platform) est un système distribué qui répond à ces problématiques. D'une part, il propose un système de fichier distribué HDFS (Hadoop Distributed File System) pour assurer le stockage et l'intégrité des données en dupliquant plusieurs copies d'un même bloc à travers des dizaines, des centaines, voire des milliers de machines différentes, ce qui amène un cluster Hadoop à être configuré sans requérir un système RAID. D'autre part, Hadoop fournit un système d'analyse de données appelé MapReduce pour réaliser des traitements sur des gros volumes de données grâce à sa répartition efficace du travail sur différents nœuds de calcul.

II.2 Présentation d'Hadoop

Hadoop est un Framework Java open source d'Apache pour réaliser des traitements sur des volumes de données massifs, de l'ordre de plusieurs pétaoctets (soit plusieurs milliers de To).

Hadoop a été conçu par Doug Cutting en 2004, également à l'origine du moteur Open Source Nutch. Doug Cutting cherchait une solution pour accroître la taille de l'index de son moteur. Il eut l'idée de créer un Framework de gestion de fichiers distribués. Yahoo! en est devenu ensuite le principal contributeur, le portail utilisait notamment l'infrastructure pour supporter son moteur de recherche historique. Comptant plus de 10 000 clusters Linux en 2008, il s'agissait d'une des premières architectures Hadoop digne de ce nom. Crée spécialement pour les gros volumes. Facebook pour l'analyse des logs, Google pour l'analyse des requêtes, etc...

Il est caractérisé par :

- **Robuste** : si un nœud de calcul tombe, ses tâches sont automatiquement réparties sur d'autres nœuds. Les blocs de données sont également répliqués;
- **Coût** : il optimise les coûts via une meilleure utilisation des ressources présentées;

- **Souple** : car il répond à la caractéristique de variété des données en étant capable de traiter différents types de données;
- **Virtualisation** : ne plus se reposer directement sur l'infrastructure physique (baie de stockage coûteuse), mais choisir la virtualisation de ses clusters Hadoop.

Trois principales distributions Hadoop sont aujourd'hui disponibles : Cloudera, Hortonworks, MapR.

Nous allons présenter deux concepts fondamentaux d'Hadoop : Sa propre version de l'algorithme MapReduce à savoir **Hadoop MapReduce** et son système de fichiers distribué **HDFS**.

II.2.1 Le système de fichier distribué d'Hadoop HDFS

Hadoop utilise un système de fichiers virtuel qui lui est propre : le HDFS (Hadoop Distributed File System). HDFS est un système de fichier distribué, extensible et portable inspiré par le Google File System (GFS).

Il a été conçu pour stocker de très gros volumes de données sur un grand nombre de machine équipées de disques durs banalisés, il permet de l'abstraction de l'architecture physique de stockage, afin de manipuler un système de fichier distribué comme s'il s'agissait d'un disque dur unique. [19]

Toutefois, HDFS se démarque d'un système de fichiers classique pour les principales raisons suivantes [15]:

- HDFS n'est pas dépendant du noyau du système d'exploitation. Il assure une portabilité et peut être déployé sur différents systèmes d'exploitation. Un de ses inconvénients est de devoir solliciter une application externe pour monter une unité de disque HDFS ;
- HDFS est un système distribué sur un système classique, la taille du disque est généralement considérée comme la limite globale d'utilisation. Dans HDFS, chaque nœud d'un cluster correspond à un sous-ensemble du volume global des données du cluster. Pour augmenter ce volume global, il suffira d'ajouter de nouveaux nœuds. On retrouvera également dans HDFS, un service central appelé NameNode qui aura la tâche de gérer les métadonnées ;
- HDFS utilise des tailles de blocs largement supérieures à ceux des systèmes classiques. Par défaut, la taille est fixée à 64 Mo. Il est toutefois possible de monter à 128 Mo, 256 Mo, 512 Mo voire 1 Go. Alors que sur des systèmes classiques, la taille est généralement de 4 Ko, l'intérêt de fournir des tailles plus

grandes permettant de réduire le temps d'accès à un bloc. Notez que si la taille du fichier est inférieure à la taille d'un bloc, le fichier n'occupera pas la taille totale de ce bloc ;

- HDFS fournit un système de réplication des blocs dont le nombre de réplications est configurable. Pendant la phase d'écriture, chaque bloc correspondant au fichier est répliqué sur plusieurs nœuds. Pour la phase de lecture, si un bloc est indisponible sur un nœud, des copies de ce bloc seront disponibles sur d'autres nœuds.

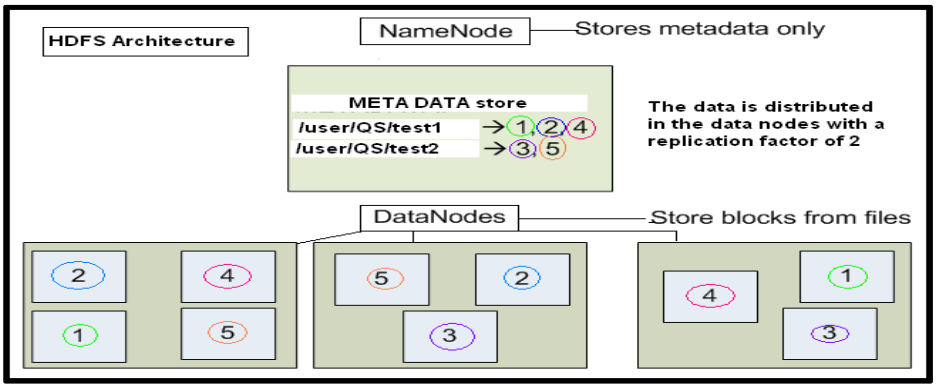

Figure II.1 :L'architecture d'HDFS [21]

II.2.1.1 Les composantes d'HDFS

HDFS définit deux types de nœuds :

- **Le nœud principal ou NameNode**

Il se caractérise par :

- Responsable de la distribution et de la réplication des blocs ;
- Serveur d'informations du Hdfs pour le client Hdfs ;
- Stocke et gère les métadonnées ;
- Comporte la liste des blocs pour chaque fichier (dans le cas de lecture) ;
- Contient la liste des DataNodes pour chaque bloc (dans le cas de l'écriture) ;
- Tenir les attributs des fichiers (ex : nom, date de création, facteur de réplication) ;
- Logs toute métadonnée et toute transaction sur un support persistant ;
- Lectures/écritures ;
- Créations/suppressions ;
- Démarre à partir d'une image d'HDFS (fsimage).

- **Le nœud de données ou DataNode**

Il se caractérise par :

- Stocke des blocs de données dans le système de fichier local ;
- Maintenir des métadonnées sur les blocs possédés (ex : CRC) ;
- Serveur de bloc de données et de métadonnées pour le client hdfs ;
- Heartbeat avec le Namenode : Heartbeat est système permettant sous Linux la mise en clusters de plusieurs serveurs pour effectuer entre eux un processus de tolérance de panne. Le processus Heartbeat se chargera de passer un message-aller vers le NameNode indiquant : son identité, sa capacité totale, son espace utilisé, son espace restant.

- **Secondary NameNode**

Le NameNode dans l'architecture Hadoop est un point unique de défaillance. Si ce service est arrêté, il n'y a pas moyen de pouvoir extraire les blocs d'un fichier donné. Pour résoudre ce problème, un NameNode secondaire appelé Secondary NameNode a été mis en place dans l'architecture Hadoop. Son rôle consiste à :

- Télécharger régulièrement les logs sur le NameNode ;
- Crée une nouvelle image en fusionnant les logs avec l'image HDFS ;
- Renvoie la nouvelle image au NameNode. [9]

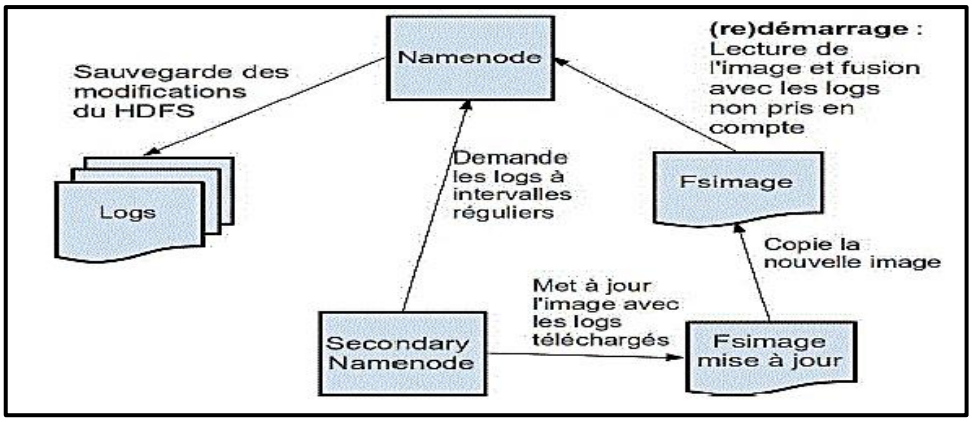

Figure II.2 : Fonctionnement du Secondary NameNode [9]

II.2.1.2 HDFS et tolérance aux fautes

Pour éviter un crash du DataNode la solution serait :

- Plus de Heartbeat (détection par le NameNode) ;
- Plus de réplication distribuée (robustesse des données).

Dans le cas d'un crash du NameNode (voir Figure II.2**)** :

- Sauvegarde des logs de transaction sur un support stable ;
- Redémarrage sur la dernière image du hdfs.

II.2.1.3 Lecture d'un fichier HDFS

Pour lire un fichier au sein de HDFS, il faut suivre les étapes suivantes :

Etape 1 : Le client indique au NameNode qu'il souhaite lire le fichier data.txt

Etape 2 : Le NameNode lui indiquera la taille de fichier (nombre de blocs) ainsi que les différents Data Node hébergeant les n blocs.

Etape 3 : Le client récupère chacun des blocs à un des DataNodes.

Etape 4 : En cas d'erreur/non réponse d'un des DataNode, il passe au suivant dans la liste fournie par le NameNode. **[9]**

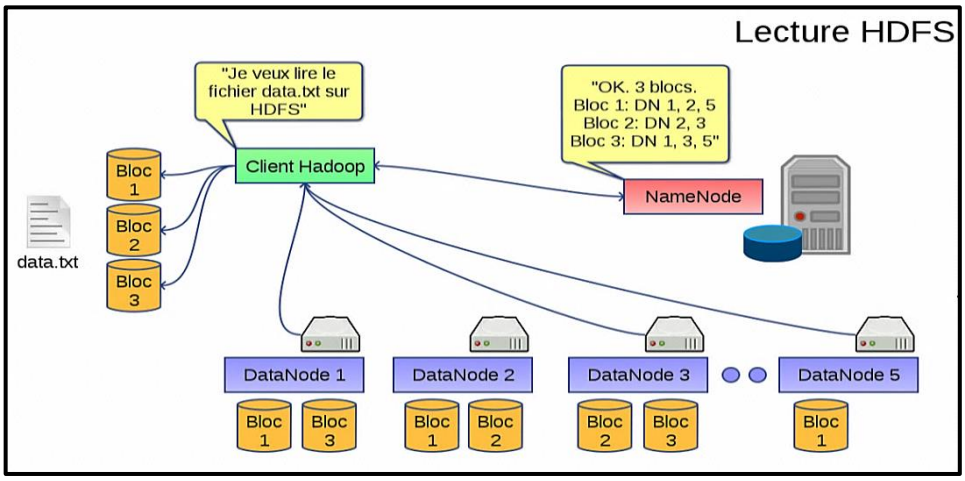

Figure II.3 : Lecture d'un fichier HDFS. [6]

II.2.1.4 Ecriture dans un fichier ou volume HDFS

Pour écrire un fichier au sein d'HDFS:

Etape 1 : On va utiliser la commande principale de gestion de Hadoop: Hadoop, avec l'option fs. Admettons qu'on souhaite stocker le fichier data.txt sur HDFS.

Etape 2 : Le programme va diviser le fichier en blocs de 64KB (ou autre, selon la configuration) – supposons qu'on ait ici 3 blocs.

Etape 3 : Le NameNode lui indique les DataNodes à contacter.

Etape 4 : Le client contacte directement le DataNode concerné et lui demande de stocker le bloc.

Etape 5 : les DataNodes s'occuperont – en informant le NameNode – de répliquer les données entre eux pour éviter toute perte de données.

Etape 6 : Le cycle se répète pour le bloc suivant.

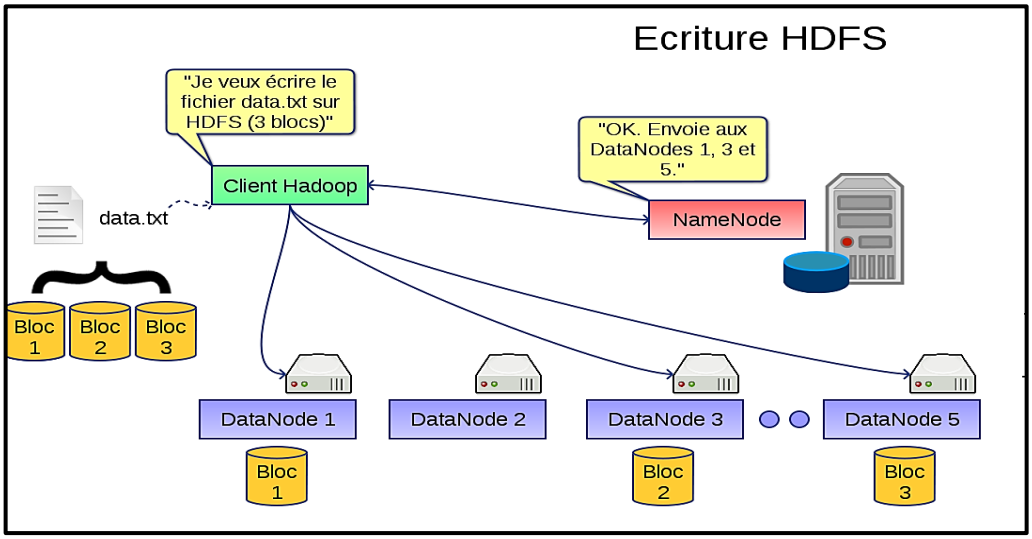

Figure II.4 : Processus d'écriture dans un volume ou fichier HDFS. [3]

II.2.1.5 HDFS : Stratégies de placement de bloc

Stratégie actuelle

- Une réplication sur un nœud aléatoire dans le rack ;
- Deuxième réplication sur un rack distant ;
- Troisième réplication sur le même rack distant ;
- Réplication additionnel placé aléatoirement ;

  Le client lit le plus proche réplica. [8]

II.2.2 MapReduce

MapReduce est un paradigme (modèle) de programmation parallèle proposé par Google. Il est principalement utilisé pour le traitement distribué sur de gros volumes de données aux seins d'un cluster de nœuds. Il est conçu pour la scalabilité et la tolérance aux pannes.

Le modèle de programmation fournit un cadre à un développeur afin d'écrire une fonction Map et une fonction Reduce. Tout l'intérêt de ce modèle de programmation est de simplifier la vie du développeur. Ainsi, ce développeur n'a pas à se soucier du travail de parallélisation et de distribution du travail. MapReduce permet au développeur de ne s'intéresser qu'à la partie algorithmique. [15]

Un programme MapReduce peut se résumer à deux fonctions Map () et Reduce ()

- La première, **MAP**, va transformer les données d'entrée en une série de couples clef /valeur. Elle va regrouper les données en les associant à des clefs, choisies de telle

sorte que les couples clef/valeur aient un sens par rapport au problème à résoudre. Par ailleurs, cette opération doit être parallélisable: on doit pouvoir découper les données d'entrée en plusieurs fragments, et faire exécuter l'opération MAP à chaque machine du cluster sur un fragment distinct. La fonction Map s'écrit de la manière suivante :

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

- La seconde, **REDUCE**, va appliquer un traitement à toutes les valeurs de chacune des clefs distinctes produite par l'opération MAP. Au terme de l'opération REDUCE, on aura un résultat pour chacune des clefs distinctes. Ici, on attribuera à chacune des machines du cluster une des clefs uniques produites par MAP, en lui donnant la liste des valeurs associées à la clef. Chacune des machines effectuera alors l'opération REDUCE pour cette clef. La fonction Reduce s'écrit de la manière suivante : Reduce (clé2, List (valeur2)) → List (valeur2).

L'exemple classique est celui du **WordCount** qui permet de compter le nombre d'occurrences d'un mot dans un fichier. En entrée l'algorithme reçoit un fichier texte qui contient les mots suivants **voiture la le elle de elle la se la maison voiture**

Dans notre exemple, la clé d'entrée correspond au numéro de ligne dans le fichier et tous les mots sont comptabilisés à l'exception du mot « se ».

Le résultat de la fonction Map est donné ci-dessous.

(voiture, 1) /(la,1) /(le,1) / (elle,1) / (de,1) / (elle,1) / (la,1) / (la,1) / (maison,1) / (voiture,1)

Avant de présenter la fonction Reduce, deux opérations intermédiaires doivent être exécutées pour préparer la valeur de son paramètre d'entrée. La première opération appelée **shuffle** permet de grouper les valeurs dont la clé est commune. La seconde opération appelée **sort** permet de trier par clé. A la différence des fonctions Map et Reduce, shuffle et sort sont des fonctions fournies par le Framework Hadoop, donc, il n'a pas à les implémenter.

Ainsi, après l'exécution des fonctions shuffle et sort le résultat de l'exemple est le suivant :

(de, [1]) / (elle, [1,1]) / (la, [1, 1,1]) / (le, [1]) / (maison, [1]) / (voiture, [1,1]

Suite à l'appel de la fonction Reduce, le résultat de l'exemple est le suivant :

(de, 1) / (elle, 2) / (la, 3) / (le, 1) / (maison, 1) / (voiture, 2)

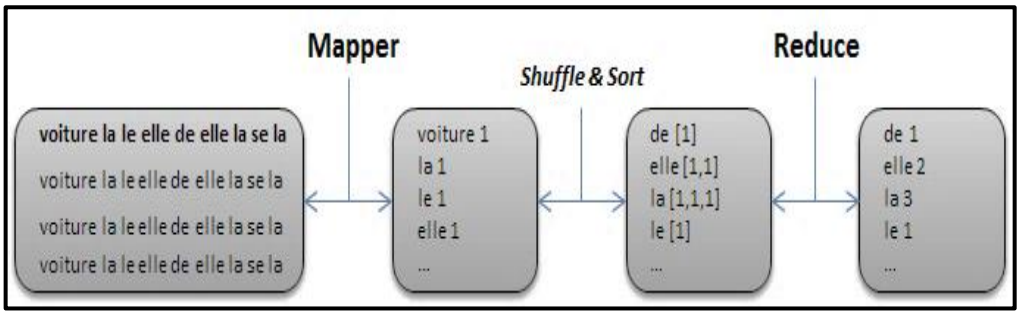

Figure II.5: Exemple d'un programme MapReduce (WordCount) [15]

Le code JAVA de la fonction Map () et Reduce () de cette exemple est en **Annexe A**.

II.2.2.1 MapReduce dans Hadoop

Il existe Deux versions notables du MapReduce :

- Version 0.x et 1.x : architecture purement maître-esclave ;
- Version 2.x YARN : architecture maître-esclave à deux niveaux

(Stable depuis oct. 2013). [9]

II.2.2.2 Architecture du Framework MapReduce (purement maître-esclave)

Le MapReduce possède une architecture maître-esclave

- Le maître MapReduce : le JobTracker ;
- Les esclaves MapReduce : les TaskTracker

.

1. **Le JobTracker**
    - Gère l'ensemble des ressources du système ;
    - Reçoit les jobs des clients ;
    - Ordonnance les différentes tâches des jobs soumis ;
    - Assigne les tâches aux TaskTrackers ;
    - Réaffecte les tâches défaillantes ;
    - Maintient des informations sur l'état d'avancement des jobs. [9]
2. **Un TaskTracker**
    - Exécute les tâches données par le Jobtracker ;
    - Exécution des tâches dans une autre JVM (Child) ;
    - A une capacité en termes de nombres de tâches qu'il peut exécuter ;
    - Heartbeat avec le JobTracker. [9]

II.2.2.3 Fonctionnement du Framework MapReduce

L'exécution d'un job MapReduce, concerne quatre entités indépendantes :

- Le client, qui soumet le job MapReduce;
- Le JobTracker, qui coordonne l'exécution et le partage du job en sous tâches. Le Jobtracker est une application Java dont la classe principale est JobTracker;
- Les TaskTrackers, qui gèrent les tâches que le JobTracker lui a confiées. Les Tasktrackers sont des applications Java dont la classe principale est TaskTracker;
- Le système de fichiers distribué, qui est utilisée pour le partage de fichiers de travail entre les autres entités (tasktrackers). [4][5]

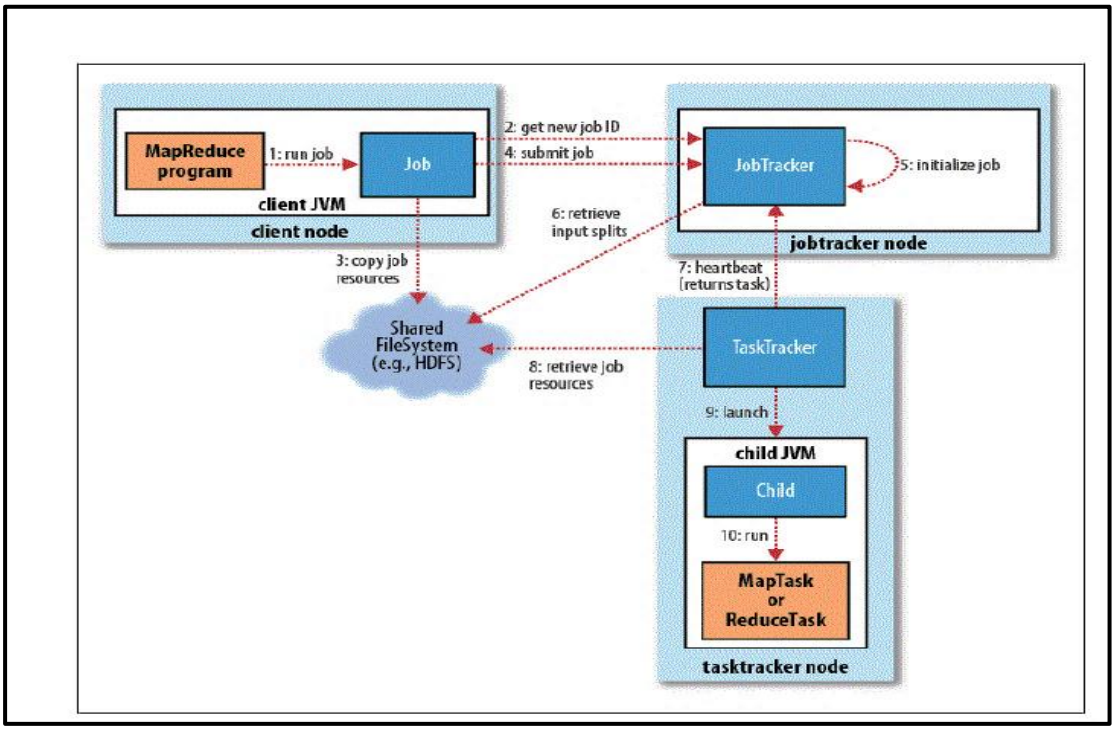

Figure II.6 : Schéma de fonctionnement de MapReduce [5]

L'exécution de MapReduce se fait selon la démarche suivante :

1- Lancement du job

Le processus de lancement du job est établi par JobSummitter qui effectue les opérations suivantes (**étape 1** de la figure):

- Demande aux JobTrackers l'ouverture d'un nouvel job ID(en appelant getNewJobId () sur JobTracker) (**étape 2** de la figureII.6) ;

- Vérifie si les fichiers en input et output existent bien, si ce n'est pas le cas il retourne une erreur ;
- Etablit la liste des sous tâches qui seront données aux tasktrackers, et renvoi une erreur en cas de problème (exemple : fichier de l'input introuvable) ;
- Copie les ressources nécessaires pour exécuter le travail, y compris le fichier JAR (programme MapReduce) du job, les paramètres de configuration Hadoop (hdfs.xml et core-site.xml) et les sous-tâches. La copie se fait dans le système de fichiers du JobTracker dans un répertoire nommé d'après l'ID du job. Le JAR du job est copié avec une réplication haute (contrôlé par le paramètre Mapred.submit.replication dans le fichier mapred-site.xml, qui est par défaut (10), de sorte qu'il y'est beaucoup de copies à travers le cluster pour les tasktrackers (**étape 3** de la Figure II.6) ;
- Indique au JobTracker que le job est prêt pour l'exécution en appelant submitJob () sur JobTracker (**étape 4** de la figure II.6).

2- Initialisation du job

Une fois que le jobtracker exécute le submitJob () les actions suivantes sont déclenchées :

- Création d'un objet pour représenter le job ou la tâche à exécutée et lance un processus pour garder une trace de l'état d'avancement du job (**étape 5**) ;
- Récupération de la liste des sous tâches (input split compute) crées précédemment lors de l'étape3, récupération du nombre maximum de Reduce tasks crées est déterminé par la variable Mapred.Reduce.tasks dans {$HADOOP_HOME}/conf/mapred-site.xml(**étape 6**).

3- Affectation des tâches

- Les TaskTrackers envoient régulièrement des Heartbeat au JobTracker pour lui signifier leurs disponibilités à exécuter des jobs, si le Jobtracker possède des jobs en fille d'attente il confiera la tâche au TaskTracker ;**(étape 7)**
- Après attribution du job au TaskTracker, il commence tout d'abord par localiser le fichier JAR (copié lors de **l'étape 3**) en le copiant depuis le système de fichiers partagé. Il copie également tous les fichiers nécessaires à partir du cache distribué par l'application sur le disque local.**(étape 8)**

4- Exécution du job

TaskRunner lance une nouvelle machine virtuelle Java (JVM, **étape 9**) pour exécuter chaque tâche (**étape 10**)

II.2.2.4 Hadoop MapReduce 2.x: YARN

YARN est un nouveau composant dans l'architecture maître-esclave à deux niveaux par rapport aux architectures précédentes des versions 0.x et 1.x. Le Yarn ressource manager, coordonne l'attribution des ressources de calcul sur le cluster;

- Les YARN node managers, lancent et contrôlent les conteneurs de calcul sur les machines de la grappe;
- Le MapReduce application master, coordonne les tâches en cours d'exécution de MapReduce. Ce dernier avec les tâches MapReduce sont exécutés dans des conteneurs qui sont programmés par le YARN ressources manager et gérés par le Node Managers;
- Le système de fichiers distribué HDFS, est utilisé pour le partage de fichiers de travail entre les autres entités. [4]

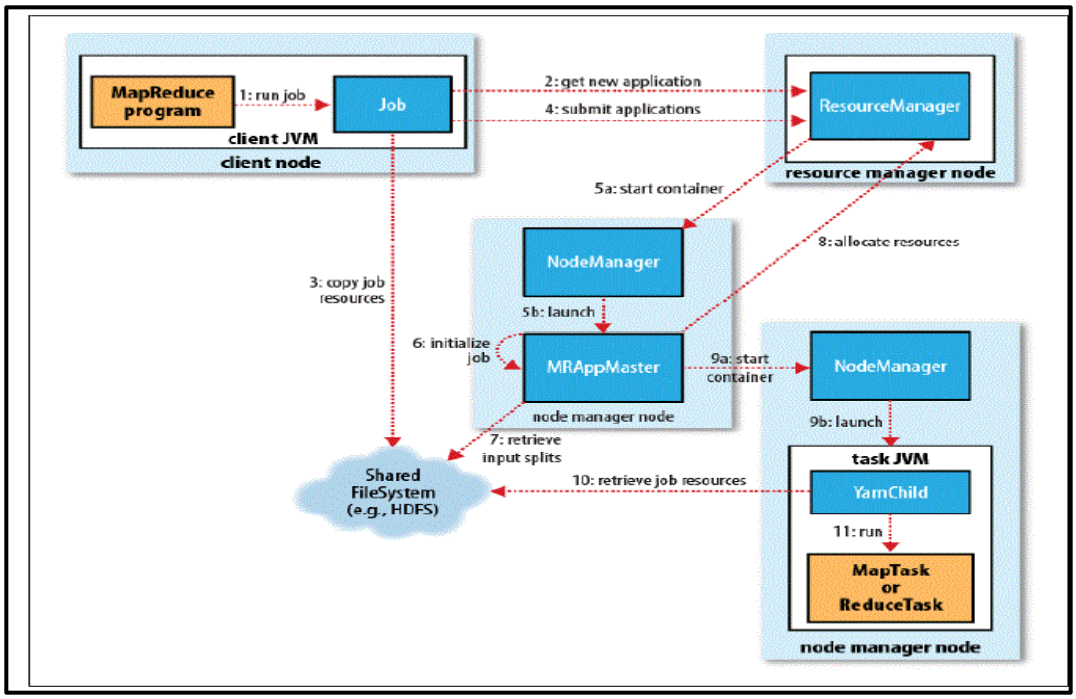

Figure II.7 : Schéma de fonctionnement de MapReduce 2. [5]

II.2.2.5 MapReduce et HDFS

Dans un écosystème d'Hadoop multi nœuds, on peut avoir sur une même machine physique l'exécution du JobTracker et du NameNode, aussi l'exécution sur une même machine physique d'un TaskTracker avec un DataNode (voir figure9).

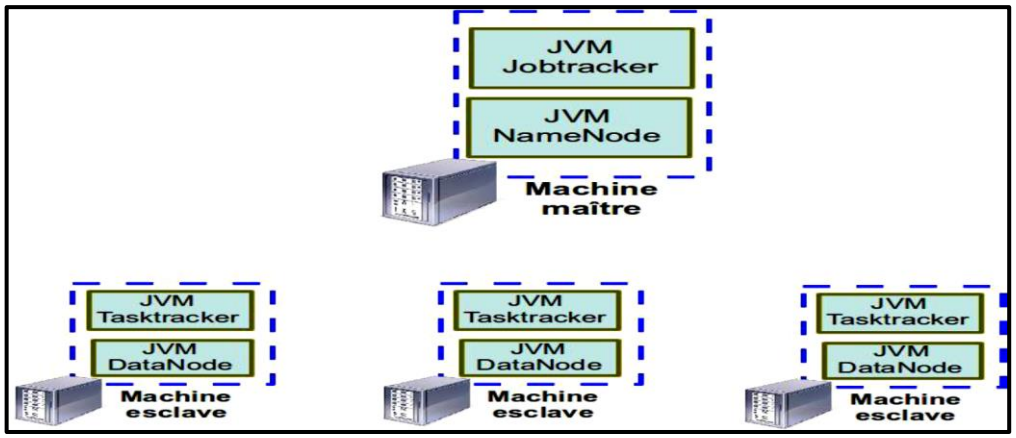

Figure II.8 : Architecture maître-esclave du MapReduce [9]

II.2.3 Écosystème d'Hadoop

Aujourd'hui il est difficile de se retrouver dans la " jungle " d'Hadoop pour les raisons suivantes :

- Ce sont des technologies jeunes ;
- Beaucoup de travaux d'acteurs différents (spécialistes du web, start-up,…) qui veulent prendre le train Big Data en marche ;

Dans une distribution Hadoop on va retrouver les éléments suivant sous leurs équivalences : HDFS, MapReduce, Zookeeper, Hbase, Hive, HCatalog, Oozie, Pig, Sqoop,…etc.

On peut toutefois distinguer trois distributions majeures qui sont Cloudera, Horton Works et MapR, toutes les trois se basant sur Apache Hadoop :

- MapR : Noyau Hadoop mais repackagé et enrichi de solutions propriétaires ;
- Cloudera: Fidèle et open source en grande partie sauf pour les outils d'administration ;
- Horton Works: Fidèle à la distribution Apache et donc 100% open source. [20]

II.2.4 Outils composant le noyau Hadoop

- **HDFS (Hadoop Distributed File System)** : HDFS est un système de fichiers Java utilisé pour stocker des données structurées ou non sur un ensemble de serveurs distribués. HDFS s'appuie sur le système de fichier natif de l'OS pour présenter un système de stockage unifié reposant sur un ensemble de disques et de systèmes de fichiers hétérogènes. la consistance des données est basée sur la redondance. Une donnée est stockée sur au moins n volumes différents.

- **MapR** : En mai 2011, MapR a annoncé une alternative au système HDFS. Ce système permet d'éviter le SPOF (Single Point Of Failure) représenté par le Name Node. Ce système n'est pas inconnu car il s'agit de HBase, dont elle propose une version propriétaire.
- **HBase (Apache)** : HBase est un sous-projet d'Hadoop, c'est un système de gestion de base de données non relationnel distribué, écrit en Java, disposant d'un stockage structuré pour les grandes tables. HBase est inspirée des publications de Google sur BigTable. Comme BigTable, c'est une base de données orientée colonnes. HBase est souvent utilisé conjointement au système de fichiers HDFS, ce dernier facilitant la distribution des données de HBase sur plusieurs nœuds. Contrairement à HDFS, HBase permet de gérer les accès aléatoires read/write pour des applications de type temps réel.
- **Cassandra (Facebook)** : Cassandra est une base de données orientée colonnes développée sous l'impulsion de Facebook. Cassandra supporte l'exécution de jobs MapReduce qui peuvent y puiser les données en entrée et y stocker les résultats en retour (ou bien dans un système de fichiers).Cassandra, comparativement à Hbase, est meilleure pour les écritures alors que ce dernier est plus performant pour les lectures.
- **MapReduce**: Initialement crée par Google pour son outil de recherche web. C'est un Framework qui permet la décomposition d'une requête importante en un ensemble de requêtes plus petites qui vont produire chacune un sous ensemble du résultat final : c'est la fonction Map.
- **YARN (HortonWorks)** YARN (Yet-Another-Resource-Negotiator) est aussi appelé MapReduce 2.0, ce n'est pas une refonte mais une évolution du Framework. [5]

II.2.5 Les outils de Requêtage et de Scripting des données dans Hadoop

- **Hive (Facebook)** : Hive est à l'origine un projet Facebook qui permet de faire le lien entre le monde SQL et Hadoop. Il permet l'exécution de requêtes SQL sur un cluster Hadoop en vue d'analyser et d'agréger les données. Le langage SQL est nommé HiveQL. C'est un langage de visualisation uniquement, c'est pourquoi seules les instructions de type "Select" sont supportées pour la

manipulation des données. Dans certains cas, les développeurs doivent faire le Mapping entre les structures de données et Hive.

- **Pig (Yahoo) :** Apportent un modèle de développement de plus haut niveau, et donc beaucoup plus expressif et simple à appréhender, afin de démocratiser l'écriture de traitements MapReduce. Pig se rapproche plus d'un ETL où on part d'un ou plusieurs flux de données que l'on transforme étape par étape jusqu'à atteindre le résultat souhaité. Les différentes étapes de la transformation sont exprimées dans un langage procédural (Pig Latin).Pig est à l'origine un projet Yahoo qui permet le requêtage des données Hadoop à partir d'un langage de script.

  Contrairement à Hive, Pig est basé sur un langage de haut niveau Pig Latin qui permet de créer des programmes de type MapReduce. Il ne dispose pas d'interface web, Pig se base sur HDFS et MapReduce pour exécuter le script en backround (en arrière-plan). [5]

II.2.6 L'outil d'intégration SGBD-R (Relationnel) Sqoop (Cloudera)

Sqoop permet le transfert des données entre un cluster Hadoop et des bases de données relationnelles. C'est un produit développé par Cloudera, Il permet d'importer/exporter des données depuis/vers Hadoop et Hive. Pour la manipulation des données Sqoop utilise MapReduce et des drivers JDBC.

II.2.7 Les outils de gestion et de supervision du cluster Hadoop [5]

- **Apache ZooKeeper** : ZooKeeper est un service de coordination des services d'un cluster Hadoop, en particulier, le rôle de ZooKeeper est de fournir aux composants Hadoop les fonctionnalités de distribution, pour cela il centralise les éléments de configuration du cluster Hadoop, propose des services de clustérisassions et gère la synchronisation des différents éléments (événements).

  ZooKeeper est un élément indispensable au bon fonctionnement de Hbase.

- **Apache Ambari (HortonWorks)** : Ambari est un projet d'incubation Apache initié par HortonWorks et destiné à la supervision et à l'administration de

clusters Hadoop. C'est un outil web qui propose un tableau de bord, cela permet de visualiser rapidement l'état d'un cluster. [5]

Ambari dispose d'un tableau de bord dont le rôle est de fournir une représentation :

- De l'état des services ;
- De la configuration du cluster et des services ;
- De l'exécution des jobs ;
- Des métriques de chaque machine et du cluster.

II.2.8Outil d'ordonnancement et de coordination : Apache Oozie (Yahoo)

Oozie est une solution de workflow (au sens scheduler d'exploitation) utilisée pour gérer et coordonner les tâches de traitement de données à destination de Hadoop.

Oozie s'intègre parfaitement avec l'écosystème Hadoop puisqu'il supporte les types de jobs suivant :

- MapReduce (Java et Streaming) ;
- Pig ;
- Hive ;
- Sqoop.

Ainsi que des jobs spécifiques du système telles que des programmes java et des scripts Shell.

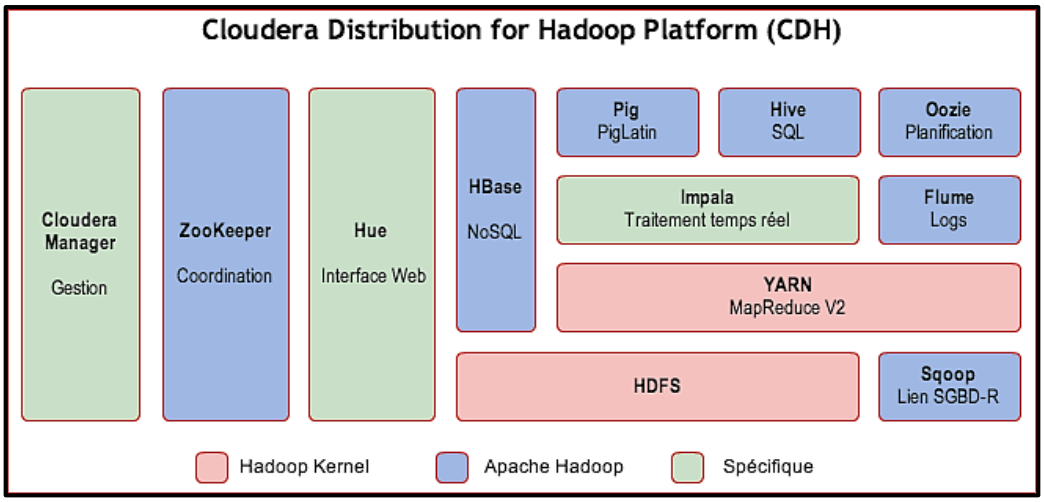

Figure II.9 : Ecosystème d'Hadoop [14]

II.3 Conclusion

Nous avons présenté dans ce chapitre les différents composants d'Hadoop, principalement, HDFS et le modèle de programmation MapReduce. On a étalé sur les aspects théoriques pour pouvoir réaliser et implémenter la solution complète dans le chapitre qui suit.

III.1 Introduction

Dans ce chapitre, nous allons aborder la partie pratique de notre projet qui consiste à décrire l'architecture, la conception technique et la mise en œuvre complète de l'environnement distribué Hadoop. Nous définirons les différentes étapes d'installation et de configuration de notre cluster Hadoop ainsi que les différents tests effectués sur des tâches MapReduce.

III.2 Description de la solution proposée

Pour tester notre système Hadoop, on a proposé comme solution d'exécuter deux jobs en s'appuyant sur le modèle MapReduce écrit en java :

- Le premier va traiter des données textuelles.
- Le deuxième va traiter des données graphiques.

Le stockage des données est effectué dans des fichiers distribués d'Hadoop HDFS.

Pour la réalisation, nous avons procédé à :

- Utilisation de trois machines virtuelles (VM) avec un système d'exploitation Linux 64 bit CentOS-6.6 ;
- Installation d'Hadoop à partir de la distribution Cloudera CDH qui est considérée, actuellement, comme la distribution la plus utilisée dans les entreprises, du fait qu'elle propose une version open source complète qui utilise les principaux composants d'Hadoop (HDFS, MapReduce, Hbase, Pig, Zookeeper).
- Création d'un volume HDFS et rediriger les fichiers textuels vers le volume HDFS ;
- Implantation des jobs MapReduce pour tester la performance de notre cluster.

III.3 L'environnement de travail

Dans la section suivante, on va décrire les outils et systèmes composant notre environnement de travail.

III.3.1 VMware Workstation 11

VMware est un logiciel qui permet la création d'une ou plusieurs machines virtuelles simulant plusieurs systèmes d'exploitation x86 au sein d'un même système d'exploitation. Celles-ci, pouvant être reliées au réseau local avec des adresses IP différentes, tout en étant sur la même machine physique qui existe réellement. Il est possible de faire fonctionner plusieurs machines virtuelles en même temps, Workstation

maximise les ressources de votre ordinateur de façons à permettre l'exécution des applications plus gourmandes dans un environnement virtuel. Il permet aussi de construire des machines virtuelles complexes pour l'exécution des applications Big Data sous la forme d'un cluster Apache Hadoop.

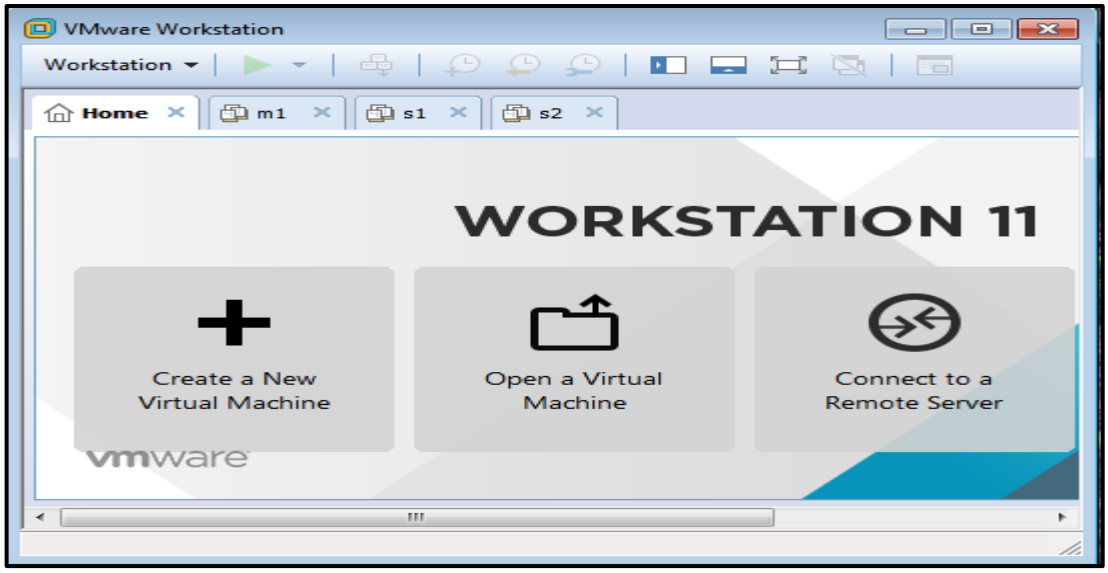

Figure III.1 : VMware Workstation 11

III.3.2 Linux CentOS-6.6

CentOS (Community entreprise Operating System) est une distribution GNU/Linux principalement destinées aux serveurs. Tous ses paquets, à l'exception du logo, sont des paquets compilés à partir des sources de la distribution RHEL (Red Hat Entreprise Linux), éditée par la société Hat. On peut télécharger CentOS sous la forme DVD OU CD.

Figure III.2 : Système linux CentOS 6.6

III.3.3 Cloudera CDH

CDH est la distribution open source la plus complète et la plus populaire dans le monde. Cloudera est une société de logiciels américaine cofondée en 2008 par le mathématicien Jeff Hammerbach, un ancien de Facebook. Les autres cofondateurs sont Christophe

Bisciglia, ex-employé de Google, Amr Awadallah, ex-employé de Yahoo, Mike Olson, PDG de Cloudera. En mars 2009, le fonds AccelPartners a investi 5 millions de dollars dans Cloudera, suivie de Diane Greene, la cofondatrice de VMware, de MartenMickos, l'ex-PDG de MySQL, et de GideonYu, le responsable des finances de Facebook.

La firme Cloudera se consacre au développement de logiciels fondés sur Apache Hadoop, permettant l'exploitation de Big Data, à savoir des bases de données accumulant plusieurs pétaoctets.

CDH contient les principales composantes d'Apache Hadoop pour le stockage évolutif et des calculs distribués.

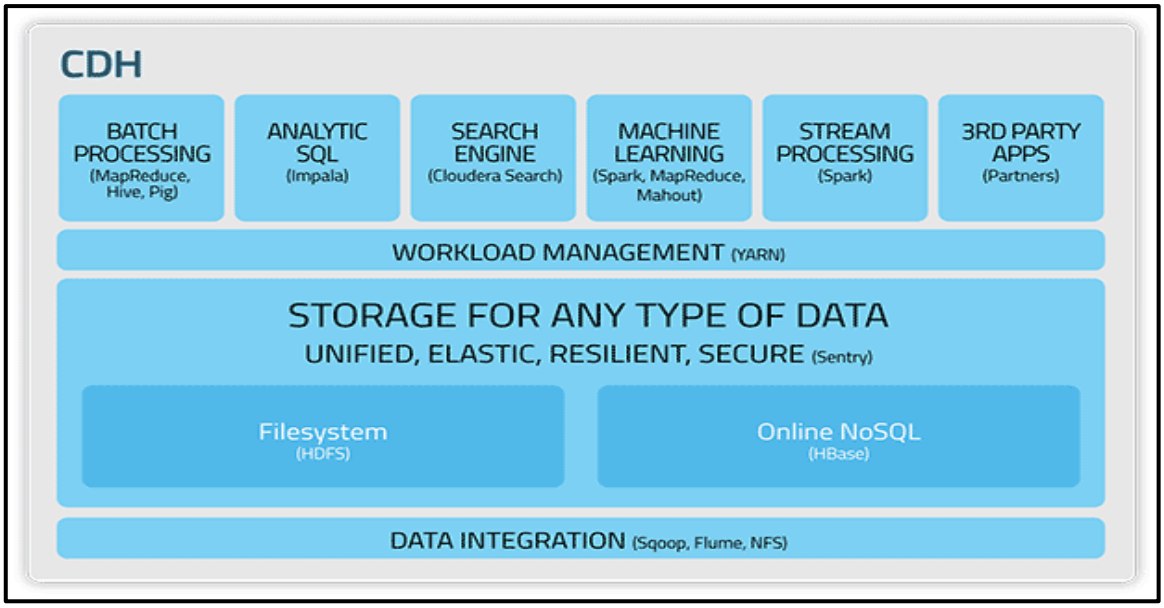

Figure III.3 : Les composants de la distribution Hadoop de Cloudera

III.4 Installation d'Hadoop en single node

Pour installer Hadoop en mode Single Node, on passe par les étapes suivantes :

- **Création d'une machine virtuelle**

Pour créer notre première machine virtuelle **m1** (adresse IP=192.168.8.155), on clique sur New Virtuel Machine sous **VMware** puis on spécifie le système utilisé par l'image du système **Centos** qu'on a téléchargé « CentOS-6.6-x86_64-bin-DVD1.iso », ensuite on passe va affecter à notre machine 1 GO de la RAM et 60 GO de disque dur. Pour le mode Network on utilise le **NAT** car il permet de connecter plusieurs machines virtuelles entre eux sur le même poste physique et d'avoir accès à internet.

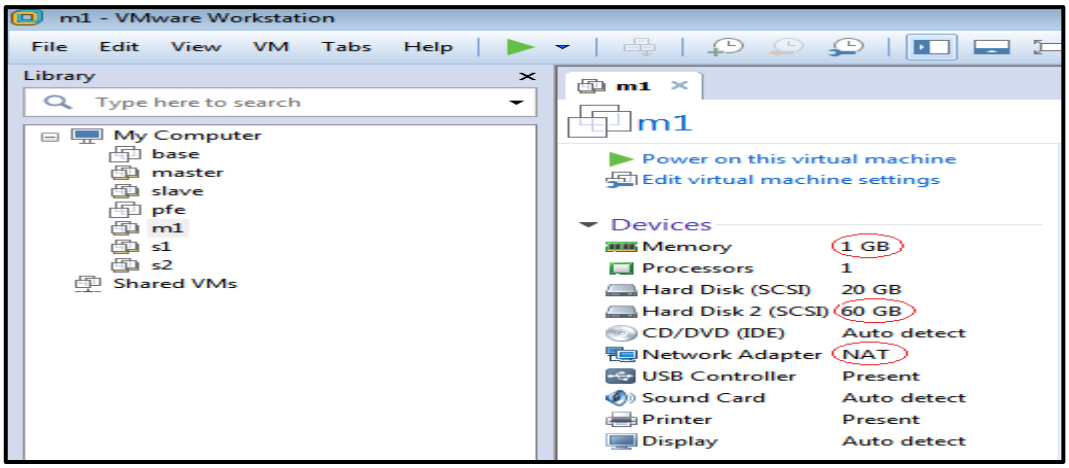

Figure III.4 : Configuration de la machine virtuelle **m1**

Avant toute installation de nouveaux paquets, on a mis à jour le cache des paquets sur notre machine. La commande suivante sert à télécharger la nouvelle liste des paquets proposés par le dépôt :

```
yum – update.
```

A la fin, on va modifier le nom du host (hostname) de la machine dans le fichier /etc/sysconfig/network.

- **Installation de JAVA**

Le composant java est un outil indispensable pour l'exécution des jobs Hadoop, dans notre cas on a installé la version de java qui est compatible avec la version du Cloudera CDH 4.7.1.

```
[root@m1 pfe]# java -version
java version "1.6.0_31"
Java(TM) SE Runtime Environment (build 1.6.0_31-b04)
Java HotSpot(TM) 64-Bit Server VM (build 20.6-b01, mixed mode)
[root@m1 pfe]# amine djamil
```

Figure III.5 : Version du Java installée

Après l'installation, on va informer Linux qu'Oracle Java 1.6.0_31 sera désormais l'environnement Java par défaut en utilisant la commande suivante :

```
Sudo update-alternatives –install "/usr/bin/java" "java" "/usr/local/java/jdk1.6.0_31/bin/java" 1
```

A la fin, on va ajouter dans le fichier descripteur des variables du système **./etc/profile**, les lignes prescrites dans la figure suivante, pour définir les variables d'environnement JAVA_HOME.

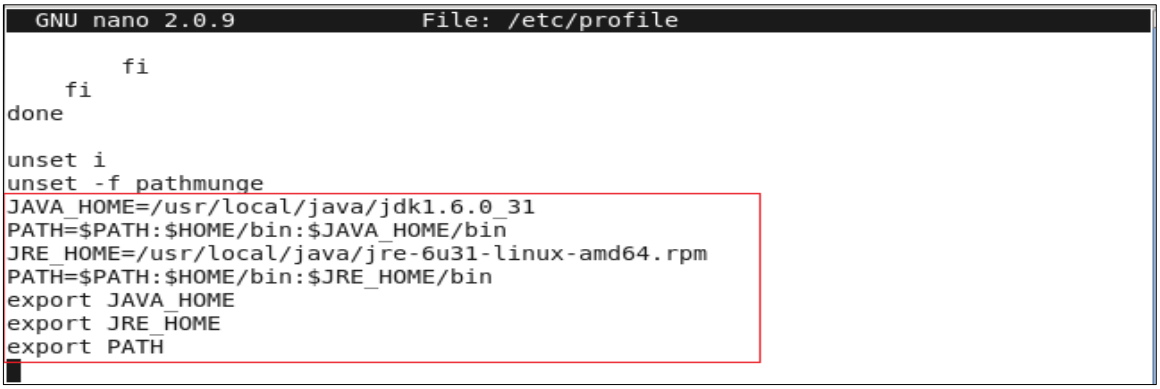

Figure III.6 : Fichier ./etc/profile

- **Installation de Cloudera**

Pour l'installation du Cloudera CDH 4.7.1 (version stable et testée), plusieurs alternatives nous étaient possibles :

1. **Installation de Cloudera Quickstart vm 4.7.0-0**

Le Quickstart VM contient un cluster d'un nœud unique Hadoop, avec des exemples de données, des requêtes, des scripts et Cloudera manager pour gérer votre cluster. Cette machines est disponible pour VMware, VirtualBox, et KVM. Elle offre la possibilité d'avoir et d'utiliser tous les services d'Hadoop. Au début on a installé et tester la version cloudera-quickstart-vm-4.7.0-0-vmware sous VMware. Elle nous a été très utile pour comprendre initialement le fonctionnement de base d'Hadoop. La figure suivante montre tous les composants configurés avec Cloudera VM.

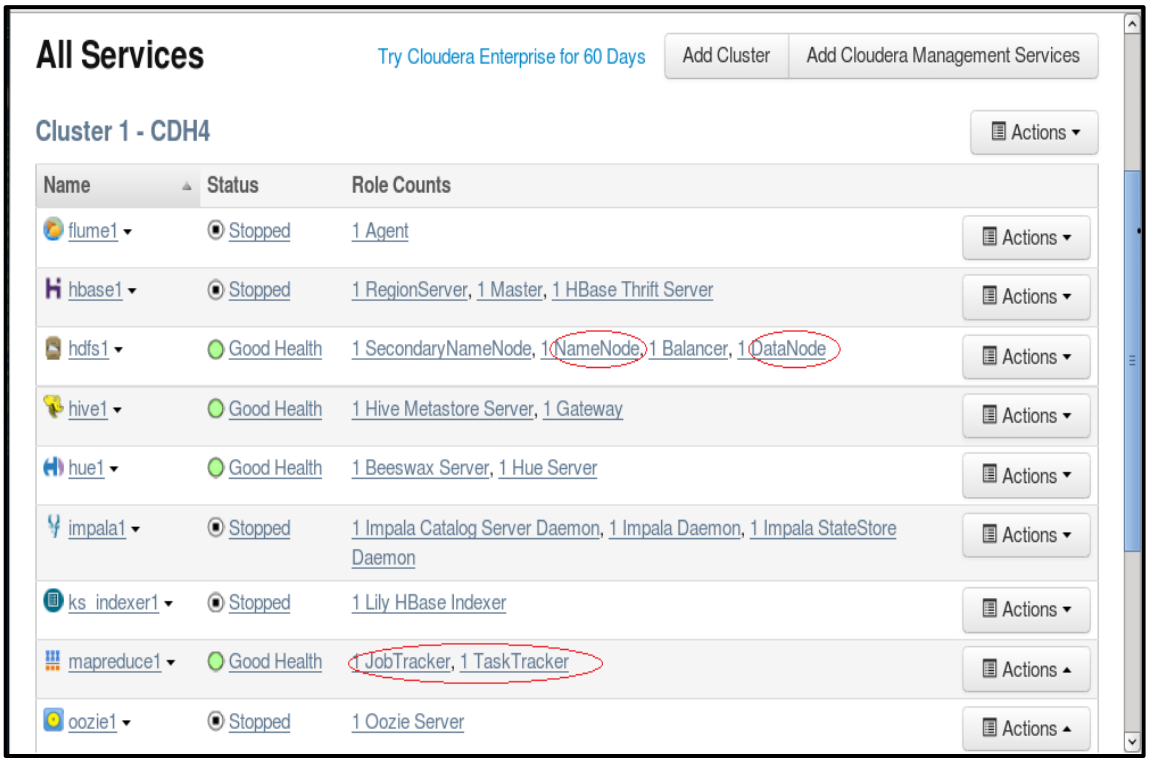

Figure III.7 : Cloudera Quickstart VM

On a remarqué que tous les services d'Hadoop ont démarré automatiquement dès qu'on lance la machine VM :

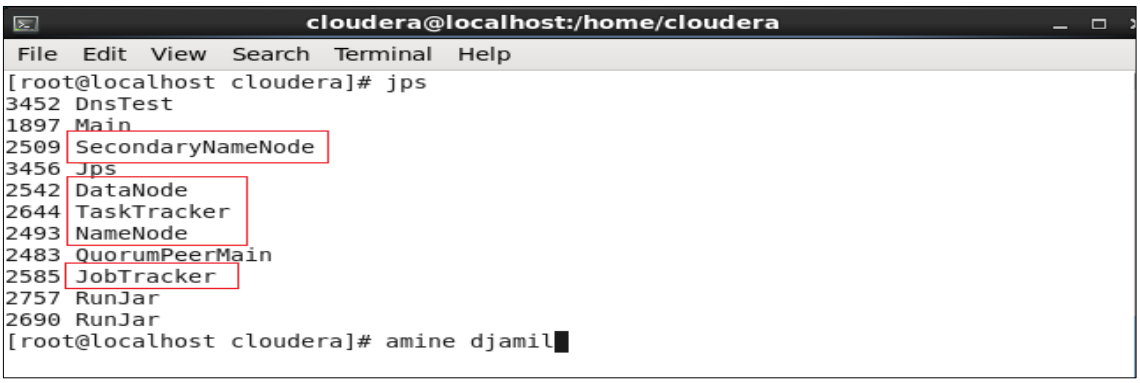

Figure III.8 : Processus java d'Hadoop

Le problème rencontré dans cette méthode d'installation est lié au fait que Cloudera Quickstart ne fonctionne qu'avec un seul nœud et toutes les modifications effectuées doivent être réinitialisées après chaque redémarrage de la machine.

2. Installation automatique avec Cloudera Manager

Cloudera Manager Edition gratuit automatise l'installation et la configuration de CDH4 sur un cluster entier si vous avez accès au root ou sudo SSH sans mot de passe pour les machines de votre cluster. On n'a pas réussi d'installer Cloudera Manager à cause des exigences et des pré-requis hardware (RAM et CPU).

3. Installation manuelle du CDH repository

La distribution Cloudera fournit une installation modulaire par package assez souple. Il est donc possible d'ajouter les différents composants de la distribution Hadoop via le gestionnaire de packages.

Après ces tentatives, on s'est fixé sur cette troisième option d'installation manuelle de CDH 4, en passant par les étapes suivantes :

- Télécharger le fichier de configuration de l'entrepôt de Cloudera à partir du site (http://archive.cloudera.com/cdh4/one-click-install/redhat/6/x86_64/ cloudera-cdh-4-0.x86_64.rpm) ou avec la ligne de commande :

```
wget archive.cloudera.com/cdh4/one-click-install/redhat/6/x86_64/Cloudera-cdh-4-0.x86_64.rpm
```

- Exécuter la commande pour installer le fichier téléchargé :

```
yum –nogpgcheck localinstall Cloudera-cdh4-0.x86_64.rpm
```

- La figure suivante montre la version de Cloudera CDH installé avec la version d'Hadoop utilisée.

```
[root@m1 pfe]# hadoop version
Hadoop 2.0.0-cdh4.7.1
Subversion file:///data/jenkins/workspace/generic-package-rhel64-6-0/topdir/BUILD/hadoop-2.0.0-cdh4.7.1/src/hadoop-common-project/hadoop-common -r Unknown
Compiled by jenkins on Tue Nov 18 08:10:25 PST 2014
From source with checksum efbc3ec8dbd142fa33469ff22e2ef51d
This command was run using /usr/lib/hadoop/hadoop-common-2.0.0-cdh4.7.1.jar
[root@m1 pfe]# amine djamil
```

Figure III.9 : Version d'Hadoop utilisée

III.5 Configuration d'un cluster Hadoop

Après avoir réalisé les étapes de la section précédente, on passe à la configuration de notre cluster Hadoop multi nœuds, on va utiliser un cluster de trois machines virtuelles, un master **m1** (Namenode, JobTracker) et deux slaves : **s1** (Datanode, TaskTracker) et **s2** (Datanode, TaskTracker). Pour y arriver il faut suivre les étapes suivantes :

- **Grâce à la virtualisation**, on a cloné 2 machines virtuelle s1(192.168.8.159) et s2 (192.168.1.157) à partir de la machine m1 (s1 et s2 sont des copies du m1), après on ne change que les hosts des deux machines.
- **Ajouter les trois hosts et leurs adresses IP** dans le chemin linux : /etc/hosts dans les trois machines.

```
  GNU nano 2.0.9              File: /etc/hosts

27.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.8.155 m1.example.com m1
192.168.8.159 s1.example.com s1
192.168.8.157 s2.example.com s2
```

Figure III.10 : Fichier /etc/hosts des trois machines.

- **Configuration de SSH,** Hadoop nécessite un accès SSH pour gérer les différents nœuds du cluster mais avant cela, on doit générer une clé SSH sans mot de passe sur la machine **m1.**

```
[root@m1 pfe]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
a6:5b:d3:63:78:64:05:6a:20:b8:7f:3e:d0:61:ee:98 root@m1.example.com
The key's randomart image is:
+--[ RSA 2048]----+
|    .. .   .     |
|   .  . . . .    |
|    .    o  .    |
|   .   o.   .    |
|    . + .S o     |
|     o +o =      |
|      B. + =     |
|     E +o + .    |
|       ..        |
+-----------------+
[root@m1 pfe]# amine djamil
```

Figure III.11 : Générateur d'une clé publique RSA dans m1 (master)

Puis on copier cette clé publique sur les machines slaves (voir la figure)

```
[root@m1 .ssh]# ls
authorized_keys  id_rsa  id_rsa.pub
[root@m1 .ssh]# scp id_rsa.pub root@s2:/home/pfe/.ssh/authorized_keys
Warning: Permanently added 's2,192.168.8.157' (RSA) to the list of known hosts.
root@s2's password:
id_rsa.pub                                    100%  401     0.4KB/s   00:00
[root@m1 .ssh]# scp id_rsa.pub root@s:/home/pfe/.ssh/authorized_keys
^C[root@m1 .ssh]# scp id_rsa.pub root@s1:/home/pfe/.ssh/authorized_keys
Warning: Permanently added 's1,192.168.8.156' (RSA) to the list of known hosts.
root@s1's password:
id_rsa.pub                                    100%  401     0.4KB/s   00:00
[root@m1 .ssh]# amine djamil
```

Figure III.12 : Copie de la clé publique sur les nœuds s1 et s2

- **Installation des composants de la distribution Hadoop**

Après avoir configuré notre cluster, il faut ajouter les différents composants requis via le gestionnaire de packages :

- Pour le master m1, on ajoute deux packages NameNode et JobTracker avec les lignes de commandes suivantes :

```
yum install Hadoop-0.20-mapreduce-jobtracker
```

```
yum install Hadoop-hdfs-namenode
```

- Pour les deux slaves, on ajoute les packages DataNode et TaskTracker en exécutons la commande suivante :

```
yum install Hadoop-0.20-mapreduce-tasktracker Hadoop-hdfs-datanode
```

➢ **Configuration des fichiers d'environnement d'Hadoop**

Tous les fichiers de configuration d'Hadoop sont disponibles dans le répertoire /etc/Hadoop/conf. Ces paramètres de configuration sont lus par le Framework Java d'HDFS et de MapReduce.

- Configuration du fichier /etc/Hadoop/conf/**core-site.xml** (voir figure III.)

On doit spécifier l'URI du NameNode, qui devra être connue par tout le système ; pour notre cas c'est la machine **m1**. La configuration doit être établie dans le master et les slaves **s1** et **s2**.

Paramètre	Valeur	Exemple
fs.default.name	URI of NameNode.	hdfs://**m1**.example.com/

```
  GNU nano 2.0.9       File: /etc/hadoop/conf/core-site.xml

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://m1.example.com:8020</value>
</property>
</configuration>
```

Figure III.13 : Exemple de notre core-site.xml

- Configuration de fichier /etc/Hadoop/conf/**mapred-site.xml** (voir figure III.)

Dans ce fichier on doit définir le host et le numéro de port du JobTracker qui est la machine **m1**.

Paramètre	Valeur	Exemple
mapred.job.tracker	Host ou IP et le port du JobTracker	m1.example.com :8021
mapred.local.dir	Liste des chemins séparés par des virgules, sur le système de fichier local ou les données de mapreduce seront écrites.	/home/disk1/mapred/local, /home/disk2/mapred/local

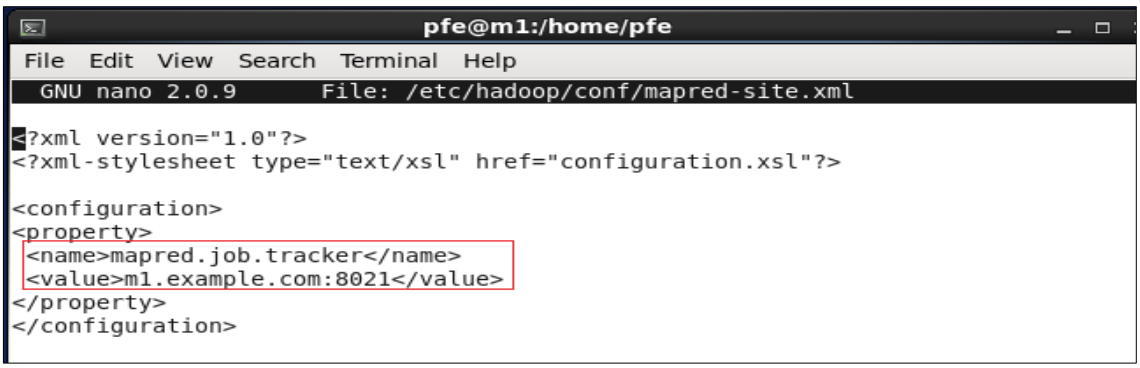

Figure III.14 : Exemple de notre mapred-site.xml (m1)

Les mêmes modifications du fichier mapred-site.xml, on les refait sur les deux nœuds **s1** et **s2** en ajoutant les répertoires ou les TaskTracker enregistrant les données.

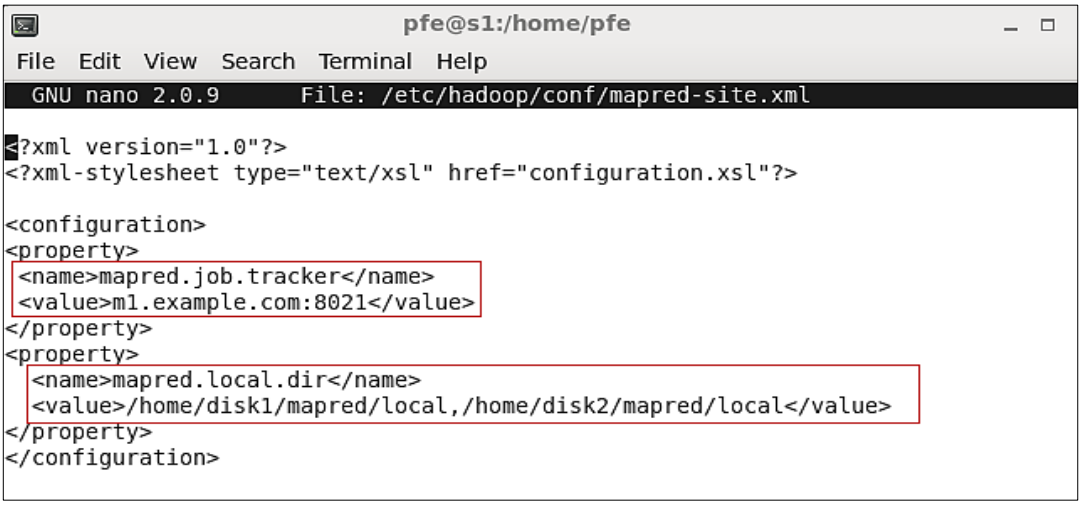

Figure III.15: Exemple de notre mapred-site.xml (s1, s2)

Ce fichier doit être configuré dans les trois nœuds, il contient les paramètres de configuration pour le processus **HDFS**.

Paramètre	Valeur	Exemple
dfs.name.dir	la liste des répertoires sur lequel le NameNode stock les	/home/disk1/dfs/nn, /home/disk2/dfs/nn

	journaux de transactions	
Dfs.replication	Nombre de réplication d'un bloc sur les DataNodes	Valeur 2
dfs.data.dir	la liste des répertoires sur lesquels leurs blocs sont stockés	/home/disk1/dfs/dn, /home/disk2/dfs/dn

Notre configuration du fichier hdfs-site.xml est illustrée dans la figure III.15 pour le master m1et dans la figure III.16 pour les slaves s1 et s2 :

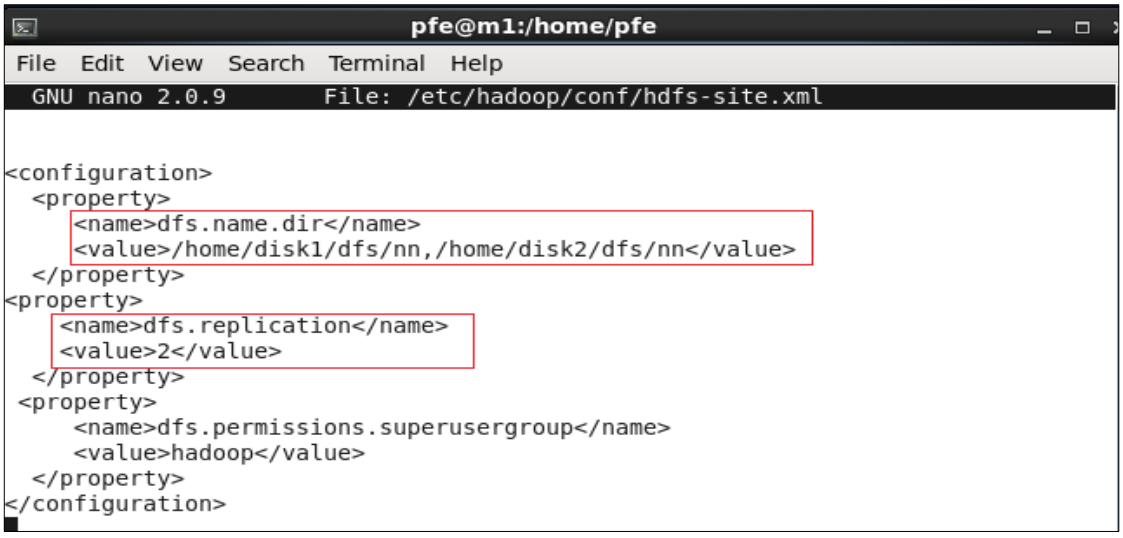

Figure III.16 :Exemple de notre hdfs-site.xml (m1)

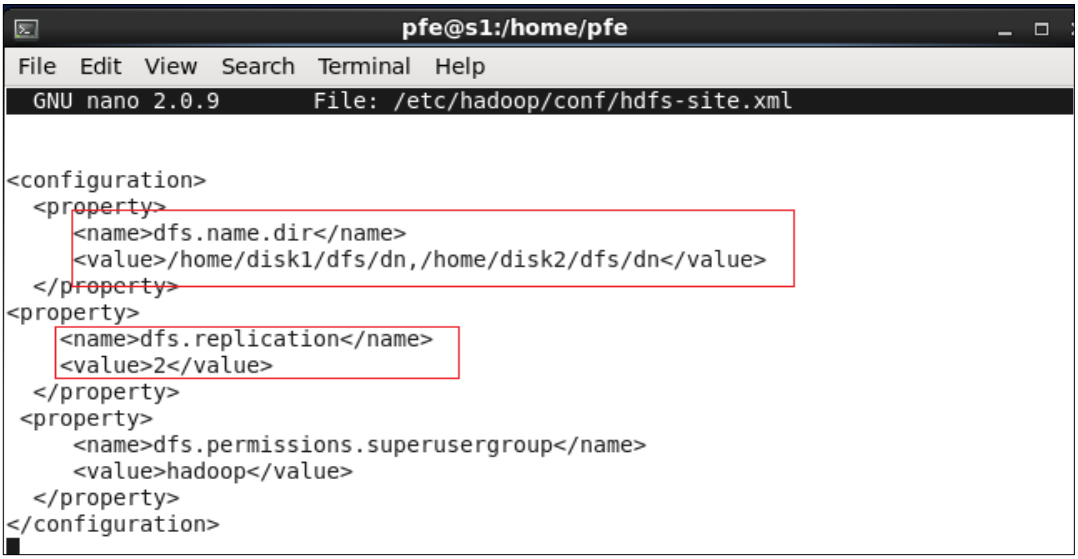

Figure III.17 : Exemple de notre hdfs-site.xml (s1, s2)

➢ **Démarrer les services d'Hadoop** en exécutant les commandes suivantes :

- Avant de démarrer le serveur Hadoop dans le nœud master (NameNode), on doit formater le système de fichiers en HDFS.

```
sudo –u hdfs Hadoop namenode –format
```

```
15/06/01 02:39:40 INFO namenode.FSImage: Image file of size 115 saved in 0 secon
ds.
15/06/01 02:39:40 INFO namenode.NNStorageRetentionManager: Going to retain 1 ima
ges with txid >= 0
15/06/01 02:39:40 INFO util.ExitUtil: Exiting with status 0
15/06/01 02:39:40 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at m1.example.com/192.168.8.155
************************************************************/
[root@m1 conf]# amine djamil
```

Figure III.18 : Résultat de formatage du Namenode.

- Pour démarrer les services HDFS ainsi que MapReduce dans le nœud master NameNode/JobTracker, on exécute :

```
/etc/init.d/Hadoop-hdfs-namenode start
```

```
/etc/init.d/Hadoop-0.20-mapreduce-jobtrackerstart
```

- Pour démarrer les services DataNode et TaskTracker dans les nœuds slave_1 et slave_2, on exécute :

```
/etc/init.d/Hadoop-hdfs-datanode start
```

```
/etc/init.d/Hadoop-0.20-mapreduce-tasktracker start
```

➢ **Vérification de l'état des services d'Hadoop** : pour vérifier l'exécution des processus Java, il faut lancer la commande **jps** dans le terminal linux pour le master (voir la figure III.18), pour le slave_1 et slave_2 voir les figures(III.19 et III.20)

```
[root@m1 conf]# jps
3080 NameNode
3835 JobTracker
3858 Jps
[root@m1 conf]# amine djamil
```

Figure III.19 : Processus java d'Hadoop sur le nœud m1 (master)

```
[root@s1 conf]# jps
2958 DataNode
3211 Jps
3168 TaskTracker
[root@s1 conf]# amine djamil
```

Figure III.20 : Processus java d'Hadoop sur le nœud s1 (slave)

```
[root@s2 conf]# jps
3223 TaskTracker
3243 Jps
3015 DataNode
[root@s2 conf]# amine djamil
```

Figure III.21 : Processus java d'Hadoop sur le nœud s2 (slave)

- Il est possible de vérifier le bon fonctionnement du notre cluster à l'aide des interfaces WEB. Par défaut, elles sont disponible sur le Web UI pour le NameNode : http://m1:50070/

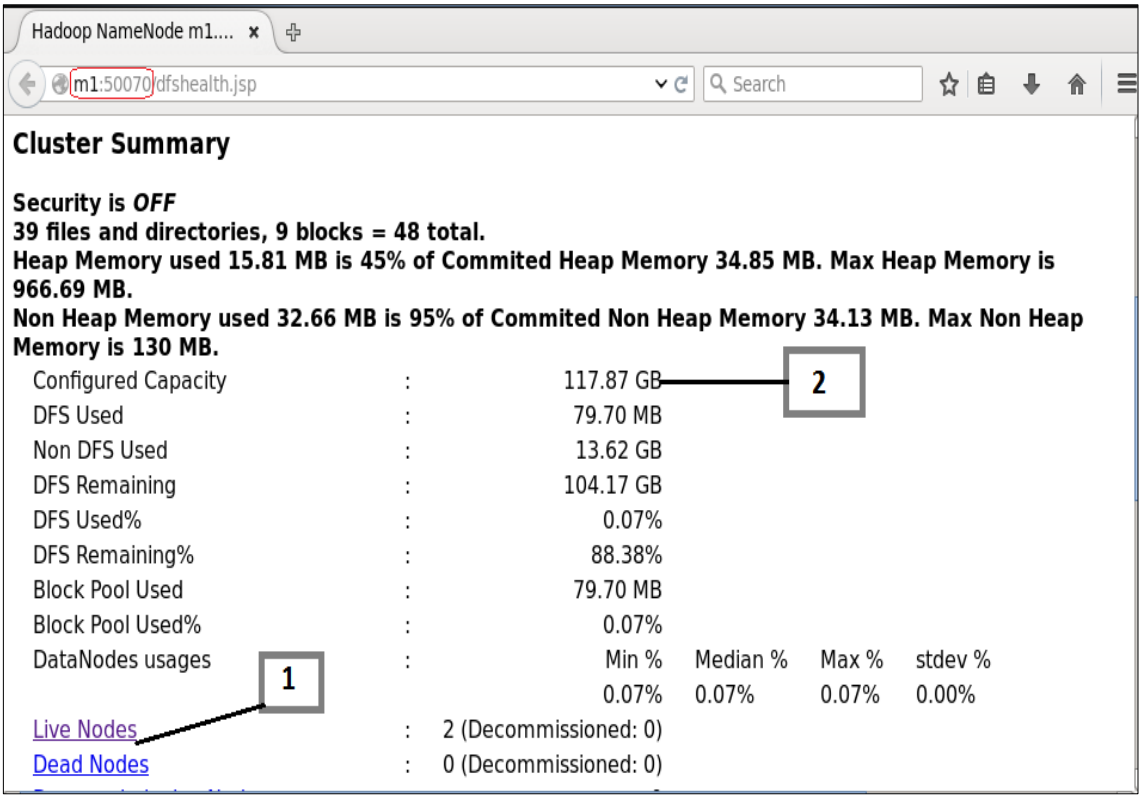

Figure III.22 : L'état de notre cluster Hadoop

- Le commentaire **N°2** dans la figure ci-dessus indique une capacité de 117 GB réservée pour notre cluster.
- Le commentaire **N°1** montre l'existence de deux DataNodes.

Si on clique sur « Live Nodes », on obtient ceux-ci (figure III.23) :

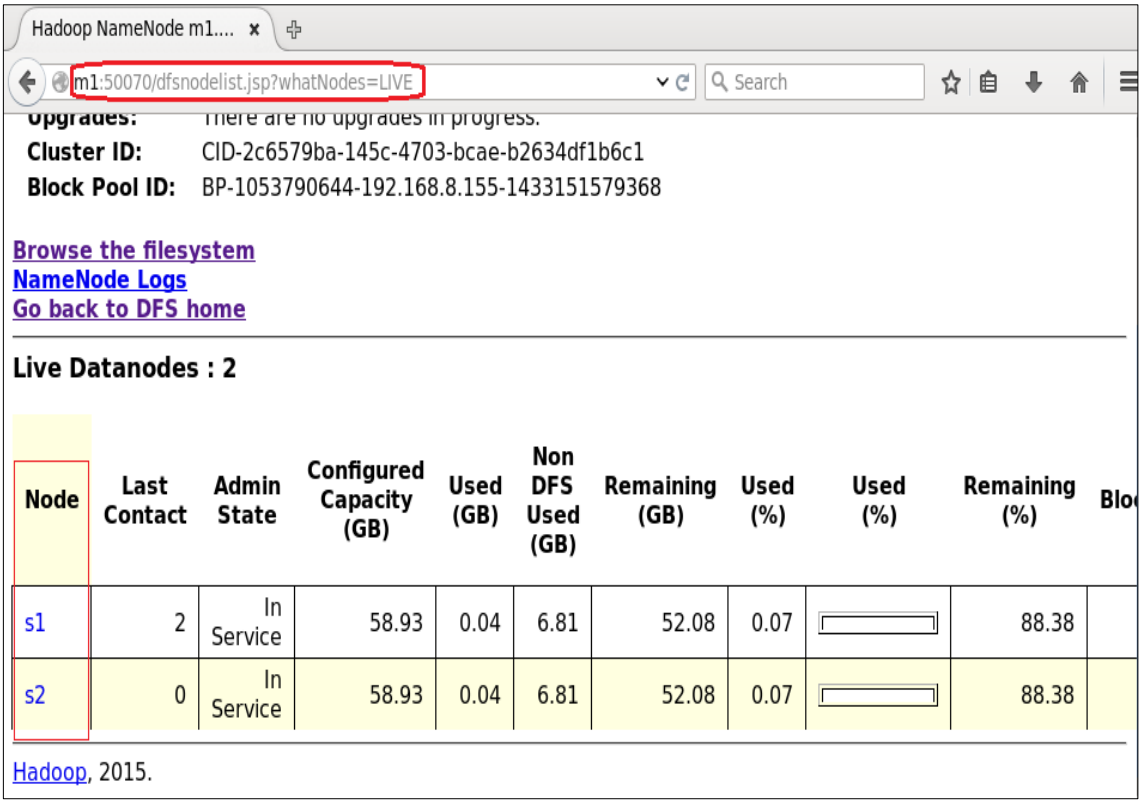

Figure III.23 : Listes des DataNodes (Live Node)

Comme pour le NameNode, il existe une Web UI pour le JobTracker : http://m1:50030

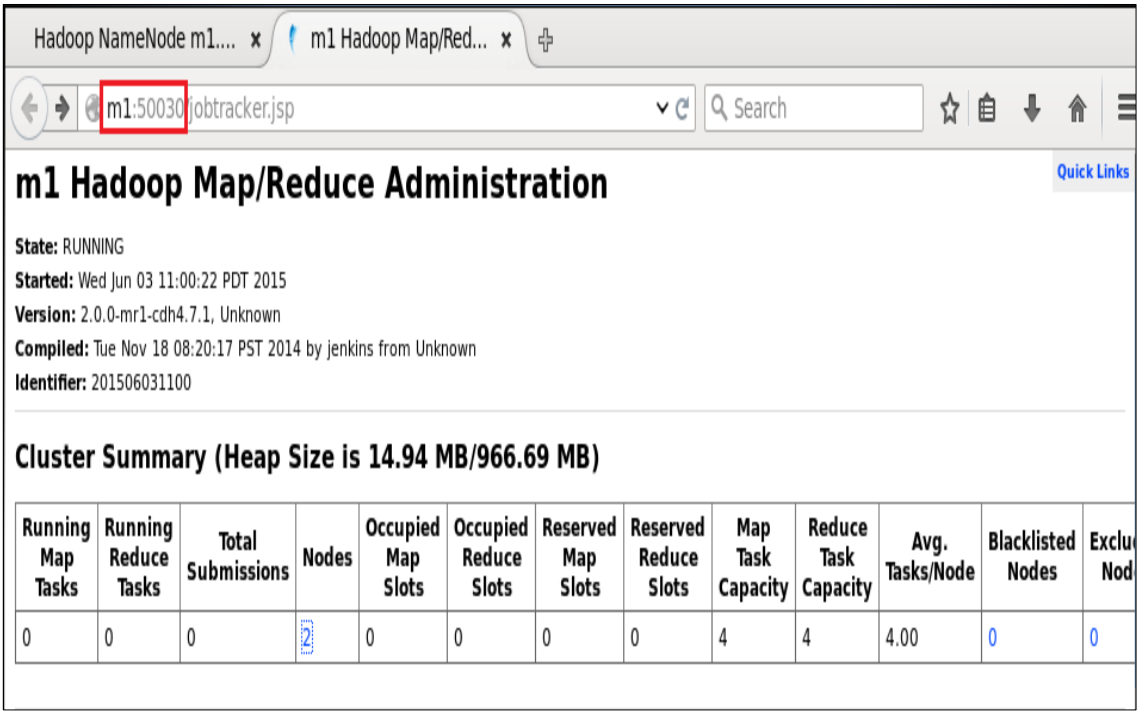

Figure III.24 : L'état de cluster Hadoop, JobTracker

On peut aussi consulter les TaskTrackers dans : http://s1:50060

Figure III.25 : L'état de cluster Hadoop, TaskTracker.

III.6 Test et Evaluation

Pour test et expérimenter l'environnement mis en œuvre, on va utiliser 2 Jobs fournis avec la distribution Cloudera CDH4 (la liste des jobs fournis par Cloudera est en **Annexe C**), à savoir le job **Wordcount** et le job **Pi**. Deux types de tests vont être effectués sur les 2 jobs dans des architectures distribuées à 2 nœuds puis à 3 nœuds.

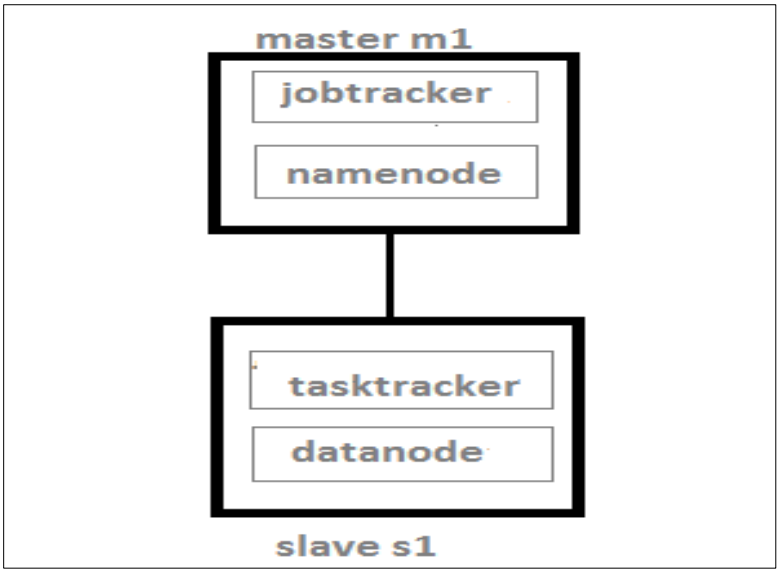

Figure III.26 : Architecture distribuée à 2 nœuds

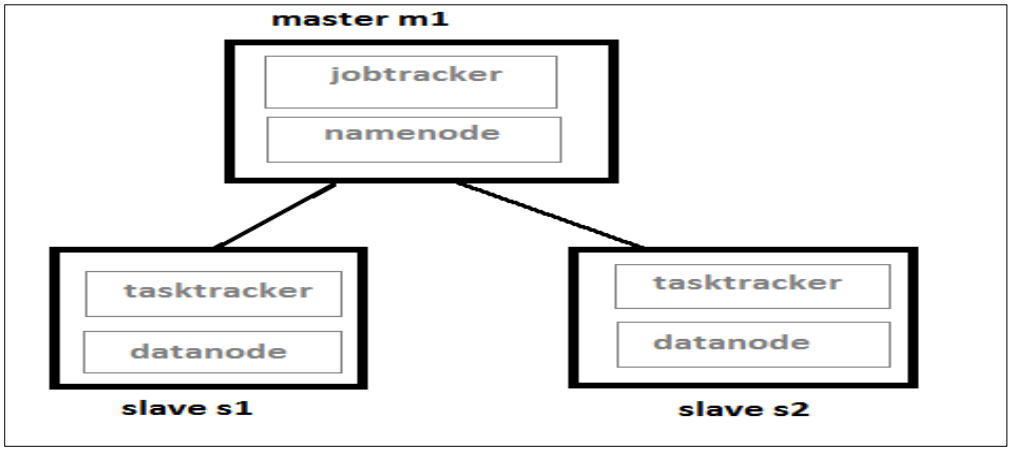

Figure III.27 : Architecture distribuée à 3 nœuds

III.6.1 Job : wordcount.jar

- **Présentation du job**

Wordcount est un programme simple qui compte le nombre d'occurrences de chaque mot dans un ensemble d'entrée donné.

- **Données**

Pour pouvoir tester le job Wordcount, on a généré un fichier qui contient des prénoms avec un script Shell dont la fonction est de prendre en paramètre une source texte (paramètre 1) et d'ajouter son contenu n fois (paramètre 3) dans un fichier de sortie (paramètre 2). Le commentaire N°1 dans la figure III.27 indique la commande utilisée pour générer le fichier prenoms.txt (194 Mo). Le code source du script sc.sh est en Annexe B.

Sur le namenode **m1** on a créé un volume HDFS par la ligne de commande indiquée par le commentaire N°2, ensuite on a inséré le fichier généré depuis le système de fichiers local vers le système de fichiers HDFS (voir la figure III.28, commentaire N°3)

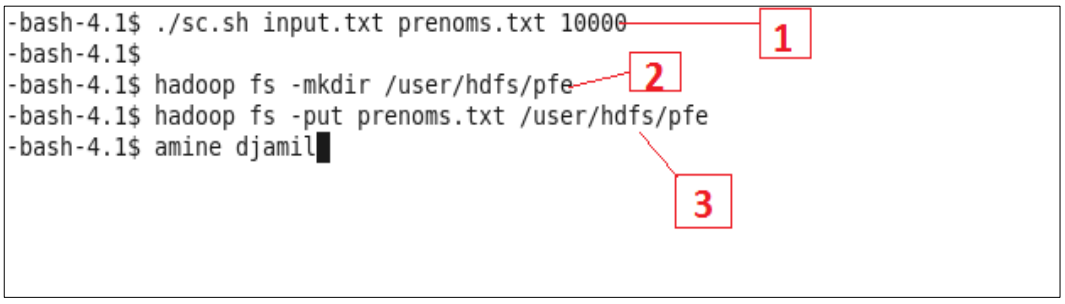

Figure III.28 : Copier fichier prenoms.txt dans le système HDFS

On peut accéder à notre fichier pour visualiser son contenu dans l'un des slaves dans le lien : http://s1:50075.

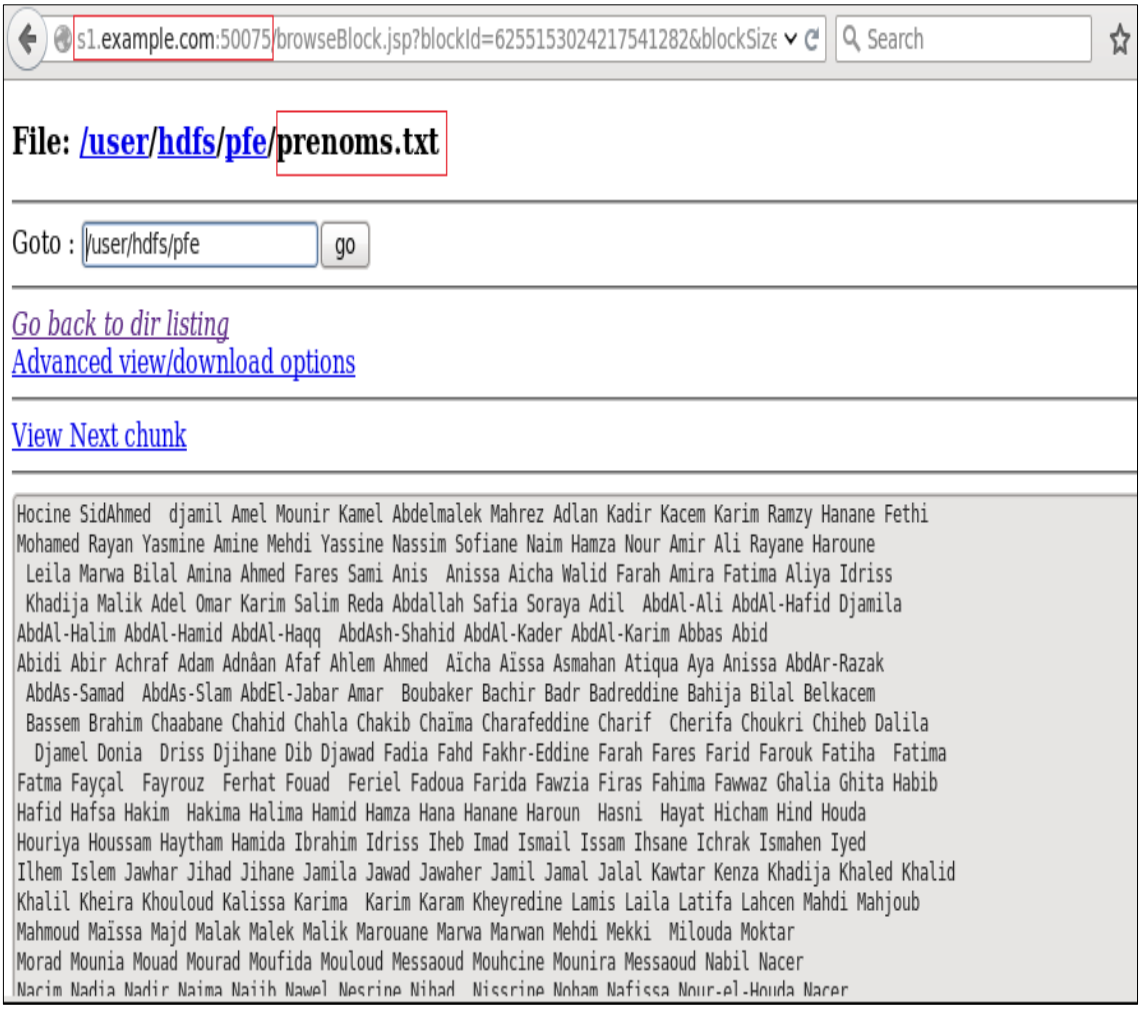

Figure III.29 : Fichier prenoms.txt dans le HDFS

Puisque la taille de notre fichier prenoms.txt est de 194 Mo et la taille du bloc HDFS est de 64 Mo alors 4 blocs seront créés et répliqués 2 fois sur les DataNodes **s1** et **s2**.

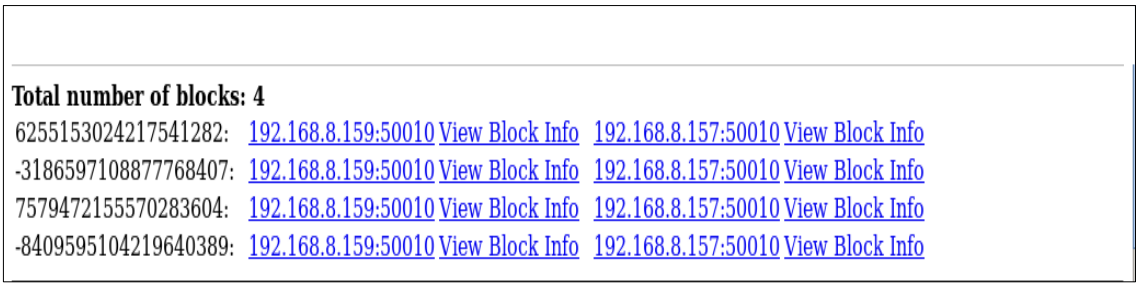

Figure III.30 : Création et réplication du bloc

- **Exécution en mode distribué avec 3 nœuds (master et 2 slaves)**

Pour lancer le job wordcount, il faut donner les paramètres d'entrées **pfe** et les paramètres de sortie **output_pfe2** à l'aide de la ligne de commande suivante :

```
Hadoop jar /usr/lib/Hadoop-0.20-mapreduce/Hadoop-examples.jar wordcount pfe pfe_output
```

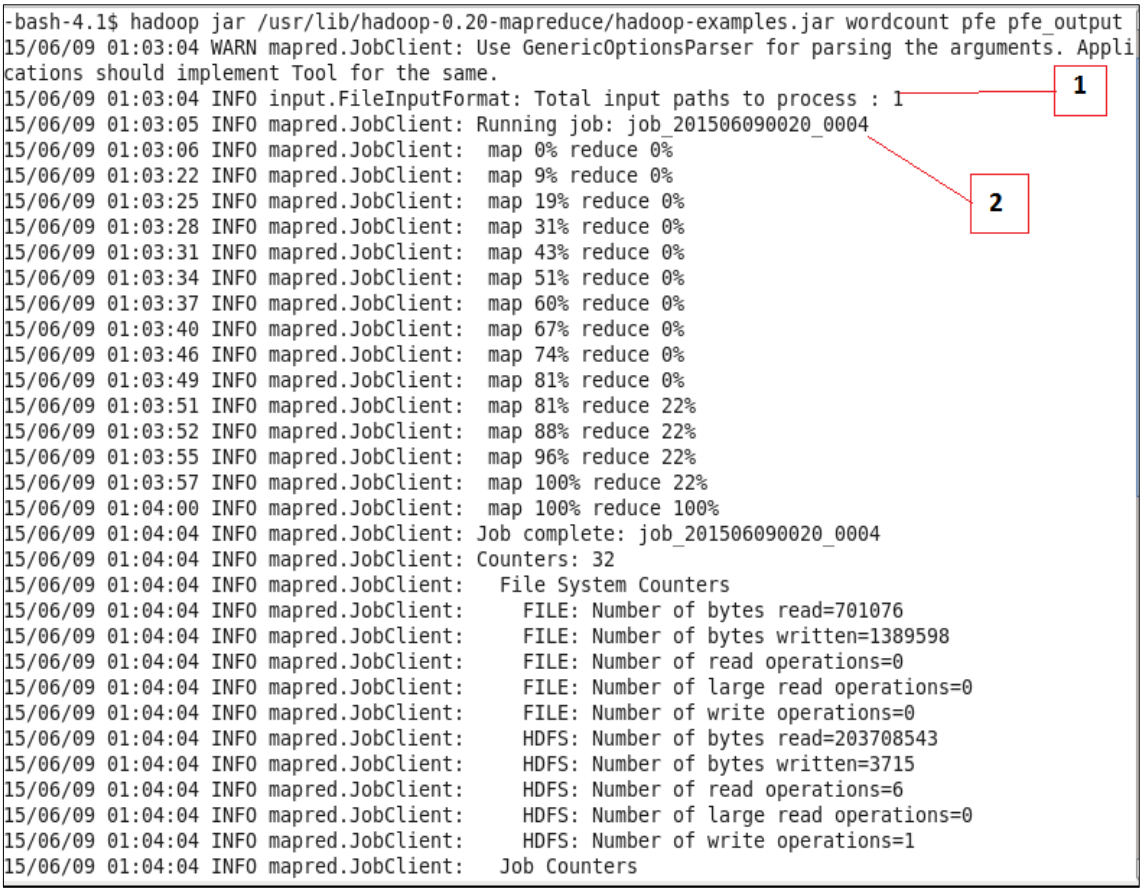

Figure III.31 : Résultat de l'exécution du job wordcount (3 nœuds)

Le commentaire N°1 indique le nombre de fichiers à exécuter, un seul fichier pour ce cas précis. Le commentaire N°2 indique l'identifiant du job qui est en exécution.

- **Exécution du job en mode distribué avec 2 nœuds (master et slave)**

```
15/06/09 01:09:47 INFO hdfs.DFSClient: Excluding datanode 192.168.8.157:50010
15/06/09 01:09:48 INFO mapred.JobClient: Running job: job_201506090020_0006
15/06/09 01:09:50 INFO mapred.JobClient:  map 0% reduce 0%
15/06/09 01:10:13 INFO mapred.JobClient:  map 5% reduce 0%
15/06/09 01:10:16 INFO mapred.JobClient:  map 11% reduce 0%
15/06/09 01:10:19 INFO mapred.JobClient:  map 19% reduce 0%
15/06/09 01:10:22 INFO mapred.JobClient:  map 26% reduce 0%
15/06/09 01:10:25 INFO mapred.JobClient:  map 32% reduce 0%
15/06/09 01:10:28 INFO mapred.JobClient:  map 40% reduce 0%
15/06/09 01:10:32 INFO mapred.JobClient:  map 46% reduce 0%
15/06/09 01:10:35 INFO mapred.JobClient:  map 53% reduce 0%
15/06/09 01:10:38 INFO mapred.JobClient:  map 60% reduce 0%
15/06/09 01:10:40 INFO mapred.JobClient:  map 62% reduce 0%
15/06/09 01:10:41 INFO mapred.JobClient:  map 66% reduce 0%
15/06/09 01:10:43 INFO mapred.JobClient:  map 67% reduce 0%
15/06/09 01:10:55 INFO mapred.JobClient:  map 67% reduce 22%
15/06/09 01:10:57 INFO mapred.JobClient:  map 77% reduce 22%
15/06/09 01:11:00 INFO mapred.JobClient:  map 84% reduce 22%
15/06/09 01:11:03 INFO mapred.JobClient:  map 91% reduce 22%
15/06/09 01:11:06 INFO mapred.JobClient:  map 98% reduce 22%
15/06/09 01:11:07 INFO mapred.JobClient:  map 100% reduce 22%
15/06/09 01:11:09 INFO mapred.JobClient:  map 100% reduce 100%
15/06/09 01:11:13 INFO mapred.JobClient: Job complete: job_201506090020_0006
15/06/09 01:11:13 INFO mapred.JobClient: Counters: 32
15/06/09 01:11:13 INFO mapred.JobClient:   File System Counters
15/06/09 01:11:13 INFO mapred.JobClient:     FILE: Number of bytes read=701076
15/06/09 01:11:13 INFO mapred.JobClient:     FILE: Number of bytes written=1389602
15/06/09 01:11:13 INFO mapred.JobClient:     FILE: Number of read operations=0
15/06/09 01:11:13 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/06/09 01:11:13 INFO mapred.JobClient:     FILE: Number of write operations=0
15/06/09 01:11:13 INFO mapred.JobClient:     HDFS: Number of bytes read=203708543
15/06/09 01:11:13 INFO mapred.JobClient:     HDFS: Number of bytes written=3715
15/06/09 01:11:13 INFO mapred.JobClient:     HDFS: Number of read operations=6
15/06/09 01:11:13 INFO mapred.JobClient:     HDFS: Number of large read operations=0
```

1

```
15/06/09 01:11:13 INFO mapred.JobClient:   Job Counters
15/06/09 01:11:13 INFO mapred.JobClient:     Launched map tasks=3
15/06/09 01:11:13 INFO mapred.JobClient:     Launched reduce tasks=1
15/06/09 01:11:13 INFO mapred.JobClient:     Data-local map tasks=3
15/06/09 01:11:13 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=12
1691
15/06/09 01:11:13 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots (ms)
=28773
15/06/09 01:11:13 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving s
lots (ms)=0
15/06/09 01:11:13 INFO mapred.JobClient:     Total time spent by all reduces waiting after reservin
g slots (ms)=0
15/06/09 01:11:13 INFO mapred.JobClient:   Map-Reduce Framework
15/06/09 01:11:13 INFO mapred.JobClient:     Map input records=2100000
15/06/09 01:11:13 INFO mapred.JobClient:     Map output records=28600000
15/06/09 01:11:13 INFO mapred.JobClient:     Map output bytes=314100000
15/06/09 01:11:13 INFO mapred.JobClient:     Input split bytes=351
15/06/09 01:11:13 INFO mapred.JobClient:     Combine input records=28629150
15/06/09 01:11:13 INFO mapred.JobClient:     Combine output records=29945
15/06/09 01:11:13 INFO mapred.JobClient:     Reduce input groups=265
15/06/09 01:11:13 INFO mapred.JobClient:     Reduce shuffle bytes=10368
15/06/09 01:11:13 INFO mapred.JobClient:     Reduce input records=795
15/06/09 01:11:13 INFO mapred.JobClient:     Reduce output records=265
15/06/09 01:11:13 INFO mapred.JobClient:     Spilled Records=54590
15/06/09 01:11:13 INFO mapred.JobClient:     CPU time spent (ms)=50460
15/06/09 01:11:13 INFO mapred.JobClient:     Physical memory (bytes) snapshot=770613248
15/06/09 01:11:13 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=2875011072
15/06/09 01:11:13 INFO mapred.JobClient:     Total committed heap usage (bytes)=495923200
-bash-4.1$ amine djamil
```

Figure III.32 : Résultat de l'exécution du job wordcount (2 nœuds)

Le commentaire N°1indique que le datanode s2 (IP=192.168.8.157) est arrêté.

On peut voir les résultats du job wordcount dans le fichier **part-r-00000** via l'interface graphique du HDFS ou avec la ligne de commande suivante (le résultat est en**AnnexeD**).

```
hdfs dfs -cat /user/hdfs/output1/part-r-00000
```

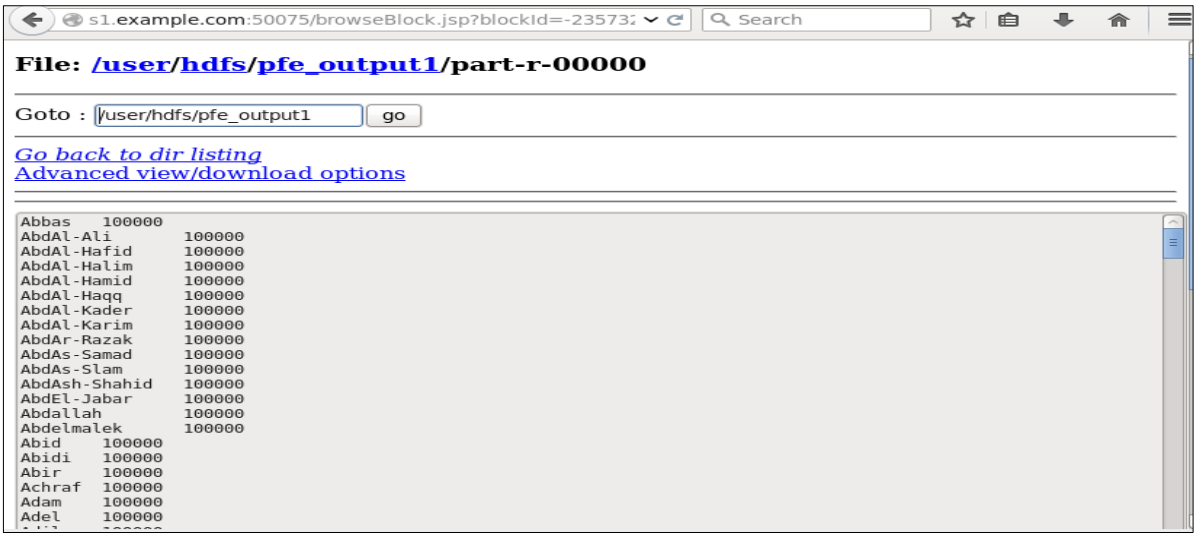

Figure III.33 : Fichier part-r-00000.txt

- **Interprétation des résultats**

Après exécution des tests précédents, on peut synthétiser les résultats dans le tableau suivant :

	3 nœuds (m1, s1, s2)	**2 nœuds (m1 et s1)**
Temps d'exécution	59 Sec	1 Mn 25 Sec
Input	194.27 Mo (203708543 octet)	194.27Mo (203708543 octet)
Output	3.63 Ko (265 records)	3.63 Ko (265 records)
CPU	55 Sec	50 Sec
Nombres des tâches (map et reduce)	4 (3 map et 1 reduce)	4 (3 map et 1 reduce)

Tableau III.1 : Résultat de l'exécution de wordcount.jar

Après lecture des résultats obtenus, on peut remarquer que le critère de performance principal dans les applications Web à savoir le temps de réponse, est en faveur de l'architecture à 3 nœuds. L'utilisation CPU a été distinguée comme un facteur limitant durant toutes nos expérimentations, ça peut être expliqué par le fait que le système à 3 nœuds applique un fort parallélisme en employant tous les cœurs du CPU, contrairement à S1, puisque c'est un temps cumulé de tous les cœurs de la machine.

Les résultats seraient différents, si on travaillait sur des machines physiques plus performantes à mémoire et CPU non partagés.

Un autre renseignement peut être tiré des résultats précédents, est le fait que des résultats obtenus en output ont été conservés malgré la défaillance d'un nœud, ce qui implique une forte tolérance aux pannes, dû principalement au principe de réplication des données, considéré comme un principe de base et un atout des systèmes distribués tel qu'**H**adoop **D**istributed **F**ile **S**ystem.

III.6.2 Exécution du job : Pi.jar

Pi est un programme qui utilise une méthode statistique (quasi-Monte Carlo) pour estimer la valeur de Pi à partir d'un ensemble de points géométriques, Il ne requiert pas de fichiers depuis le système de fichiers HDFS.

Pour lancer le job Pi il faut lui donner le nombre de Map à traiter (arg. 1) et le nombre d'échantillons par Map (arg. 2).

Hadoop jar /usr/lib/Hadoop-0.20-mapreduce/Hadoop-examples.jar Pi <arg.1><arg.2>

On a commencé à tester le job Pi en utilisant deux machines master et slave sur des données différentes puis on a refait les mêmes tests en ajoutant le deuxième slave.

1. **Cas : nbre map (4), nbre d'échantillons par Map (10^3)**
   - **Exécution du job en mode distribué avec 2 nœuds (master et slave)**

```
-bash-4.1$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 4 1000
Number of Maps  = 4
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Starting Job
15/06/09 08:46:21 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications s
hould implement Tool for the same.
15/06/09 08:46:21 INFO mapred.FileInputFormat: Total input paths to process : 4
15/06/09 08:46:21 INFO mapred.JobClient: Running job: job_201506090840_0003
15/06/09 08:46:22 INFO mapred.JobClient:  map 0% reduce 0%
15/06/09 08:46:36 INFO mapred.JobClient:  map 50% reduce 0%
15/06/09 08:46:49 INFO mapred.JobClient:  map 100% reduce 0%
15/06/09 08:46:51 INFO mapred.JobClient:  map 100% reduce 100%
15/06/09 08:46:55 INFO mapred.JobClient: Job complete: job_201506090840_0003
15/06/09 08:46:55 INFO mapred.JobClient: Counters: 33
15/06/09 08:46:55 INFO mapred.JobClient:   File System Counters
15/06/09 08:46:55 INFO mapred.JobClient:     FILE: Number of bytes read=94
15/06/09 08:46:55 INFO mapred.JobClient:     FILE: Number of bytes written=846545
15/06/09 08:46:55 INFO mapred.JobClient:     FILE: Number of read operations=0
15/06/09 08:46:55 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/06/09 08:46:55 INFO mapred.JobClient:     FILE: Number of write operations=0
15/06/09 08:46:55 INFO mapred.JobClient:     HDFS: Number of bytes read=976
15/06/09 08:46:55 INFO mapred.JobClient:     HDFS: Number of bytes written=215
15/06/09 08:46:55 INFO mapred.JobClient:     HDFS: Number of read operations=13
15/06/09 08:46:55 INFO mapred.JobClient:     HDFS: Number of large read operations=0
15/06/09 08:46:55 INFO mapred.JobClient:     HDFS: Number of write operations=3
15/06/09 08:46:55 INFO mapred.JobClient:   Job Counters
15/06/09 08:46:55 INFO mapred.JobClient:     Launched map tasks=4
15/06/09 08:46:55 INFO mapred.JobClient:     Launched reduce tasks=1
15/06/09 08:46:55 INFO mapred.JobClient:     Data-local map tasks=4
15/06/09 08:46:55 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms)=50155
```

```
15/06/09 08:46:55 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots (ms)=15582
15/06/09 08:46:55 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms
)=0
15/06/09 08:46:55 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots
(ms)=0
15/06/09 08:46:55 INFO mapred.JobClient:   Map-Reduce Framework
15/06/09 08:46:55 INFO mapred.JobClient:     Map input records=4
15/06/09 08:46:55 INFO mapred.JobClient:     Map output records=8
15/06/09 08:46:55 INFO mapred.JobClient:     Map output bytes=72
15/06/09 08:46:55 INFO mapred.JobClient:     Input split bytes=504
15/06/09 08:46:55 INFO mapred.JobClient:     Combine input records=0
15/06/09 08:46:55 INFO mapred.JobClient:     Combine output records=0
15/06/09 08:46:55 INFO mapred.JobClient:     Reduce input groups=2
15/06/09 08:46:55 INFO mapred.JobClient:     Reduce shuffle bytes=112
15/06/09 08:46:55 INFO mapred.JobClient:     Reduce input records=8
15/06/09 08:46:55 INFO mapred.JobClient:     Reduce output records=0
15/06/09 08:46:55 INFO mapred.JobClient:     Spilled Records=16
15/06/09 08:46:55 INFO mapred.JobClient:     CPU time spent (ms)=2940
15/06/09 08:46:55 INFO mapred.JobClient:     Physical memory (bytes) snapshot=828399616
15/06/09 08:46:55 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=3584286720
15/06/09 08:46:55 INFO mapred.JobClient:     Total committed heap usage (bytes)=656031744
15/06/09 08:46:55 INFO mapred.JobClient:   org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter
15/06/09 08:46:55 INFO mapred.JobClient:     BYTES_READ=96
Job Finished in 33.9 seconds
Estimated value of Pi is 3.14000000000000000000
-bash-4.1$ amine djamil
```

Figure III.34 : Résultat de l'exécution du job Pi (2 nœuds)

- **Exécution en mode distribué avec 3 nœuds (master et 2 slaves)**

```
-bash-4.1$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 4 1000
Number of Maps  = 4
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Starting Job
15/06/09 08:58:03 WARN mapred.JobClient: Use GenericOptionsParser for parsing the argume
nts. Applications should implement Tool for the same.
15/06/09 08:58:03 INFO mapred.FileInputFormat: Total input paths to process : 4
15/06/09 08:58:04 INFO mapred.JobClient: Running job: job_201506090840_0009
15/06/09 08:58:05 INFO mapred.JobClient:  map 0% reduce 0%
15/06/09 08:58:25 INFO mapred.JobClient:  map 100% reduce 0%
15/06/09 08:58:31 INFO mapred.JobClient:  map 100% reduce 100%
15/06/09 08:58:34 INFO mapred.JobClient: Job complete: job_201506090840_0009
15/06/09 08:58:34 INFO mapred.JobClient: Counters: 33
15/06/09 08:58:34 INFO mapred.JobClient:   File System Counters
15/06/09 08:58:34 INFO mapred.JobClient:     FILE: Number of bytes read=94
15/06/09 08:58:34 INFO mapred.JobClient:     FILE: Number of bytes written=846545
15/06/09 08:58:34 INFO mapred.JobClient:     FILE: Number of read operations=0
15/06/09 08:58:34 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/06/09 08:58:34 INFO mapred.JobClient:     FILE: Number of write operations=0
15/06/09 08:58:34 INFO mapred.JobClient:     HDFS: Number of bytes read=976
15/06/09 08:58:34 INFO mapred.JobClient:     HDFS: Number of bytes written=215
15/06/09 08:58:34 INFO mapred.JobClient:     HDFS: Number of read operations=13
15/06/09 08:58:34 INFO mapred.JobClient:     HDFS: Number of large read operations=0
15/06/09 08:58:34 INFO mapred.JobClient:     HDFS: Number of write operations=3
15/06/09 08:58:34 INFO mapred.JobClient:   Job Counters
15/06/09 08:58:34 INFO mapred.JobClient:     Launched map tasks=4
15/06/09 08:58:34 INFO mapred.JobClient:     Launched reduce tasks=1
15/06/09 08:58:34 INFO mapred.JobClient:     Data-local map tasks=4
15/06/09 08:58:34 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms
)=51235
```

```
15/06/09 08:58:34 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=5820
15/06/09 08:58:34 INFO mapred.JobClient:     Total time spent by all maps waiting after reservi
ng slots (ms)=0
15/06/09 08:58:34 INFO mapred.JobClient:     Total time spent by all reduces waiting after rese
rving slots (ms)=0
15/06/09 08:58:34 INFO mapred.JobClient:   Map-Reduce Framework
15/06/09 08:58:34 INFO mapred.JobClient:     Map input records=4
15/06/09 08:58:34 INFO mapred.JobClient:     Map output records=8
15/06/09 08:58:34 INFO mapred.JobClient:     Map output bytes=72
15/06/09 08:58:34 INFO mapred.JobClient:     Input split bytes=504
15/06/09 08:58:34 INFO mapred.JobClient:     Combine input records=0
15/06/09 08:58:34 INFO mapred.JobClient:     Combine output records=0
15/06/09 08:58:34 INFO mapred.JobClient:     Reduce input groups=2
15/06/09 08:58:34 INFO mapred.JobClient:     Reduce shuffle bytes=112
15/06/09 08:58:34 INFO mapred.JobClient:     Reduce input records=8
15/06/09 08:58:34 INFO mapred.JobClient:     Reduce output records=0
15/06/09 08:58:34 INFO mapred.JobClient:     Spilled Records=16
15/06/09 08:58:34 INFO mapred.JobClient:     CPU time spent (ms)=3470
15/06/09 08:58:34 INFO mapred.JobClient:     Physical memory (bytes) snapshot=828530688
15/06/09 08:58:34 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=3584286720
15/06/09 08:58:34 INFO mapred.JobClient:     Total committed heap usage (bytes)=656031744
15/06/09 08:58:34 INFO mapred.JobClient:   org.apache.hadoop.mapreduce.lib.input.FileInputForma
tCounter
15/06/09 08:58:34 INFO mapred.JobClient:     BYTES_READ=96
Job Finished in 31.008 seconds
Estimated value of Pi is 3.14000000000000000000
-bash-4.1$ amine djamil
```

Figure III.35 : Résultat de l'exécution du job Pi (3 nœuds)

2. Cas : nbre map (8), nbre d'échantillons par Map (10^6)

- **Exécution en mode distribué avec 2 nœuds (master et slave)**

```
-bash-4.1$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 8 10
00000
Number of Maps  = 8
Samples per Map = 1000000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Starting Job
15/06/09 09:18:16 WARN mapred.JobClient: Use GenericOptionsParser for parsing th
e arguments. Applications should implement Tool for the same.
15/06/09 09:18:16 INFO mapred.FileInputFormat: Total input paths to process : 8
15/06/09 09:18:16 INFO mapred.JobClient: Running job: job_201506090914_0002
15/06/09 09:18:17 INFO mapred.JobClient:  map 0% reduce 0%
15/06/09 09:18:31 INFO mapred.JobClient:  map 25% reduce 0%
15/06/09 09:18:46 INFO mapred.JobClient:  map 50% reduce 0%
15/06/09 09:18:50 INFO mapred.JobClient:  map 50% reduce 17%
15/06/09 09:18:54 INFO mapred.JobClient:  map 63% reduce 17%
15/06/09 09:18:55 INFO mapred.JobClient:  map 75% reduce 17%
15/06/09 09:18:56 INFO mapred.JobClient:  map 75% reduce 25%
15/06/09 09:19:03 INFO mapred.JobClient:  map 88% reduce 25%
15/06/09 09:19:04 INFO mapred.JobClient:  map 100% reduce 25%
15/06/09 09:19:06 INFO mapred.JobClient:  map 100% reduce 100%
15/06/09 09:19:09 INFO mapred.JobClient: Job complete: job_201506090914_0002
15/06/09 09:19:09 INFO mapred.JobClient: Counters: 33
15/06/09 09:19:09 INFO mapred.JobClient:   File System Counters
15/06/09 09:19:09 INFO mapred.JobClient:     FILE: Number of bytes read=182
15/06/09 09:19:09 INFO mapred.JobClient:     FILE: Number of bytes written=1524093
15/06/09 09:19:09 INFO mapred.JobClient:     FILE: Number of read operations=0
15/06/09 09:19:09 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/06/09 09:19:09 INFO mapred.JobClient:     FILE: Number of write operations=0
```

```
15/06/09 09:19:09 INFO mapred.JobClient:     HDFS: Number of write operations=3
15/06/09 09:19:09 INFO mapred.JobClient:   Job Counters
15/06/09 09:19:09 INFO mapred.JobClient:     Launched map tasks=8
15/06/09 09:19:09 INFO mapred.JobClient:     Launched reduce tasks=1
15/06/09 09:19:09 INFO mapred.JobClient:     Data-local map tasks=8
15/06/09 09:19:09 INFO mapred.JobClient:     Total time spent by all maps in occupied slots
(ms)=86742
15/06/09 09:19:09 INFO mapred.JobClient:     Total time spent by all reduces in occupied slo
ts (ms)=33465
15/06/09 09:19:09 INFO mapred.JobClient:     Total time spent by all maps waiting after rese
rving slots (ms)=0
15/06/09 09:19:09 INFO mapred.JobClient:     Total time spent by all reduces waiting after r
eserving slots (ms)=0
15/06/09 09:19:09 INFO mapred.JobClient:   Map-Reduce Framework
15/06/09 09:19:09 INFO mapred.JobClient:     Map input records=8
15/06/09 09:19:09 INFO mapred.JobClient:     Map output records=16
15/06/09 09:19:09 INFO mapred.JobClient:     Map output bytes=144
15/06/09 09:19:09 INFO mapred.JobClient:     Input split bytes=1008
15/06/09 09:19:09 INFO mapred.JobClient:     Combine input records=0
15/06/09 09:19:09 INFO mapred.JobClient:     Combine output records=0
15/06/09 09:19:09 INFO mapred.JobClient:     Reduce input groups=2
15/06/09 09:19:09 INFO mapred.JobClient:     Reduce shuffle bytes=224
15/06/09 09:19:09 INFO mapred.JobClient:     Reduce input records=16
15/06/09 09:19:09 INFO mapred.JobClient:     Reduce output records=0
15/06/09 09:19:09 INFO mapred.JobClient:     Spilled Records=32
15/06/09 09:19:09 INFO mapred.JobClient:     CPU time spent (ms)=5460
15/06/09 09:19:09 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1572929536
15/06/09 09:19:09 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=6446964736
15/06/09 09:19:09 INFO mapred.JobClient:     Total committed heap usage (bytes)=1251639296
15/06/09 09:19:09 INFO mapred.JobClient:   org.apache.hadoop.mapreduce.lib.input.FileInputFo
rmatCounter
15/06/09 09:19:09 INFO mapred.JobClient:     BYTES_READ=192
Job Finished in 52.992 seconds
Estimated value of Pi is 3.14159800000000000000
-bash-4.1$ amine djamil
```

Figure III.36 : Résultat d'exécution du job Pi (2 nœuds)

- **Exécution du job en mode distribué avec 3 nœuds (master et 2 slaves)**

```
-bash-4.1$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar pi 8 1000000
Number of Maps  = 8
Samples per Map = 1000000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Starting Job
15/06/09 09:00:45 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Ap
plications should implement Tool for the same.
15/06/09 09:00:46 INFO mapred.FileInputFormat: Total input paths to process : 8
15/06/09 09:00:46 INFO mapred.JobClient: Running job: job_201506090840_0010
15/06/09 09:00:47 INFO mapred.JobClient:  map 0% reduce 0%
15/06/09 09:01:01 INFO mapred.JobClient:  map 38% reduce 0%
15/06/09 09:01:02 INFO mapred.JobClient:  map 50% reduce 0%
15/06/09 09:01:11 INFO mapred.JobClient:  map 75% reduce 0%
15/06/09 09:01:16 INFO mapred.JobClient:  map 100% reduce 0%
15/06/09 09:01:19 INFO mapred.JobClient:  map 100% reduce 100%
15/06/09 09:01:22 INFO mapred.JobClient: Job complete: job_201506090840_0010
15/06/09 09:01:22 INFO mapred.JobClient: Counters: 33
15/06/09 09:01:22 INFO mapred.JobClient:   File System Counters
15/06/09 09:01:22 INFO mapred.JobClient:     FILE: Number of bytes read=182
15/06/09 09:01:22 INFO mapred.JobClient:     FILE: Number of bytes written=1524093
15/06/09 09:01:22 INFO mapred.JobClient:     FILE: Number of read operations=0
15/06/09 09:01:22 INFO mapred.JobClient:     FILE: Number of large read operations=0
15/06/09 09:01:22 INFO mapred.JobClient:     FILE: Number of write operations=0
15/06/09 09:01:22 INFO mapred.JobClient:     HDFS: Number of bytes read=1952
15/06/09 09:01:22 INFO mapred.JobClient:     HDFS: Number of bytes written=215
15/06/09 09:01:22 INFO mapred.JobClient:     HDFS: Number of read operations=25
15/06/09 09:01:22 INFO mapred.JobClient:     HDFS: Number of large read operations=0
```

```
15/06/09 09:01:22 INFO mapred.JobClient:   Job Counters
15/06/09 09:01:22 INFO mapred.JobClient:     Launched map tasks=8
15/06/09 09:01:22 INFO mapred.JobClient:     Launched reduce tasks=1
15/06/09 09:01:22 INFO mapred.JobClient:     Data-local map tasks=8
15/06/09 09:01:22 INFO mapred.JobClient:     Total time spent by all maps in occupied slots (ms
)=93912
15/06/09 09:01:22 INFO mapred.JobClient:     Total time spent by all reduces in occupied slots
(ms)=17166
15/06/09 09:01:22 INFO mapred.JobClient:     Total time spent by all maps waiting after reservi
ng slots (ms)=0
15/06/09 09:01:22 INFO mapred.JobClient:     Total time spent by all reduces waiting after rese
rving slots (ms)=0
15/06/09 09:01:22 INFO mapred.JobClient:   Map-Reduce Framework
15/06/09 09:01:22 INFO mapred.JobClient:     Map input records=8
15/06/09 09:01:22 INFO mapred.JobClient:     Map output records=16
15/06/09 09:01:22 INFO mapred.JobClient:     Map output bytes=144
15/06/09 09:01:22 INFO mapred.JobClient:     Input split bytes=1008
15/06/09 09:01:22 INFO mapred.JobClient:     Combine input records=0
15/06/09 09:01:22 INFO mapred.JobClient:     Combine output records=0
15/06/09 09:01:22 INFO mapred.JobClient:     Reduce input groups=2
15/06/09 09:01:22 INFO mapred.JobClient:     Reduce shuffle bytes=224
15/06/09 09:01:22 INFO mapred.JobClient:     Reduce input records=16
15/06/09 09:01:22 INFO mapred.JobClient:     Reduce output records=0
15/06/09 09:01:22 INFO mapred.JobClient:     Spilled Records=32
15/06/09 09:01:22 INFO mapred.JobClient:     CPU time spent (ms)=5870
15/06/09 09:01:22 INFO mapred.JobClient:     Physical memory (bytes) snapshot=1568387072
15/06/09 09:01:22 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=6448558080
15/06/09 09:01:22 INFO mapred.JobClient:     Total committed heap usage (bytes)=1251639296
15/06/09 09:01:22 INFO mapred.JobClient:   org.apache.hadoop.mapreduce.lib.input.FileInputForma
tCounter
15/06/09 09:01:22 INFO mapred.JobClient:     BYTES_READ=192
Job Finished in 36.945 seconds
Estimated value of Pi is 3.14159800000000000000
-bash-4.1$ amine djamil
```

Figure III.37 : Résultat d'exécution du job Pi (3 nœuds)

- **Interprétation des résultats**

Les résultats obtenus à partir des tests précédents, sont résumés dans le tableau suivant :

	Map (4), Nb échantillons (10^3)		Map (8), Nb échantillons (10^6)	
Nombre de nœuds	3 nœuds	2 nœuds	3 nœuds	2 nœuds
Temps d'exécution	**31.01 Sec**	**33.9 Sec**	**36.95 Sec**	**52.99 Sec**
Estimation de Pi	**3.14**	**3.14**	**3.141598**	**3.141598**

Tableau III.2 : Résultat de l'exécution de pi.jar

On peut affirmer que la complexité des données influe directement sur le temps d'exécution. La différence entre les temps d'exécutions des deux architectures augmente proportionnellement par rapport à la taille des échantillons : dans le premier cas « Map (4), nbre d'échantillons (1000) » : différence dans le temps d'exécution =2.89 Sec, par contre dans le deuxième cas « Map (8), nbre d'échantillons (1000) » : différence dans le temps d'exécution =16.04 Sec et la différence continue à accroître au fur et à mesure lorsqu'on augmente le nombre d'échantillons. Cela implique l'influence de la parallélisation des traitements des grands volumes de données sur les performances du système.

La même déduction faite lors du job précédent Wordcount, est observée ici aussi : la précision et la qualité du résultat obtenue de Pi est préservée malgré la défaillance d'un nœud, ce qui confirme la forte tolérance aux pannes d'**H**adoop **D**istributed **F**ile **S**ystem.

III.7 Conclusion

Les tests et les expérimentations effectués dans ce chapitre ont amené à la conclusion des points suivants :

- L'augmentation du nombre de nœuds influe sur l'exécution des programmes MapReduce, réduit son temps d'exécution et en conséquence la **latence** est minimisée.
- La réplication des données garantit un niveau de fiabilité pour le système Hadoop en cas de défaillance d'un de ses nœuds et en conséquence assure la **disponibilité** du système.
- La flexibilité d'une telle architecture distribuée en augmentant le nombre de nœuds des clusters reflète le passage à l'échelle et assure **l'extensibilité**.

Conclusion générale

Notre projet a fait l'objet d'une étude des nouvelles technologies du Big Data, Nous nous sommes intéressés au Framework Hadoop. La partie applicative consistait à un déploiement d'Hadoop avec ses composants MapReduce et HDFS dans un environnement distribué à base de la distribution Cloudera.

Les différents systèmes de gestion des données conçus ces dernières années, se distinguent à base de trois facteurs principaux : La performance, caractérisée par la latence et le débit, l'extensibilité (élasticité ou scalabilité) et la disponibilité.

La mise en œuvre d'un cluster Hadoop avec ses composants, l'exécution de plusieurs jobs MapReduce sur des fichiers HDFS dans des architectures à plusieurs nœuds et l'évaluation des différents statistiques des opérations effectuées nous ont permis d'approuver l'impact de la distribution et le parallélisme sur l'efficacité des systèmes gérant de grands volumes de données en réduisant la latence et en garantissant et une meilleure extensibilité.

D'un autre côté, on a pu constater, que la réplication des données sur les différents nœuds du réseau garanti un certain niveau de disponibilité du système même en cas de défaillance d'un ou plusieurs nœuds du réseau.

Perspectives

Le domaine de recherche reste très ouvert aux différents travaux et différents tests dans des architectures fortement distribuées, néanmoins le futur projet qui se voit très intéressant, est la mise en œuvre d'un cluster Hadoop sur des machines physiques puissantes dans une plateforme High-Performance-Computing (HPC) pour pallier les problèmes du Hardware qu'on a rencontré. Aussi, la démonstration des points forts d'Hadoop à savoir la performance, la latence et l'extensibilité en appliquant plusieurs tests sur des données très volumineux de plusieurs téraoctets sur des centaines de nœuds voire des milliers de nœuds peut faire l'objet d'autres prochains sujets de recherche.

Références bibliographiques

Livres électroniques

[1] Olivier BENDAVID, Bi in the Cloud, 18/06/10.

[2] Big data application and archetecteure; de himanshu et soumendra mohanty.

[3] Big Data for Dummies, par Wiley Brand.

[4] Hadoop The Definitive Guide.3rd.Edition, May 2012.Edition OReilly.

Thèse

[5] Mekideche Mounir, Conception et implémentation d'un moteur de recherche à base d'une architecture Hadoop (Big Data), Avril 2015.

PDF

[6]Benjamin Renaut, Hadoop/Big Data, Université de Nice Sophia-Antipolis, 114p, 2013-2014.

[7]Bernard ESPINASSE, Introduction aux systèmes NoSQL (Not Only SQL), Ecole Polytechnique Universitaire de Marseille, 19p, Avril 2013.

[8] Charley CLAIRMONT, Introduction à HDFS Hadoop Distributed File System, HUG France SL2013, 20p, Mai 2013.

[9]Jonathan Lejeune, Hadoop : une plate-forme d'exécution de programme Map-reduce, École des Mines de Nantes, 83p, Janvier 2015.

[10]M.CORINUS, T.Derey, J.Marguerie, W.Techer, N.Vic, Rapport d'étude sur le Big Data, SRS Day, 54p, 2012.

Sites internet

[11]http://www.journaldunet.com/solutions/analytics/big-data, consulté le1/03/2015.

[12]http://webcache.googleusercontent.com/search?q=cache:2DfXDe1Z_mEJ:www.aubay.com/fileadmin/user_upload/Publications_FR/Regard_Aubay__Big_Data_Web.pdf+&cd=1&hl=ar&ct=clnk&gl=dz, consulté le2/03/2015.

[13]http://webcache.googleusercontent.com/search?q=cache:GTE9JlAU6mAJ:www.bigdataparis.com/guide/Guide_du_Big_Data_2013_2014.pdf+&cd=1&hl=ar&ct=clnk&gl=dz, consulté le3/03/2015.

[14] http://blog.ippon.fr/2013/05/14/big-data-la-jungle-des-differentes-distributions-open-source-hadoop/, consulté le15/04/2015.

[15] http://mbaron.developpez.com/tutoriels/bigdata/hadoop/introduction-hdfs-map-reduce/, consulté le 27/04/2015.

[16] http://www.redsen-consulting.com/2013/06/big-data/, consulté le 9/06/2015.

[17] http://fr.wikipedia.org/wiki/Entrep%C3%B4t_de_donn%C3%A9es,consulté le 9/04/2015.

[18] http://www.technologies-ebusiness.com/langages/les-bases-no-sql, consulté le. 6/06/2015

[19] http://fr.wikipedia.org/wiki/Hadoop, consulté le 10/04/2015.

[20] http://blog.ippon.fr/2013/05/14/big-data-la-jungle-des-differentes-distributions-open-source-hadoop/, consulté le 10/04/2015.

[21] http://www.devx.com/opensource/exploring-the-hadoop-distributed-file-system-hdfs.html, consulté le 25/04/2015.

Annexes

1. **Annexe A**

- Implémentation de la fonction map

```
public class WordCountMapper extends Mapper<LongWritable, Text,
Text, IntWritable> {
  public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
         String string = itr.nextToken();
         if (!string.equals("se")) {
            Text word = new Text();
word.set(string);
context.write(word, new IntWritable(1));} } } }
```

- Implémentation de la fonction reduce

```
Public class WordCountReducer extends Reducer<Text, IntWritable,
Text, IntWritable>{
Public void reduce(TextKey, Iterable<IntWritable>values, Context
context)
Int sum=0;
For(IntWritablecurrent:values){
Sum+=current.get();
}
Context.write(Key, new IntWritable(sum));}}
```

2. **Annexe B :** sc.sh script

```
-bash-4.1$ cat sc.sh
#!/bin/bash

OUTPUT=${2:-bigfile.txt}

ITERATE=${3:-10000}

for ((c=1; c<=$ITERATE; c++))

do

cat $1 >> $OUTPUT

done

-bash-4.1$ amine djamil
```

Annexe C

```
File  Edit  View  Search  Terminal  Help
-bash-4.1$ hadoop jar /usr/lib/hadoop-0.20-mapreduce/hadoop-examples.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the words in the input
 files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the
words in the input files.
  dbcount: An example job that count the pageview counts from a database.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino problems.
  pi: A map/reduce program that estimates Pi using monte-carlo method.
  randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.
  randomwriter: A map/reduce program that writes 10GB of random data per node.
  secondarysort: An example defining a secondary sort to the reduce.
  sleep: A job that sleeps at each map and reduce task.
  sort: A map/reduce program that sorts the data written by the random writer.
  sudoku: A sudoku solver.
  teragen: Generate data for the terasort
  terasort: Run the terasort
  teravalidate: Checking results of terasort
  wordcount: A map/reduce program that counts the words in the input files.
-bash-4.1$ amine djamil
```

Annexe D