Université Hassiba Benbouali de Chlef
Faculté des Sciences
Département d'Informatique

جامعة حسيبة بن بوعلي الشلف
كلية العلوم
قسم الإعلام الآلي

Mémoire pour l'obtention du diplôme de Master en Informatique

Une Approche Scalable pour le Traitement de Grande Quantité de Données

Encadré par :
- M. Mohamed ARIDJ

Réalisé par :
- Abdeldjalil FELLAH
- Abdelhamid BAACH

Soutenu le 13/06/2016 devant le jury :
- M. Ahmed LOUAZANI (Président)
- M. Mohamed SLIMANE
- M. Mohamed ARIDJ

Année Universitaire : 2015 / 2016

Remerciements

C'est avec l'aide d'Allah qu'a vu le jour ce présent travail.

Ensuite, il n'aurait pas pu être achevé sans le soutien, les conseils et les encouragements de certaines personnes auxquelles nous tenons ici à exprimer nos sincères remerciements.

En premier lieu, nous exprimons toute notre gratitude pour notre encadrant, Monsieur Mohamed ARIDJ pour ses précieux conseils, sa disponibilité, la confiance qu'il nous a toujours témoigné et la sollicitude dont il nous a entouré.

Nous adressons une pensée particulièrement affective à nos collègues et amis : Abdelkader BACHA, Karim MOUHOUB, Saddam Houcine LANKRI, Baghdad GHERBI, Ali CHERID, Abdelkader CHERID, Nabil HENNI MANSEUR, Said KHATIR, Elarabi SOUKEHALI, Farid MAMERI, Hamza SOUDAKI, Mohamed BENMHANI, Mohamed CHERABRAB, Azzeddine AMEUR, Abdennour KHLOUF, Said KHALDI, Mohamed SALLAM et Abdellah RIBOUH.

Nous tenons enfin à remercier tous ceux qui ont collaborés de près ou de loin à l'élaboration de ce travail.

Abdeldjalil & Abdelhamid

Résumé

Face aux exigences nouvelles des entreprises (quantité massive de données, variété de données, multitudes de sources hétérogènes de données, etc.), les technologies traditionnelles utilisées pour l'intégration, le traitement, le stockage et l'analyse de données massives atteignent désormais leurs limites. Et c'est la cause essentielle de l'apparition d'un nouveau concept qui est le Big data.

Ce travail consiste à présenter la plateforme la plus populaire dans le monde du Big Data "Hadoop" avec son système de fichier HDFS, son modèle de traitement MapReduce et son gestionnaire de ressources Yarn. Ensuite, à décrire et présenter le problème de performance d'Hadoop dans un milieu hétérogène de machines et trouver des solutions appropriées.

L'application consiste à construire un cluster de plusieurs machines et tester les solutions proposées avec une application Java Swing qui extrait des statistiques météorologiques depuis la base de données du NCDC.

Mots clés :

Big Data, Hadoop, HDFS, Yarn, MapReduce, Milieu Hétérogène

Sommaire

Introduction générale

Nous sommes actuellement dans l'ère de la production massive de données (Big Data). Les sources de données sont nombreuses : d'une part, les applications génèrent des données issues des logs, des réseaux de capteurs, des rapports de transactions, des traces de GPS, etc. et d'autre part, les individus produisent des données telles que des photographies, des vidéos, des musiques ou encore des données sur l'état de santé (rythme cardiaque, pression ou poids).

Un problème se pose alors quant au stockage et à l'analyse des données. La capacité de stockage des disques durs augmente mais le temps de lecture croît également. Il devient alors nécessaire de paralléliser les traitements en stockant sur plusieurs unités de disques durs. Toutefois, cela soulève forcément le problème de fiabilité des disques durs qui engendre la panne matérielle. La solution envisagée est la duplication des données comme le ferait un système RAID.

Hadoop est un système distribué qui répond à ces problématiques. D'une part, il propose un système de stockage distribué via son système de fichier HDFS et ce dernier offre la possibilité de stocker la donnée en la dupliquant, un cluster Hadoop n'a donc pas besoin d'être configuré avec un système RAID qui devient inutile. D'autre part, Hadoop fournit un système d'analyse des données appelé MapReduce. Ce dernier officie sur le système de fichiers HDFS pour réaliser des traitements sur des gros volumes de données.[4]

Problématiques

Hadoop présente une haute performance et scalabilité qunad il s'agit de stocker et traiter des données dans un environnement homogène, cependant il est moins adapté dans le cas d'un environnement hétérogène de machines.

Objectifs

Au vu de ces problématiques, notre étude consiste à :

— Présenter Hadoop comme une solution scalable pour le problème de stockage et de traitement de grande quantité de données.
— Trouver des solutions pour le problème de la scalabilié d'Hadoop dans le cas des milieux hétérogènes.

Chapitre 1

Big Data

1.1 Introduction

De nos jours, l'humanité produit chaque année un volume d'informations numériques de l'ordre du Zettaoctet. Soit presque autant d'octets qu'il existe d'étoiles dans l'Univers ! En 2010 déjà, Eric Schmidt, le patron de Google, estimait que « tous les deux jours, nous produisons autant d'informations que nous en avons générées depuis l'aube de la civilisation jusqu'en 2003 ». Des chiffres et des phrases chocs sur le sujet qui agite les mondes scientifique et économique aujourd'hui : les très grands volumes de données. Les chiffres parlent d'eux-mêmes. Chaque seconde, plus d'une heure de vidéo est mise en ligne sur YouTube, et plus de 1,5 million d'e-mails sont envoyés. Nous somme entré dans l'ère du Big Data.[7]

1.2 Définitions

Big Data littéralement les « grosses données », « mégadonnées » ou encore « données massives » c'est l'appellation couramment utilisée par les spécialistes pour évoquer les grandes masses de données numériques. Il désigne 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 ou de gestion de l'information.

Le Big Data se caractérise par trois dimensions (3V) qui sont le Volume, la Variété et la Vélocité, certains auteurs ont rajoutés d'autres V comme la Valeur.

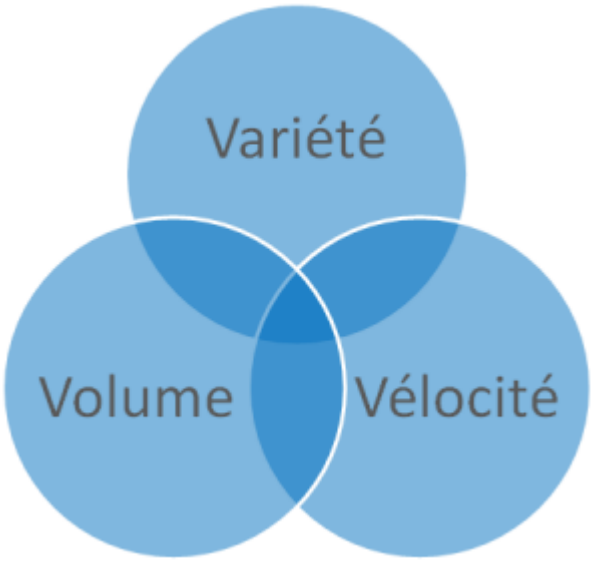


FIGURE 1.1 – 3V du Big Data

1. Volume : désigne la masse de données collectées (giga-octets, téraoctets, ...)
2. Variété : désigne l'origine variée des sources de données qui sont soit structurées ou non structurées (images, mails, tweets, données de géo-localisation,...),
3. Vélocité : représente à la fois la fréquence à laquelle les données sont générées, capturées et partagées et mises à jour.[2]

Scalabilité c'est le terme utilisé pour définir l'aptitude d'un système à maintenir un même niveau de performance face à l'augmentation de charge ou de volumétrie de données par augmentation des ressources matérielles.[16]

1.3 Architecture des systèmes Big Data

On distingue principalement les couches suivantes :

— Couche d'infrastructure : les serveurs (physique et virtuel).
— Couche de stockage : les données seront stockées soit dans une base NoSQL, ou bien directement dans le système de fichier distribué ou les Datawarehouse.
— Couche du traitement : on trouve dans cette couche les outils de traitement et analyse des données comme MapReduce et Pig. [5]

1.4 Technologies et Solutions

Pour répondre aux problématiques de la volumétrie de données, le monde bu Big Data propose plusieurs solutions et technologies :

1.4.1 Les systèmes des fichiers distribuées (DFS)

Les données ne sont plus stockées sur une seule machine car la quantité à stocker est beaucoup trop importante. Les données, les fichiers sont "découpés" en morceaux d'une taille définie et chaque morceau est envoyé sur une machine bien précise utilisant du stockage local. Le stockage local est préféré au stockage SAN/NAS pour des raisons de goulots d'étranglement au niveau du réseau et des interfaces réseaux des SAN. De plus, utiliser un stockage de type SAN coûte bien plus cher pour des performances bien moindres. Dans les systèmes de stockage distribué pour le Big Data, l'on introduit le principe de "Data locality". Les données sont sauvegardées là où elles peuvent être traitées.

Exemple : *Hadoop Distributed File System*, *Google File System*

1.4.2 Les bases de données NoSQL (Not only SQL)

Grâce à leur flexibilité et leur souplesse, ces bases non relationnelles permettent en effet de gérer de gros volumes de données hétérogènes sur un ensemble de serveurs de stockage distribués, avec une capacité de montée en charge très élevée. Elles peuvent aussi fournir des accès de paires clé-valeur en mémoire avec une très grande célérité. Réservées jusqu'à peu à une minorité, elles tendent aujourd'hui à se poser en complément du modèle relationnel qui dominait le marché depuis plus de 30 ans. [6]

Exemples : *BigTable* (Google), *Dynamo* (Amazon), *Voldemort* (LinkedIn), *Cassandra et HBase* (Facebook), MongoDB, CouchDB

1.4.3 Traitement parallélisé

Les requêtes sont séparées et distribuées à des noeuds parallélisés, puis exécutées en parallèles. Les résultats sont ensuite rassemblés et récuperés. Google, Teradata, Oracle ou EMC proposent de telles structures, basées sur des serveurs standards dont les configurations sont optimisées.

Exemple : Le modèle *MapReduce*.

1.4.4 Cloud Computing

Le Cloud est apparu comme un facilitateur principal du Big data, tant au niveau de l'infrastructure et les niveaux d'analyses. Le Cloud offre une gamme d'options pour l'analyse des données dans les paramètres Big Cloud public et privé. Du côté des infrastructures, Cloud IT ou bien IaaS (Infrastructure as a Service) fournit des options pour la gestion et l'accès aux données de très grands volumes ainsi que pour supporter des éléments d'infrastructure puissants pour des calculs complexes à un coût relativement faible.

Dans l'aspect d'analyse, les solutions Cloud consistent à louer des services Big data en SaaS (Software as a Service) auprès de fournisseurs tiers comme les réseaux

sociaux publics dans la mesure où ceux-ci seront disposés à monnayer les données dont ils disposent. Un bon nombre de problèmes commerciaux de données massives portent sur des données en ligne telles que les cliques sur la publicité et les commentaires des consommateurs qui sont utiles pour le marketing, ce qui les rend particulièrement adaptés au traitement par le biais de solutions Cloud. [16]

1.5 Cas d'usage du Big Data

Le Big Data couvre de nombreux domaines d'applications telles que l'industrie, la distribution, les banques, l'assurance, le transport, loisirs et le télécom. Des exemples sont cités ci-dessous :

1.5.1 Transports :

Contrôle du trafic : exploitation de données de tous types (GPS, Radars, sondes, etc..) afin de fluidifier le trafic et d'évaluer précisément le temps de transport d'un point à un autre.

Planification des voyages : mise à disposition du citoyen de données jusque là réservées aux administrations (gagner du temps / réduire le coût).

Systèmes de transport intelligents (ITS) : les applications des NTIC (Nouvelles Technologies de l'Information et de la Communication) destinées au domaine des transports. Parmi les thématiques d'actualité exposé durant le 20ème congrès mondial des Systèmes de Transport Intelligents 3 nous citons comme exemple : les véhicules autonomes, les véhicules coopératifs et les systèmes de positionnement par satellite.

1.5.2 Santé :

Exploitation des données à des fins d'études épidémiologiques, un cas d'utilisation est l'exemple du site « Openhelth.fr » qui affiche en temps réel des informations sur la santé des Français et des cartes en rapport (épidémies, allergies...).

Exploitation des données stockées depuis des années, jamais exploitées, qui permettraient de comprendre des liens de cause à effet « legacy data ».

Suivi des patients (dossier médical du patient).

1.5.3 Économie :

Connaissance des clients, actions personnalisées et ciblées, amélioration de la satisfaction.

Accélération des temps d'analyse des données clients pour l'identification des comportements atypiques, - Ciblage marketing (ex. micro segmentation). - Analyse prédictive de l'acte d'achat.

1.5.4 Recherche :

En TALN, deux approches coexistent : les technologies « speech-to-text » (transcription automatique de discours livrés sous forme orale) et les technologies de « machine translation » (traduction automatique de discours écrits).

Dans le domaine de l'Image Processing (traitement automatique de l'image), deux secteurs émergent : l'indexation automatique de flux d'images et de fichiers vidéo, de la reconnaissance faciale et de la reconnaissance d'objets. [2]

1.6 Avantages

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.

1.7 Mise en oeuvre

La mise en oeuvre d'un projet Big Data nécessite le choix d'une méthode de stockage, d'une technologie d'exploitation et des outils d'analyse de données. Pour optimiser les temps de traitement sur les données volumineuses, une panoplie de solutions existe, certains sont en open-source et d'autres sont propriétaires.

1.8 Conclusion

Dans ce chapitre, nous avons abordé les principes des Big Data, ces caractéristiques, son fonctionnement ainsi que les différents domaines dans lesquels elles sont utilisées. On a aussi parlé des bases de données NoSQL et les différentes technologies et outils du Big Data. Dans le chapitre suivant on va parler de la solution open-source la plus populaire dans le monde du Big Data : *Hadoop*.

Chapitre 2

Hadoop et HDP

2.1 Introduction

Hadoop est un projet Apache open-source destiné à faciliter la création d'applications distribuées et échelonnables (scalables), permettant aux applications de travailler avec des milliers de noeuds et des pétaoctets de données. Il est lancé et dirigé par Yahoo. [15]

2.2 Historique

En 2004, Google a publié un article de recherche présentant son algorithme MapReduce, conçu pour réaliser des opérations analytiques à grande échelle sur un grand cluster de serveurs, et sur son système de fichier en cluster GFS. Doug Cutting, qui travaillait alors sur le développement du moteur de recherche libre Apache Lucene et butait sur les mêmes problèmes de volumétrie de données qu'avait rencontré Google, s'est alors emparé des concepts décrits dans l'article du géant de la recherche et a décidé de répliquer en open source les outils développés par Google pour ses besoins. Employé chez Yahoo, il s'est alors lancé dans le développement de ce qui est aujourd'hui le projet Apache Hadoop. Il s'inspire du doudou de son fils de 5 ans, un éléphant jaune, pour le logo ainsi que pour le nom.

2.3 Architecture

Hadoop est principalement constitué de quatre composants :

Hadoop Distributed File System (HDFS) : un système de fichiers distribués qui fournit un accès haut-débit aux données de l'application.

Hadoop YARN : un framework pour la planification des tâches et la gestion des ressources du cluster

Hadoop MapReduce : un système basé sur YARN pour le traitement parallèle des gros volumes de données.

Hadoop Common : les utilitaires communs qui supportent les autres modules d'Hadoop.

Plus concrètement l'écosystème Hadoop comprend de nombreux autres outils couvrant le stockage et la répartition des données, les traitements distribués, l'entrepôt de données, le workflow, la programmation, sans oublier la coordination de l'ensemble des composants. On parle des outils comme Hive, Pig, Hbase, Flume,...etc.

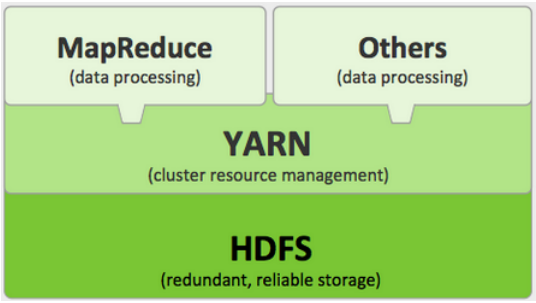


FIGURE 2.1 – Hadoop v2 : Architecture

2.4 HDFS

Le système Hadoop Distributed File (HDFS) est un système de fichiers distribué conçu pour fonctionner sur du matériel de base. Il présente de nombreuses similitudes avec les systèmes de fichiers distribués existants. Cependant, les différences par rapport à d'autres systèmes de fichiers distribués sont importants. HDFS est très tolérant aux pannes et est conçu pour être déployé sur du matériel à faible coût. HDFS fournit un accès haut débit aux données d'application et est adapté pour les applications qui ont de grands ensembles de données. HDFS détend quelques exigences de la norme POSIX pour permettre un accès en continu à déposer les données du système. HDFS a été initialement construit comme infrastructure pour le projet de moteur de recherche web Apache Nutch. HDFS est partie du projet Hadoop de base.

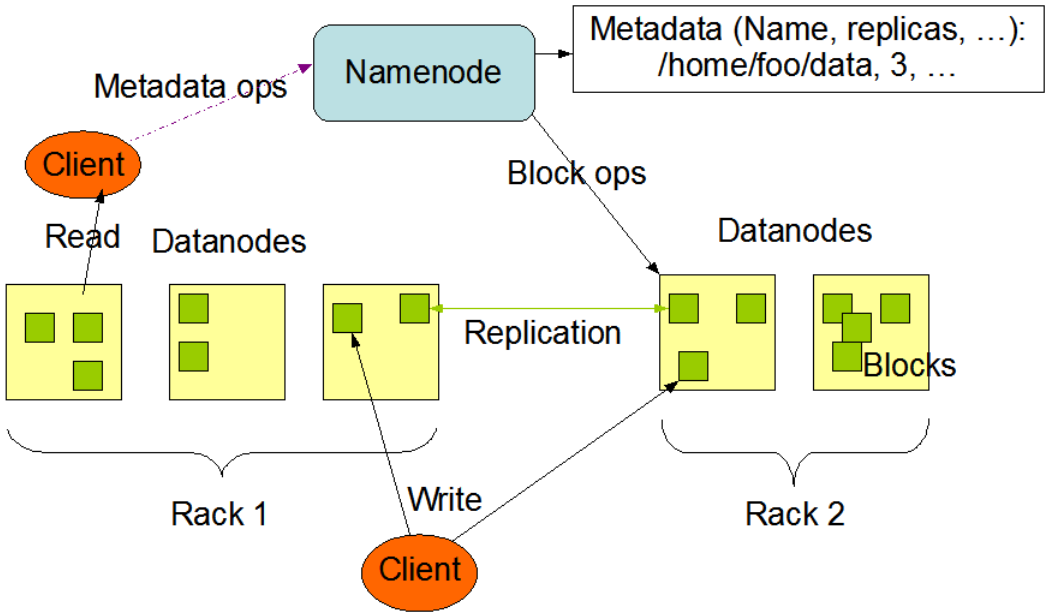


FIGURE 2.2 – HDFS : Architecture

2.4.1 Organisation des données

HDFS soutient la sémantique à écriture unique-lecture multiple sur les fichiers. Une taille de bloc typique utilisée par HDFS est de 64 MB. Ainsi, un fichier HDFS est découpé en 64 morceaux Mo, et si possible, chaque morceau sera résider sur un DataNode différente.

2.4.2 NameNode et DataNodes

HDFS a une architecture maître / esclave. Un cluster HDFS compose d'une seule NameNode, un serveur maître qui gère l'espace de noms de système de fichiers et réglemente l'accès aux fichiers par les clients. En outre, il ya un certain nombre de DataNodes, habituellement un noeud par du cluster, qui gèrent le stockage attaché aux noeuds qu'ils exécutent sur. HDFS expose un espace de noms de système de fichiers et permet aux données d'utilisateur à être stockées dans des fichiers. En interne, un fichier est divisé en un ou plusieurs blocs et ces blocs sont stockés dans un ensemble de DataNodes. Le NameNode exécute des opérations d'espace de noms de système de fichiers comme l'ouverture, la fermeture et renommer des fichiers et des répertoires. Il détermine également la mise en correspondance de blocs de DataNodes. Les DataNodes sont responsables de servir les requêtes de lecture et d'écriture des clients du système de fichiers. Les DataNodes effectuent également la création de bloc, la suppression et la réplication sur instruction du NameNode.

2.4.3 Replication des données

HDFS est conçu pour stocker de manière fiable des fichiers très grands à travers des machines dans un grand cluster. Il stocke chaque fichier comme une séquence de blocs ; tous les blocs dans un fichier à l'exception du dernier bloc ont la même taille. Les blocs d'un fichier sont répliquées pour la tolérance aux pannes. La taille de bloc et le facteur de réplication sont configurables par fichier. Une application peut spécifier le nombre de répliques d'un fichier. Le facteur de réplication peut être spécifiée au moment de la création du fichier et peut être modifié ultérieurement.

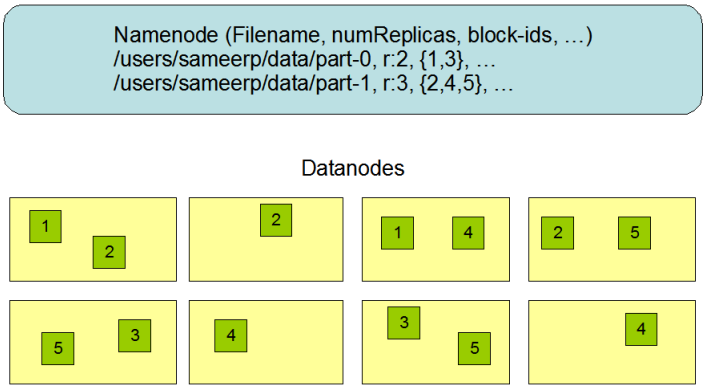


FIGURE 2.3 – HDFS : Replication des blocs

2.5 YARN

YARN (Yet Another Resource Negotiator) est un élément du projet central Hadoop. C'est un framework de nouvelle génération qui élargit les fonctionnalités de MapReduce dans le traitement des données Hadoop grâce à la prise en charge de flux de travail associés à des modèles de programmation. Il gère la planification des tâches et les ressources du cluster. [9] [3]

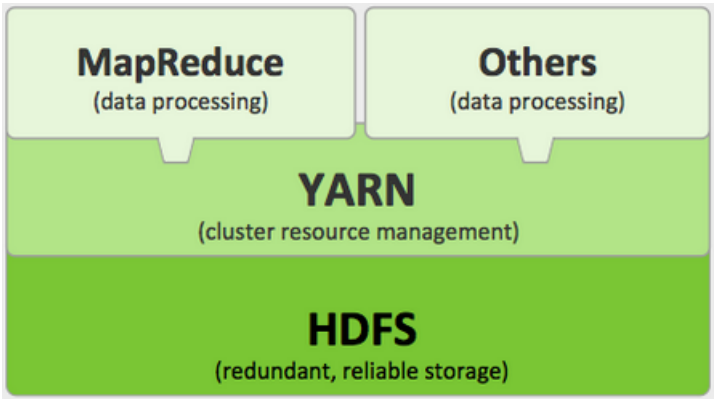


FIGURE 2.4 – Hadoop : YARN

2.6 MapReduce

MapReduce est un modèle de programmation basé sur YARN et conçu spécifiquement pour lire, traiter et écrire des volumes de données très importants. Un programme Hadoop met généralement en œuvre à la fois des tâches de type map et des tâches de type reduce. [10]

2.6.1 Principes de fonctionnement de MapReduce

Un programme Hadoop se divise généralement en trois parties :

— Le driver, qui s'exécute sur une machine client, est chargé de configurer le job puis de le soumettre pour exécution.
— Le mapper est chargé de lire les données stockées sur disque et les traiter.
— Le reducer est chargé de consolider les résultats issus du mapper puis de les écrire sur disque. [10]

Lorsque le programme d'utilisateur appelle la fonction MapReduce, la séquence suivante d'actions se produit :

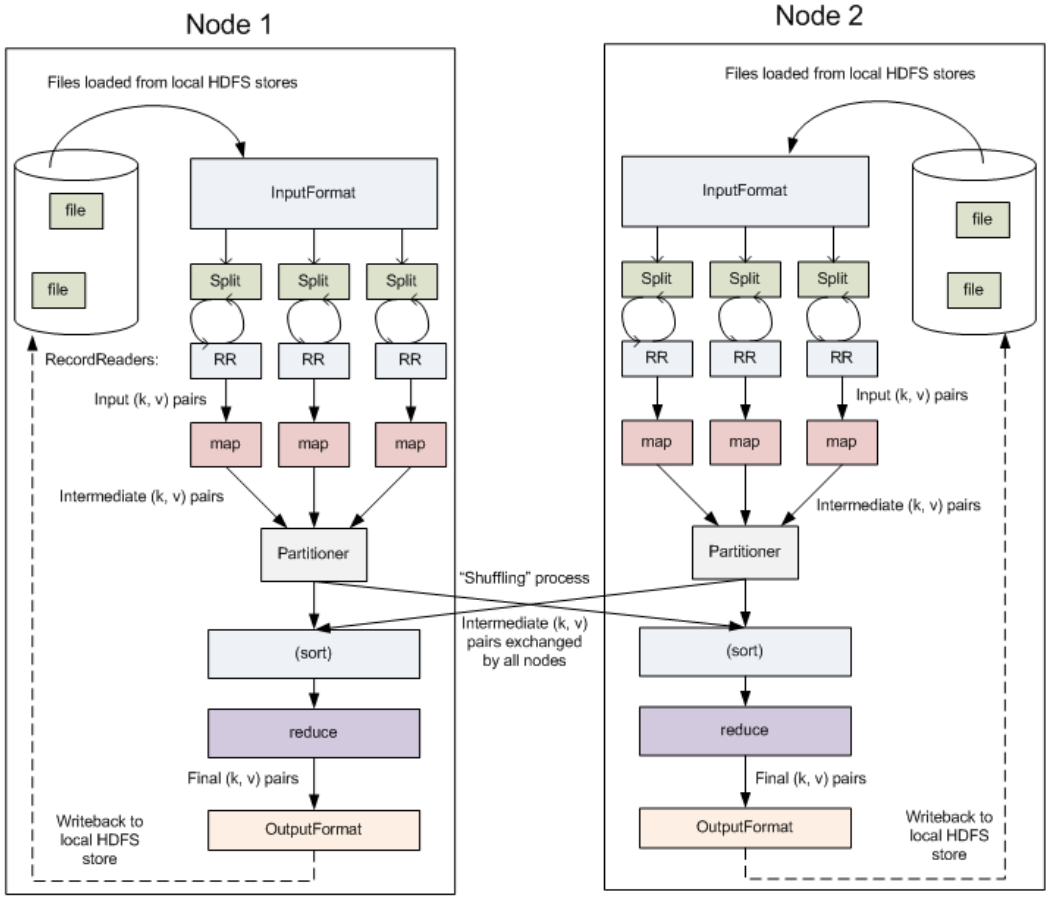


FIGURE 2.5 – MapReduce : Schéma d'exécution

1. La bibliothèque MapReduce découpe les fichiers d'entrée en M morceaux 16 Mo à 64 Mo par pièce. ensuite elle démarre plusieurs copies du programme sur

un cluster de machines.

2. Le maître affecte des tâches aux travailleurs(Workers). Il y a M tâche de « Map » et R tâches de « Reduce » à affectés. Le maître choisit les travailleurs inactifs et attribue à chacun une tâche map ou une tâche reduce.
3. Un travailleur qui est chargé d'une tâche map lit le contenu de fragment d'entrée correspondant. Il analyse les paires clé / valeur à partir des données d'entrée et passe chaque paire à la tâche map définie par l'utilisateur. Les intermédiaires paires clé / valeur produites par la tâche map sont mémorisé en buffer.
4. Périodiquement, les paires mémorisées seront écrite sur le disque local. Les emplacements de ces paires mémorisées sur le disque local sont passés vers le maître, qui est chargé de transmettre ces endroits pour les travailleurs reducer.
5. Quand un travailleur reducer est notifiée par le Maître au sujet de ces endroits, il utilise appels de procédure distante pour lire les données mémorisées dans disques locaux des mapper. Quand un travailleur reducer a lu toutes les données intermédiaires, il trie les clés intermédiaires de telle sorte que toutes les occurrences de la même clé sont regroupées. Si la quantité de données intermédiaires est trop volumineuse pour tenir dans la mémoire, un tri externe est utilisé.
6. Le travailleur reduce passe chaque clé et l'ensemble correspondant de valeurs intermédiaires à la tâche reduce de l'utilisateur. La sortie de la fonction Reduce est ajoutée à un fichier de sortie définitive.
7. Lorsque toutes les tâches « map » et « reduce » accomplies, le maître serveille le programme utilisateur. À ce point, l'appel MapReduce dans le programme utilisateur retourne a son code et la sortie de l'exécution est disponible dans les fichiers de sortie.[1][11]

2.6.2 Exemple : compteur d'occurrences de mots

Le schéma ci-dessous montre les étapes de MapReduce pour compter les occurrences des mots du texte suivant :

Mon Cher Ami
Latex Ami Latex
Mon Cher Latex

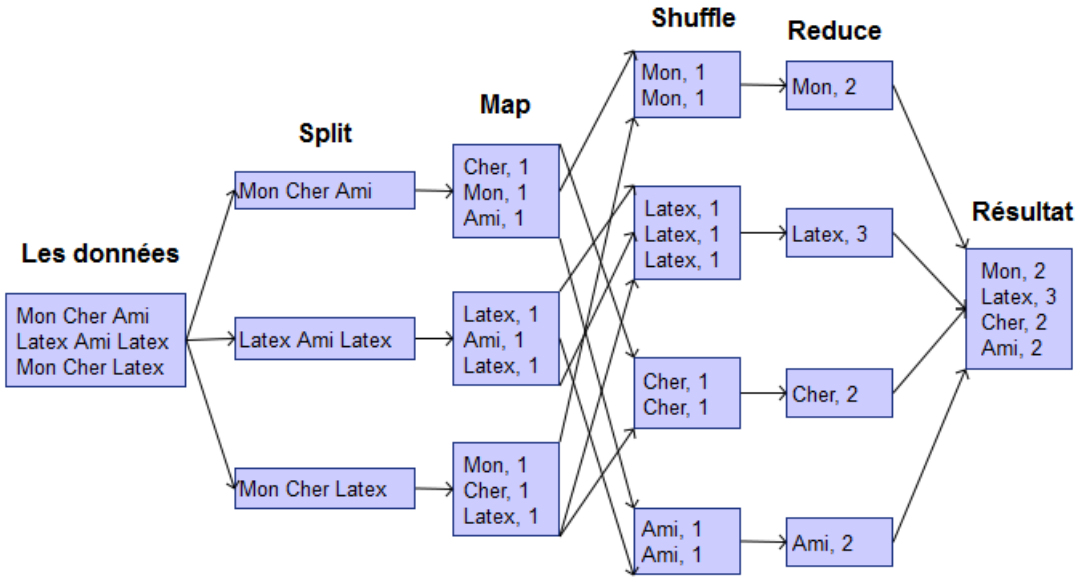


FIGURE 2.6 – MapReduce : Exemple WordCount

Split découpe les données d'entrée, de telle sorte que la fragmentation résulte en des blocs de données susceptibles d'être traités indépendamment sans influence sur le résultat final. Ici, on prend chacune des lignes du texte d'entrée comme un « bloc » de données. On considère conceptuellement ces 3 blocs de données d'entrée comme 4 couples <clef, valeur> pour lesquels la clef est nulle et la valeur est constituée de la ligne de texte. Chacun de ces blocs sera passé en entrée de la fonction Map.

Map génère comme clef le mot lui-même et comme valeur correspondante, la constante « 1 ». Le rôle de la fonction map sera donc de parcourir la ligne d'entrée (bloc), et pour chaque mot distinct de générer un couple <mot, 1>.

Shuffle les couples <mot, 1> ainsi générés seront regroupés (par le framework d'exécution du programme sur le cluster, par exemple Hadoop) par clef distincte. Chacun de ces groupes distincts sera passé en entrée de la fonction Reduce.

Reduce reçoit un groupe de couples <mot, 1> en entrée : ceux qui correspondent à une des clefs distinctes. Son rôle va simplement être de conserver la clef unique, d'additionner les valeurs de tous les couples reçus en entrée, et générer un unique couple <mot, somme> en sortie, composé de la clef unique et du total obtenu.

2.7 Distributions

Apache Hadoop est très facile à installer et l'installation se fait de manière autonome sur un système local : il suffit de dézipper, puis de mettre à jour certaines variables d'environnement et de commencer à tester WordCount par exemple. Cependant, pour l'installer sur plusieurs noeuds et l'utiliser dans des vrais projets, son administration devient plus complexe. Une distribution Hadoop résout cette complexité.

Les fournisseurs des distribution offrent des packages, des outils et le support technique, ce qui réduit beaucoup les efforts à mettre en oeuvre, pas seulement pour le développement mais aussi pour l'opérationnel. Une distribution contient différents projets de l'écosystème Hadoop. Ceci assure que toutes les versions utilisées fonctionnent ensemble sans problèmes. Il y a des releases régulières avec des versions mises à jour de différents projets. En plus du package, les fournisseurs de distribution offrent des outils graphiques pour le déploiement, l'administration et le monitoring des clusters Hadoop. De cette façon, il est beaucoup plus facile d'installer, de gérer et de surveiller les clusters.

En plus de Apache Hadoop, il y a plus ou moins trois fournisseurs de distributions Hadoop, à savoir HortonWorks, Cloudera et MapR. Bien que dans le même temps, d'autres distributions Hadoop voient aussi le jour. Par exemple : IBM InfoSphere BigInSights.

2.7.1 Cloudera

Cloudera se veut comme la compagnie commerciale d'Hadoop. Elle est fondée par des experts Hadoop en provenance de Facebook, Google, Oracle et Yahoo. Un autre avantage est de disposer dans ses rangs de Doug Cutting le créateur d'Hadoop. Si leur plate forme est en grande partie basée sur Hadoop d'Apache, elle est complétée avec des composants maison essentiellement pour la gestion du cluster. Parmi ces composants : Cloudera Manager qui sert pour le déploiement et la gestion des composants Hadoop, il n'est pas entièrement open source mais dispose d'une version gratuite avec quelques restrictions.

Les principaux partenaires sont IBM, HP, Oracle.

2.7.2 MapR

MapR a été fondée en 2009 par d'anciens membres de Google. Bien que son approche soit commerciale, MapR contribue à des projets Apache Hadoop comme HBase, Pig, Hive, ZooKeeper et surtout Drill. MapR se distingue surtout de la version d'Apache Hadoop par sa prise de distance avec le cœur de la plate-forme. Ils proposent ainsi leur propre système de fichier distribué ainsi que leur propre version de MapReduce : MapR FS et MapR MR.

C'est la distribution la plus éloignée d'Apache Hadoop car elle intègre leur propre vision de MapReduce et HDFS. Après Cloudera c'est la solution la plus mature.

2.7.3 Hortonworks

C'est la seule plateforme 100% open source basé sur Apache Hadoop. La stratégie assumée d'Hortonworks est de se baser sur les versions stables et testées d'Apache Hadoop plutôt que sur les dernières versions. Leur solution de gestion du cluster, Ambari, n'est pas aussi mature que la concurrence : Cloudera Manager et HeatMap.

HortonWorks a signé des partenariats importants avec IBM, Microsoft, Teradata et Talend. Ils ont notamment signé avec Microsoft un accord pour le déploiement de leur plate forme sur Azure. [12]

2.8 Hortonworks Data Platform

2.8.1 Présentation

HortonWorks est une société de logiciels informatique basée à Santa Clara, en Californie. La société se concentre sur le développement et le soutien d'Hadoop. Elle a été fondé en juin 2011, financé par 23 millions de dollar de Yahoo et formé par des membres de l'équipe Yahoo en charge du projet Hadoop. Leur but est de faciliter l'adoption de la plate forme Hadoop d'Apache, c'est pourquoi tous les composants sont open source et sous licence Apache. Le modèle économique d'Hortonworks est de ne pas vendre de licence mais uniquement du support et des formations.

Hortonworks Data Platform (HDP) est la plateforme de données Apache Hadoop, 100% open source et conçue pour les entreprises. Elle est conçu, développé et construit selon le principe du libre accès, fournit une plateforme prête à l'utilisation par les entreprises, leur permettant d'adopter une architecture moderne de données. HDP est construite autour de YARN, ce qui lui permet de fournir une plateforme pour le traitement de données provenant de charges multiples, en utilisant un large éventail de méthodes (des traitements par lot aux approches interactives ou en temps réel) tout en bénéficiant des fonctionnalités essentielles nécessaires à une plateforme de données d'entreprise : gouvernance, sécurité et exploitation.

2.8.2 Architecture

Hortonworks Data Platform facilite le déploiement d'Hadoop pour les entreprises en open source : en tirant parti de composants entièrement open source, en encourageant les exigences spécifiques aux entreprises et en favorisant l'adoption d'innovations tirées directement de l'Apache Software Foundation et de projets Apache majeurs. Cet ensemble complet de capacités inclut les fonctionnalités de gestion des données, d'accès aux données, de gouvernance et d'intégration des données, de sécurité et d'exploitation.

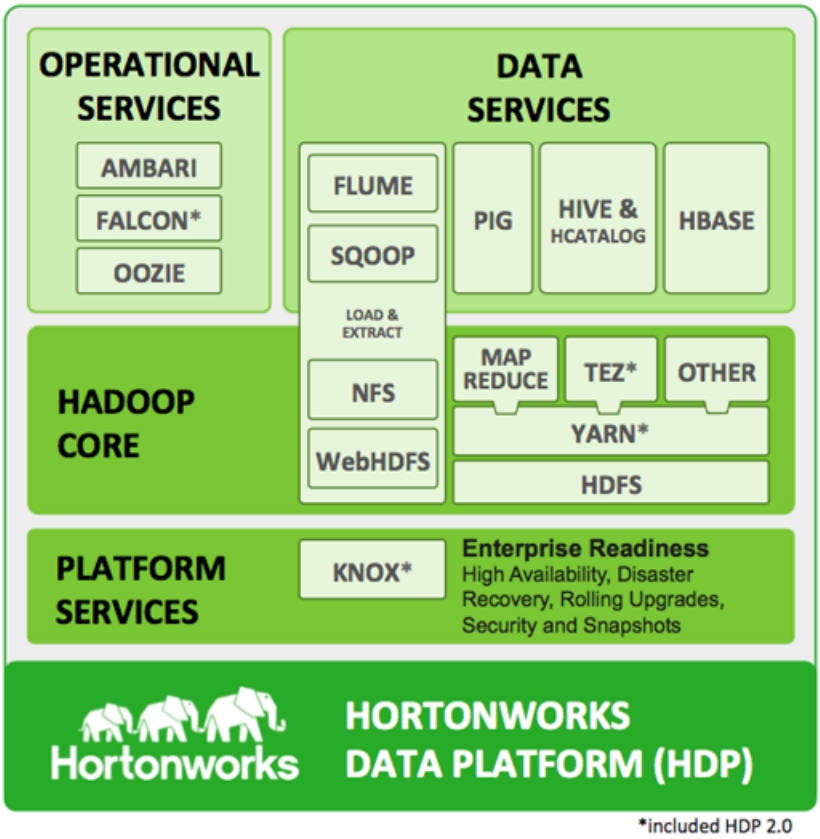


FIGURE 2.7 – HDP : Architecture

Gestion des données : HDFS et YARN

Les composants clés de HDP sont YARN et Hadoop Distributed Filesystem (HDFS). Avec YARN, au cœur de l'architecture d'Hadoop, on dispose de plusieurs modes de traitement simultané des données. Les fonctionnalités de gestion des ressources YARN ainsi que son architecture adaptable permettent de prendre en charge un grand nombre de méthodes d'accès aux données. HDFS permet de stocker les big data de façon évolutive, insensible aux défaillances et rentable.

Accès au données : Hive, Tez, Pig, Storm, Spark, HBase, Accumulo et Solr

YARN se trouve à la base d'un grand nombre de systèmes de traitement, on peut ainsi utiliser simultanément plusieurs méthodes de traitement des mêmes données. Les applications peuvent donc interagir avec les données de façon optimale : traitements par lots, SQL interactif ou accès à faible temps de latence avec NoSQL. Apache Spark, Solr et Storm prennent également en charge les nouvelles utilisations dans le cadre des opérations d'analyse, de recherche et de streaming. En outre, d'autres systèmes d'accès aux données pour YARN, encore plus spécialisés, sont disponibles auprès de notre écosystème de partenaires.

Gouvernance et intégration des données : Falcon, Oozie, Sqoop, Flume, Kafka

HDP élargit l'accès aux données ainsi que leur gestion grâce à des outils de gouvernance et d'intégration des données particulièrement efficaces. Ils fournissent un cadre fiable, simple et reproductible pour la gestion du flux de données vers et depuis Hadoop. Cette structure de contrôle ainsi que les outils de simplification et d'automatisation de l'application de schémas ou de métadonnées aux sources, sont vitales pour la réussite de l'intégration d'Hadoop dans votre architecture moderne de données.

Sécurité : Knox et Ranger

La sécurité de HDP est assurée de façon intégrée à tous les niveaux. Les fonctionnalités vitales d'authentification, d'autorisation, de responsabilité et de protection des données vous garantissent la sécurité de HDP sur l'ensemble de ces exigences clés. À l'image de l'ensemble des fonctionnalités d'Hadoop pour les entreprises, HDP permet également d'intégrer et d'élargir vos solutions de sécurité actuelles afin de protéger votre architecture de données moderne sous une ombrelle unique, cohérente et sécurisée.

Exploitation : Ambari, Zookeeper

Les équipes d'exploitation déploient, surveillent et gèrent un cluster Hadoop au sein de leur écosystème élargi de données d'entreprise. HDP fournit un ensemble complet de fonctionnalités opérationnelles qui apportent de la visibilité sur l'état du cluster ainsi que des outils de gestion de la configuration et d'optimisation des performances de l'ensemble des méthodes d'accès aux données. Apache Ambari fournit une API d'intégration avec les systèmes de gestion existants : par exemple Microsoft System Center et Teradata ViewPoint.

Intégration des applications

Hortonworks se trouve au cœur d'un écosystème florissant de fournisseurs de fonctionnalités complémentaires et/ou points d'intégration. Ces partenaires contribuent à Hadoop et l'enrichissent avec des fonctionnalités d'analyse et de veille commerciale ainsi que des outils et des infrastructures de gestion des données. De nombreux intégrateurs systèmes, petits ou grands, développent leurs compétences afin de contribuer à l'intégration et au développement de solutions.

Déploiement

HDP fournit le plus d'options de déploiement pour Hadoop : serveur Windows, Linux ou encore mode cloud. C'est la plateforme de distribution d'Hadoop la plus polyvalente : la migration d'un type de déploiement à un autre est à la fois fiable et

simple. Il fournit également des fonctions d'automatisation pour les sauvegardes vers Microsoft Azure et Amazon S3.

Hortonworks Sandbox est une application d'un noeud simple Hadoop, basée sur la Data Platform de Hortonworks. Intégrée dans une machine virtuelle, elle comprend plusieurs composants que l'on trouve dans un déploiement Hadoop, y compris le sous-système de gestion du stockage HCatalog, Hive pour le datawarehouse et Pig, un ensemble d'outils d'analyse de données.

2.9 Conclusion

Nous avons présenté dans ce chapitre les différents composants d'Hadoop, principalement, HDFS et le modèle de programmation MapReduce. On a aussi présenté la plateforme 100% open source basé sur Apache Hadoop : Hortonworks Data Platform qui simplifie et automatise le déploiement d'un cluster Hadoop.

Chapitre 3

Optimisation d'Hadoop dans un milieu hétérogène

3.1 Introduction

L'implémentation d'Hadoop considère que le traitement se réalise sur un cluster de machines homogènes (c'est-à-dire qu'elles possèdent toutes les mêmes caractéristiques matérielles). Il ne tient pas compte de la puissance des nœuds ni de la localité des données. Malheureusement, ces facteurs influencent les performances du MapReduce de manière conséquente.

En milieu homogène, tous les nœuds ont la même charge de travail, ce qui indique qu'aucune donnée ne devra être transférée d'un nœud vers un autre. En milieu hétérogène, un nœud ayant des performances élevées peut terminer son traitement local plus rapidement qu'un nœud ayant des performances plus faibles. Lorsque le nœud rapide a terminé son traitement, il devra récupérer les données non traitées d'un ou plusieurs autres nœuds plus lents. Le transfert d'une donnée d'un nœud lent vers un nœud rapide a un coût élevé.

3.2 Optimisation du stockage

Le transfert d'une large quantité de données entre les nœuds rapides et les nœuds lents influence la performance de manière significative. Donc, afin d'améliorer les performances d'Hadoop, il faut minimiser le transfert des données entre les nœuds rapides et lents. Pour cela, il faut trouver un mécanisme de placement de données qui distribue et stocke les données à travers de nombreux nœuds hétérogènes en fonction de leurs capacités. Avec cela, le transfert des données peut être réduit si le nombre de données placées dans le disque de chaque nœud est proportionnel à la vitesse de traitement des nœuds.

La réplication des données présente de nombreuses limitations. Premièrement, cela représente un coût important pour créer chaque réplique des données à l'intérieur d'un cluster ayant un nombre important de nœuds. Deuxièmement, distribuer un nombre

important de répliques peut provoquer une surcharge de la bande passante du cluster. Troisièmement, stocker les répliques requiert des disques ayant une énorme quantité de stockage.

Des chercheurs [1] se sont focalisés sur ce problème afin de produire un mécanisme de placement des données. Pour corriger le problème, ils se sont penchés sur le meilleur moyen de placer les données où les fichiers ne seront découpés et distribués sur plusieurs nœuds sans les dupliquer.

Ce mécanisme est basé sur deux algorithmes qui sont incorporés dans le HDFS d'Hadoop :

1. Le premier algorithme consiste à distribuer des fragments du fichier en entrée.
2. Le deuxième algorithme consiste à réorganiser les fragments du fichier et à corriger les erreurs qui peuvent arriver après l'exécution du premier algorithme.

3.2.1 Algorithme de distribution :

Il fonctionne comme ceci :

1. Il commence par diviser le fichier d'entrée en un nombre de fragments de même taille.
2. Puis il assigne les fragments aux nœuds du cluster en fonction de la vitesse de traitement des nœuds. (Cela aura pour conséquence qu'un nœud avec des faibles performances aura moins de fragments à traiter qu'un nœud avec des meilleures performances).

Si l'on considère une application, utilisant le MapReduce, et un fichier d'entrée dans un cluster hétérogène d'Hadoop. Le placement initial des données ne tient pas compte des performances des nœuds car Hadoop estime que tous les nœuds vont exécuter et terminer leur tâche avec, environ, le même temps. Des expérimentations, ont montré qu'en règle général, le temps de traitement de chaque nœud était stable car le temps de réponse de chaque nœud est linéairement proportionnel à la taille des données. Avec ceci, on peut quantifier la vitesse de traitement de chaque nœud dans un environnement hétérogène. Le terme pour définir la performance de chaque nœud est **Ratio Performance**.

Le Ratio Performance de chaque nœud est déterminé à l'aide de cette procédure :

1. Les opérations d'une application utilisant le MapReduce sont réalisées séparément sur chaque nœud.
2. On récupère ensuite les temps de réponse de chaque nœud
3. Le temps de réponse le plus court est utilisé comme temps pour normaliser les mesures du temps de réponse
4. Les valeurs normalisées, appelées Ratio Performance, sont utilisées par l'algorithme de placement pour distribuer les fragments de fichier aux nœuds.

1. Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares et Xiao Qin : "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters"

Exemple :

Prenons un exemple afin de montrer comment les Ratio Performance sont calculées. Prenons 3 nœuds hétérogènes A, B et C dans un cluster Hadoop. Après avoir exécuté l'application séparément sur chaque nœud, on récupère le temps de réponse de chaque nœud A, B et C (10 s, 20 s et 30 s respectivement). Le temps de réponse du nœud A est le plus court, par conséquent, le Ratio Performance de A vaut 1. Les Ratio Performance de B et C valent 2 et 3 respectivement. Cela signifie que le nœud A pourra gérer 30 fragments du fichier d'entrée tandis que le nœud C en gérera seulement 10.

Machine	**Temps de réponse**	**Ratio Performance**
A	10 s **(plus court)**	**1**
B	20 s	2
C	30 s	3

TABLE 3.1 – Amélioration de Stockage : Exemple Ratio Performance

Après l'exécution de l'algorithme de distribution des fragments pour réaliser le placement initial, les fragments peuvent être corrompus pour plusieurs raisons :

— de nouvelles données ont été ajoutées au fichier de départ,
— des blocs de données ont été supprimés du fichier,
— de nouveaux nœuds ont été ajoutés au cluster.

Pour éviter ces problèmes, un deuxième algorithme a été implémenté pour réorganiser les fragments du fichier en le basant sur le Ratio Performance des nœuds.

3.2.2 Algorithme de réorganisation :

Cet algorithme fonctionne comme ceci :

1. Les informations concernant la topologie du réseau et l'espace disque du cluster sont collectées par le serveur de distribution des données.
2. Le serveur crée deux listes de nœuds : une liste de nœuds qui contient le nombre de fragments dans chaque nœud qui peuvent être ajoutés, une liste de nœuds qui contient le nombre de fragments locals dans chaque nœud qui excédent la capacité de celui-ci.
3. Le serveur de distribution des données transfère les fragments d'un nœud, dont la capacité du disque a été dépassé, vers un nœud qui possède encore de l'espace disque.

Dans le processus de migration des données entre 2 nœuds (un surchargé et un ayant de l'espace disque), le serveur transfère les fragments du fichier depuis le nœud source de la liste des nœuds en surcharge vers un nœud ayant encore de l'espace de la liste des nœuds sous-utilisés. Ce processus est répété jusqu'à ce que le nombre de fragments dans chaque nœud soit en accord avec son Ratio Performance.

3.2.3 Résultat :

Afin de montrer les résultats de ces algorithmes de cette étude, deux tests ont été effectués, le Grep et le WordCount. Ces 2 applications tournent sur des clusters Hadoop. Grep est un outil de recherche pour une expression régulière dans un texte. WordCount est un programme utilisé pour compter les mots dans un texte.

Les caractéristiques des machines utilisés :

Machine	CPU	CPU L1 Cache
A	Intel Core 2 Duo 2 GHz	204 KB
B	Intel Celeron 2.8 GH	256 KB
C	Intel Pentium 3 1.2 GHz	256 KB
D	Intel Pentium 3 1.2 GHz	256 KB
E	Intel Pentium 3 1.2 GHz	256 KB

TABLE 3.2 – Amélioration de stockage : Machines utilisé pour l'exemple

Les Ratios obtenus :

Machine	Ratio du Grep	Ratio du WordCount
A	1	1
B	2	2
C	3.3	5
D	3.3	5
E	3.3	5

TABLE 3.3 – Amélioration de stockage : Ratios obtenus

En comparant ces configurations on obtient les résultat ci-après :

Notation	Description de la configuration
R/G	Distribution des fichiers selon les ratios du Grep
R/W	Distribution des fichiers selon les ratios du WordCount
T=	Distribution équitable entre tous les noeuds
>A	Tous les fichiers placés sur le noeud A
>B	Tous les fichiers placés sur le noeud B
>C	Tous les fichiers placés sur le noeud C

TABLE 3.4 – Amélioration de stockage : Configuration comparées

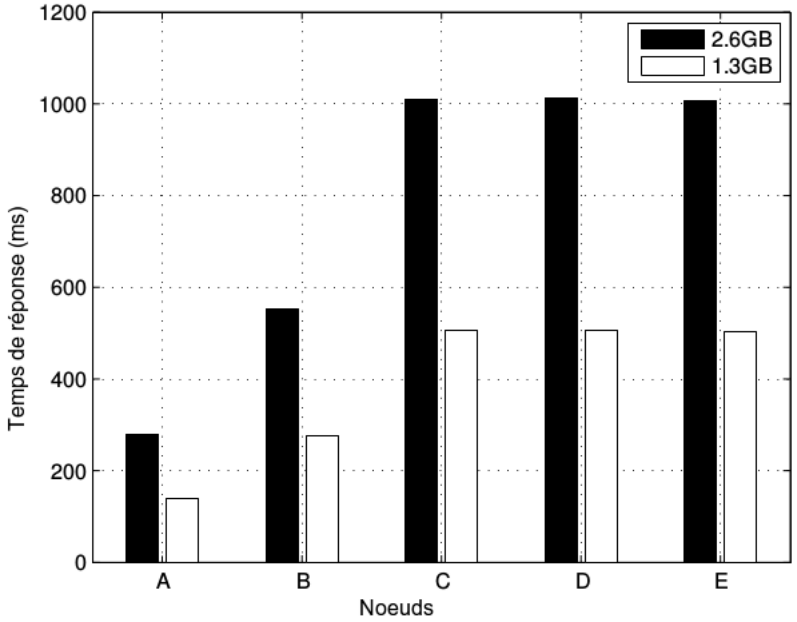


FIGURE 3.1 – Amélioration de stockage : Temps de réponse des noeuds - Grep

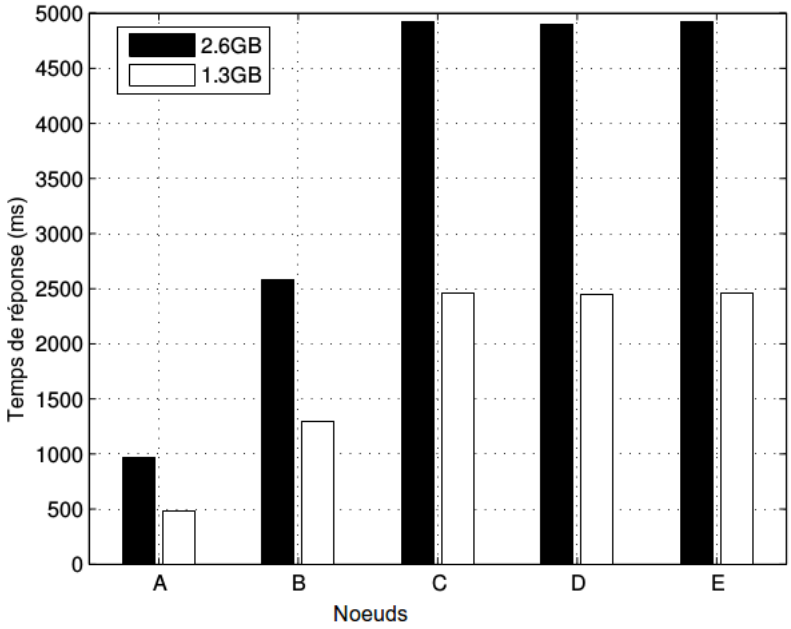


FIGURE 3.2 – Amélioration de stockage : Temps de réponse des noeuds - WordCount

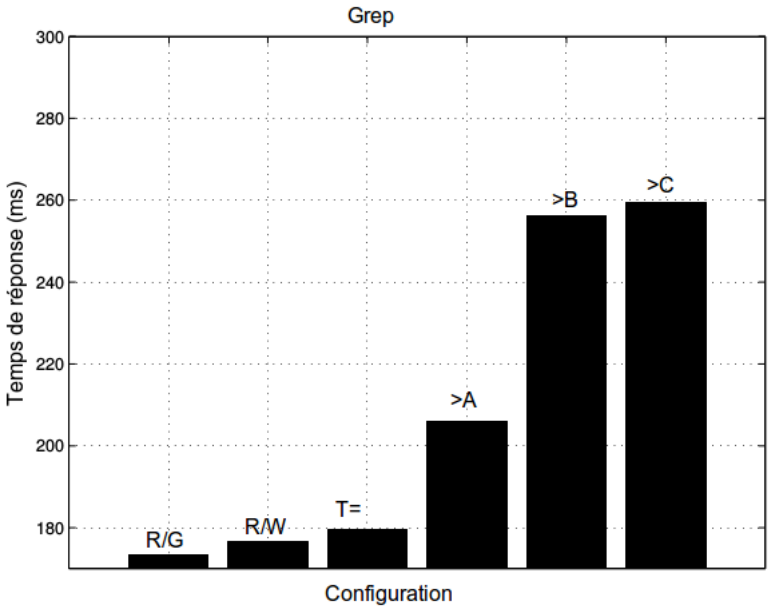


FIGURE 3.3 – Amélioration de stockage : Temps de réponse des configuration - Grep

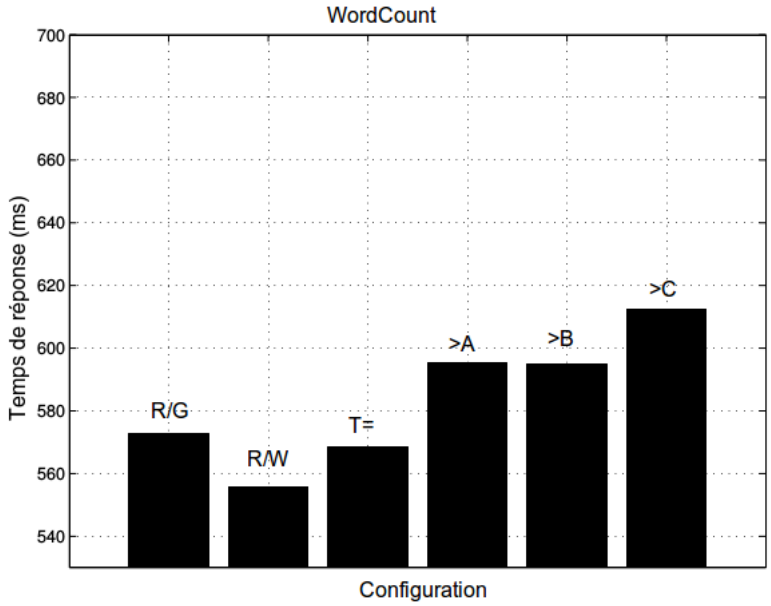


FIGURE 3.4 – Amélioration de stockage : Temps de réponse des configuration - WordCount

Les résultats ont montré que ces algorithmes ont amélioré les performances du Grep de 17% en moyenne et le WordCount de 7% en moyenne.[14]

3.3 Optimisation du traitement

Pour améliorer le traitement de MapReduce dans un milieu hétérogène de nœuds, il faut assigner aux nœuds des quantité de données d'une façon équilibrée selon ses

puissances de telle sorte que les nœuds puissants traitent plus de données que les nœuds moins puissants. Hadoop, par défaut, considère que tous les nœuds du cluster ont la même puissance, mais heureusement le gestionnaire de ressources YARN nous permet de configurer la puissance des nœuds.

3.3.1 Fonctionnement de YARN

Le fonctionnement de YARN se fait avec un système maître (**RessourceManager**) et esclaves (**NodeManager**). Le ResourceManager représente l'autorité suprême du cluster. Il est composé de deux rôles : la gestion des ressources du cluster et la gestion des applications. Il va donc gérer soumission des applications sur le cluster, et va donc assigner à chaque application des ressources d'un nœud (ce qu'on appelle un conteneur ou **Container**) qui pourra gérer l'exécution de cette application. L'exécution des applications n'est donc pas centralisé sur un seul noeud. Chaque application aura donc son **ApplicationMaster** tournant sur un noeud du cluster.

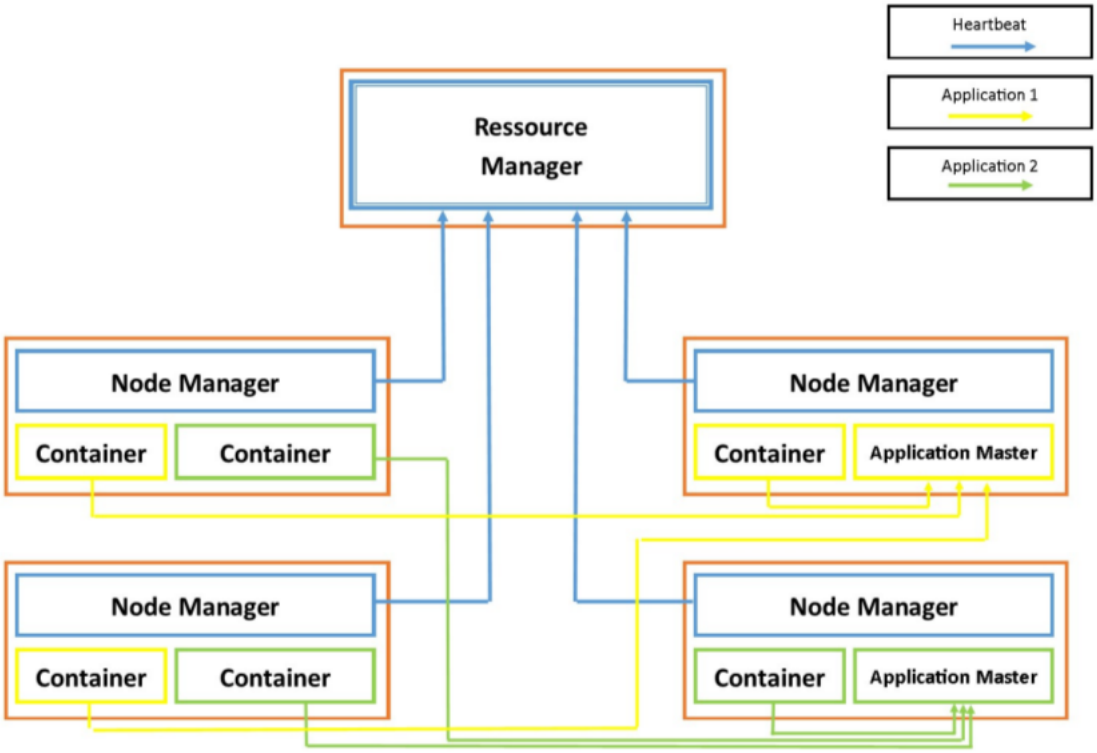


FIGURE 3.5 – Présentation de l'architecture de YARN avec 2 applications

La gestion des ressources du cluster se fait avec le **Scheduler**, qui fait parti du ResourceManager. Il va devoir assigner aux ApplicationMaster des ressources venant de nœuds suivant la demande de ces-derniers, et suivant le type d'ordonnancement. Les applications peuvent être différentes, mais elles ne sont traitées selon leurs types par le Scheduler, elles sont traitées par leurs demande en ressources sur le cluster. Le figure 3.5 schématise un exemple de distribution des ressources d'un cluster pour deux applications.

Chaque noeud du cluster est composé d'un NodeManager, qui va gérer les demandes de ressources sur ce noeud. Il va tenir le ResourceManager au courant grâce au **heartbeat**. Le heartbeat est envoyé par tous les noeuds au ResourceManager pour donner ses informations. Les ressources demandées, regroupées en conteneurs, sont des ressources d'une machine :

— La mémoire
— Le CPU

Le rôle du Scheduler du RessourceManager n'est pas de gérer tous les détails d'exécution des applications, mais bien de gérer les ressources demandées par chaque application par rapports aux noeuds disponibles. Il y a également un Scheduler pour chaque ApplicationMaster, qui va gérer les conteneurs alloués à cette application et distribuer ses propres tâches sur ces conteneurs.[8]

3.3.2 Configuration des esclaves (NodeManager) :

Fichier	Attribut	Description
yarn-site.xml	yarn.nodemanager.resource.cpu-vcores	le nombre maximal des VCORES que YARN pourra utiliser sur le noeud
yarn-site.xml	yarn.nodemanager.resource.memory-mb	la mémoire physique maximale que YARN pourra utiliser sur le noeud
yarn-site.xml	yarn.nodemanager.vmem-pmem-ratio	la proportion entre la mémoire physique et la mémoire virtuelle autorisée

TABLE 3.5 – Configuration des esclaves (NodeManager)

Remarques :

La valeur recommandée de *yarn.nodemanager.resource.cpu-vcores* est le minimum entre ces deux valeur :

— (Nbre de HDD du noeud) * 2
— (Nbre total des VCORE du noeud) - (Nbre des VCORE réservé pour autres applications que YARN)

La valeur de *yarn.nodemanager.resource.memory-mb* est calculée par :

— (Mémoire totale du noeud) - (Mémoire réservé pour autres application que YARN)

La mémoire totale autorisé est calculée par :

— yarn.nodemanager.resource.memory-mb (mémoire physique totale autorisée) * yarn.nodemanager.vmem-pmem-ratio (ratio de la mémoire virtuelle, par défaut : 2.1)

3.3.3 Configuration du maître (ResourceManager) :

Fichier	Attribut	Description
yarn-site.xml	yarn.scheduler.minimum-allocation-vcores	le nombre minimal des VCORE que le RM pourra attribuer à chaque Container demandé
yarn-site.xml	yarn.scheduler.maximum-allocation-vcores	le nombre maximal des VCORE que le RM pourra attribuer à chaque Container demandé
yarn-site.xml	yarn.scheduler.minimum-allocation-mb	La mémoire physique minimale que le RM pourra attribuer à chaque Container demandé
yarn-site.xml	yarn.scheduler.maximum-allocation-mb	La mémoire physique maximale que le RM pourra attribuer à chaque Container demandé
mapred-site.xml	mapreduce.(map|reduce).cpu.vcores	Le nombre des VCORE exigé pour chaque tâche map/reduce
mapred-site.xml	mapreduce.(map|reduce).memory.mb	La mémoire physique exigée pour chaque tâche map/reduce
mapred-site.xml	mapreduce.(map|reduce).java.opts	La mémoire Java Heap dédiée pour chaque tâche map/reduce
mapred-site.xml	yarn.app.mapreduce.am.resource.mb	La mémoire physique dédiée pour l'ApplicationMaster de MapReduce
mapred-site.xml	yarn.app.mapreduce.am.resource.cpu-vcores	Le nombre des VCORE dédiée pour l'ApplicationMaster de MapReduce

TABLE 3.6 – Configuration du maître (ResourceManager)

Remarques :

Le ResourceManager assigne les ressources à l'AM (ApplicationMaster) en trouvant le plus petit N tel que :

- Mémoire physique allouée = yarn.scheduler.minimum-allocation-mb * N
- Mémoire physique demandé <= Mémoire physique allouée
- Mémoire physique allouée <= yarn.scheduler.maximum-allocation-mb

La même chose s'applique pour le nombre des VCORE.

La mémoire dédiée aux tâches map/reduce (mapreduce.(map|reduce).memory.mb) est défini par :

- La mémoire Java Heap (mapreduce.map.java.opts) + la mémoire Java Non-Heap
- Recommandée : mapreduce.map.java.opts = 0.8 * mapreduce.map.memory.mb
- mapreduce.map.memory.mb >= yarn.scheduler.minimum-allocation-mb

3.3.4 La configuration de la mémoire recommandée par Hortonworks :

Calcul du nombre maximal des containers pour chaque nœud (#CR) :

#CR = MIN (1.8*DISKS, 2*CORES, YMR_PMEM / CR_MIN_PMEM), tel que :

- **CORES** : le nombre des cœurs physique du CPU
- **DISKS** : le nombre des disques durs
- **YMR_PMEM** : la mémoire physique recommandée pour YARN et MapReduce, voir le tableau 3.7.
- **CR_MIN_PMEM** : la mémoire physique minimale recommandée pour les containers, elle est indiquée dans le tableau, voir le tableau 3.8.

Mémoire Totale du Noeud	Reservé à l'OS	HBase et autres	YMR_PMEM
4 GB	1 GB	1 GB	2 GB
8 GB	2 GB	1 GB	5 GB
16 GB	2 GB	2 GB	12 GB

TABLE 3.7 – Hortonworks : Calcul de la mémoire physique recommandée pour YARN et MapReduce

Mémoire Totale du Noeud	CR_MIN_PMEM
inférieur à 4 GB	256 MB
entre 4 et 8 GB	512 MB
entre 8 et 24 GB	1 GB

TABLE 3.8 – Hortonworks : Calcul de la mémoire physique minimale recommandée pour les containers

Calcul des valeurs des attributs mémoire de la configuration :

CR_PMEM = MAX (CR_MIN_PMEM, YMR_PMEM / #CR) : mémoire physique pour chaque container

Attribut	Valeur
yarn.nodemanager.resource.memory-mb	#CR * CR_PMEM
yarn.scheduler.minimum-allocation-mb	CR_PMEM
yarn.scheduler.maximum-allocation-mb	#CR * CR_PMEM
mapreduce.map.memory.mb	CR_PMEM
mapreduce.reduce.memory.mb	2 * CR_PMEM
mapreduce.map.java.opts	0.8 * CR_PMEM
mapreduce.reduce.java.opts	0.8 * 2 * CR_PMEM
yarn.app.mapreduce.am.resource.mb	2 * CR_PMEM
yarn.app.mapreduce.am.command-opts	0.8 * 2 * CR_PMEM

TABLE 3.9 – Hortonworks : calcul des valeurs des attributs mémoire de la configuration

3.3.5 Configuration optimisée

Pour optimiser la configuration des noeuds selon ses puissances, on va procéder comme suit :

1. Calculer le **Facteur de Performance** des noeuds en exécutant et récupérant le **temps de réponse** d'un job MapReduce sur chaque nœud séparément.
2. Calculer le nombre optimisé des containers de chaque nœud selon son Facteur de Performance.
3. Configurer YARN en utilisant le nombre optimisé des containers au lieu de la configuration par défaut (tous les noeuds ont le même nombre de container).

Le Facteur de Performance (**RP**) de chaque noeud est calculé en appliquant la formule :

RP = Le temps de réponse le plus grand / le temps de réponse du noeud (**TR**)

Le nombre optimisé des containers **#CRO** est calculé en appliquant la formule :

#CRO = Le nombre de container par défaut (**#CR**) * Facteur de Performance

Exemple :

Machine	**TR**	**RP**	**#CR**	**#CRO**
M1	70 s	100/70 = 1.43	2	2 * 1.43 = 3
M2	70 s	100/70 = 1.43	2	2 * 1.43 = 3
M3	70 s	100/70 = 1.43	2	2 * 1.43 = 3
M4	100 s **(plus grand)**	100/100 = 1	2	2 * 1 = 2
M5	100 s	100/100 = 1	2	2 * 1 = 2

TABLE 3.10 – Amélioration du Traitement : Exemple

3.4 Optimisation de l'ordonnancement :

3.4.1 Le Scheduler :

Dans Hadoop, le Scheduler du ResourceManager a pour but d'attribuer des ressources aux applications. Donc, il doit gérer les applications qui lui sont soumises de la façon qu'il le souhaite, et leur attribuer des conteneurs (Container), suivant les demandes de ces applications. Une application doit aussi gérer les ressources qui leur sont attribuées en les partageant à des tâches qu'elle doit exécuter. Ceci peut être fait par l'utilisateur codant son application, puisque les demandes de ressources peuvent être faites à la main.

L'objectif du Scheduler est de gérer toutes les demandes d'applications en utilisant le maximum de ressource du cluster. En effet, il ne serait pas tolérable d'avoir un Scheduler n'attribuant pas les ressources du cluster alors qu'un certain nombre d'applications en attendent. Le rôle du Scheduler est donc de satisfaire au maximum les applications. Au niveau de l'ApplicationMaster, le but de l'ordonnancement est d'exécuter toutes les tâches en un minimum de temps.

Hadoop supporte trois type d'ordonnanceur :

3.4.2 FIFO :

Le FIFO Scheduler est l'ordonnanceur de base d'Hadoop. Il gère les applications dans une file First In First Out. L'application la plus prioritaire sera la plus vieille. Ce scheduler va tout de même répartir toutes les ressources du scheduler, une assignation n'étant pas bloquante. C'est le premier scheduler qui a été implémenté dans Hadoop.

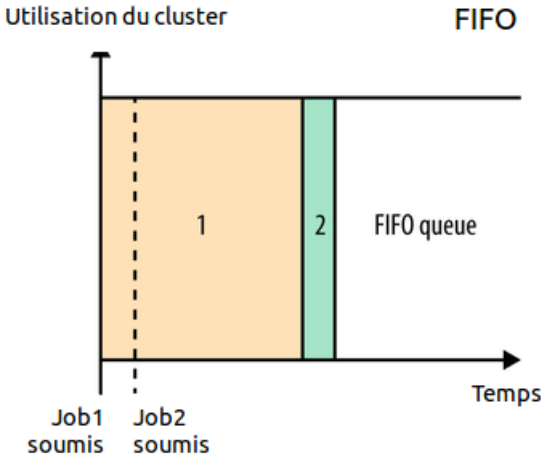


FIGURE 3.6 – Ordonnancement : FIFO

L'inconvénient d'un tel scheduler est le temps d'attente qui peut être long, voir très long dans certains cas où il y beaucoup d'applications gourmandes en tête de file. C'est pour cela que deux autres scheduler ont été implémentés dans Hadoop.

3.4.3 Capacity Scheduler :

Le Capacity Scheduler est un ordonnanceur qui va gérer les applications selon les utilisateurs qui les ont soumises. Il va répartir les applications dans des queues selon un certain critère, par défaut l'utilisateur.

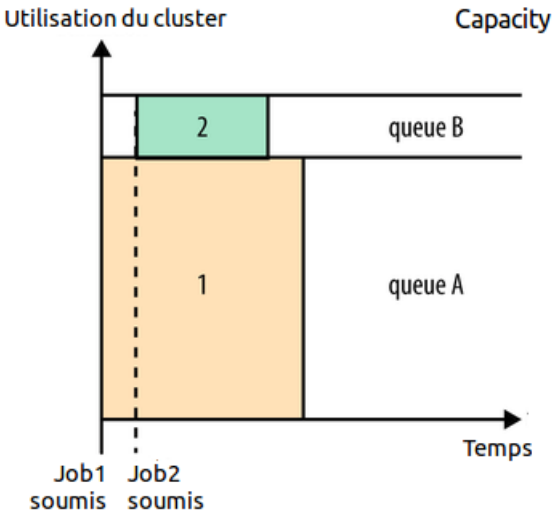


FIGURE 3.7 – Ordonnancement : Capacity

L'allocation des ressources se fait de façon équitable avec les queues. Les ressources inutilisées sont réparties aux autres queues jusqu'à ce que la queue demande de nouvelles ressources. Les queues peuvent avoir une priorité supérieure aux autres queues. L'intérêt de ce scheduler est de partager équitablement un cluster entre plusieurs entités.

3.4.4 Fair Scheduler :

Le Fair Scheduler va répartir les applications dans des pools, et va s'efforcer à partager les ressources de façon équitable. Par défaut, un pool va être créé pour chaque utilisateur, mais on peut également spécifier le pool dans lequel on veut que l'application se situe. L'assignation des ressources se fait de façon équitable.

Un système de priorité est mis en place pour les pools qui n'ont pas assez de ressources, on dit que ces pools sont en dessous de leur minimum share (part minimum). Ils sont alors prioritaires pour recevoir de nouvelles ressources disponibles. La part équitable d'un pool peut être calculée avec une priorité configurée au préalable. Cela permet également de prioriser certaines personnes ou entités soumettant des applications.

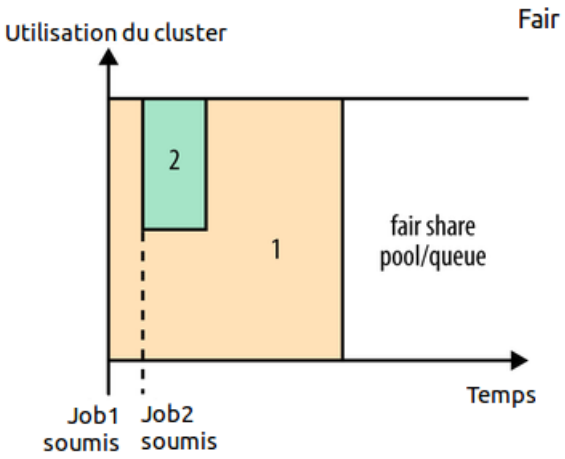


FIGURE 3.8 – Ordonnancement : Fair

Dans les pools, le partage des ressource se fait en FIFO, mais cela peut être configurable pour être également fait de façon équitable. En effet, un pool peut également contenir des pools, avec qui il pourra également partager les ressources, comme pour les pools principaux. On a donc un système d'arbre avec ces pools, et les ressources sont distribuées de façon équitable ou en FIFO selon l'endroit dans l'arbre.

Les feuilles de cet arbre sont les applications. Ce système permet de partager les ressources équitablement et d'éviter la famine de certaines applications. Cela diminue également le temps d'attente des petites applications, qui vont se voir allouer plus facilement ce qu'ils demandent. [8]

3.4.5 Ordonnanceurs améliorés :

Plusieurs schedulers ont été implémentés pour améliorer la performance de MapReduce dans les environnement hétérogènes :

— **DyScale** : un ordonnanceur des jobs MapReduce pour un système hétérogène de processeurs multicoeurs. [1]
— **ThroughputScheduler** : un ordonnanceur pour réduire le temps d'exécution des jobs MapReduce dans un milieu hétérogène. [2]
— **SAMR** : un ordonnanceur qui s'adapte dynamiquement pendant l'exécution des job MapReduce. [3]

1. "DyScale : a MapReduce Job Scheduler for Heterogeneous Multicore Processors" par Feng Yan, Ludmila Cherkasova, Zhuoyao Zhang et Evgenia Smirni
2. "ThroughputScheduler : Learning to Schedule on Heterogeneous Hadoop Clusters" par Shekhar Gupta, Christian Fritz, Bob Price, Roger Hoover et Johan de Kleer
3. "A Self-adaptive MapReduce Scheduling Algorithm in Heterogeneous Environment" par Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng and Song Guo

3.5 Conclusion

Plusieurs solutions sont proposées pour optimiser la performance d'Hadoop dans un milieu hétérogène, ces solutions touchent le stockage, le traitement et l'ordonnancement. Notre travail consistait a optimiser la performance du traitement en configurant le gestionnaire des ressources YARN d'une façon optimale.

Chapitre 4

Application et Résultats

4.1 Introduction

Dans ce chapitre on va :

— Installer Hadoop 2.7.1 sur un cluster constitué de 5 machines (Ubuntu 14.04 LTS avec LXC).
— Calculer le nombre optimal de containers pour chaque machine
— Copier des fichiers GZIP (200 KB et 6000 lignes chaqu'un) du NCDC National Climatic Data Center dans le HDFS
— Développer et exécuter une application Java (Swing) pour extraire des statistiques météorologiques (température maximale d'une station météo dans une période donnée).
— Implémenter une classe Mapper pour optimiser le traitement des petits fichiers GZIP.
— Comparer les résultats obtenus dans différentes configurations (configuration standard et configuration optimisée).

4.2 Environnement de travail

Pour appliquer et tester les améliorations sur un cluster Hadoop on a construit un environnement basé sur :

— Système d'exploitation : **Ubuntu 14.04.3 LTS 64bit**
— Système de virtualisation : **LXC 1.0.8**
— Apache Hadoop : **v2.7.1**

Machine Phys.	Noeud	Caractéristiques	Adresse IP
i5 6GB	hadoopmaster1	i5 2GB (Phys.)	10.10.10.10
	hadoopslave11	i5 2GB (LXC)	10.10.10.11
	hadoopslave12	i5 2GB (LXC)	10.10.10.12
i3 4GB	hadoopslave20	i3 2GB (Phys.)	10.10.10.20
	hadoopslave21	i3 2GB (LXC)	10.10.10.21

TABLE 4.1 – Application : Environnement de Travail

4.2.1 Présentation de LXC

LXC, contraction de l'anglais LinuX Containers est un système de virtualisation, utilisant l'isolation comme méthode de cloisonnement au niveau du système d'exploitation. Il est utilisé pour faire fonctionner des environnements Linux isolés les uns des autres dans des conteneurs partageant le même noyau et une plus ou moins grande partie du système hôte. Le conteneur apporte une virtualisation de l'environnement d'exécution (Processeur, Mémoire vive, réseau, système de fichier...) et non pas de la machine. Pour cette raison, on parle de « conteneur » et non de machine virtuelle.

Ce système est similaire aux autres systèmes de virtualisations au niveau du système d'exploitation comme openVZ. [1]

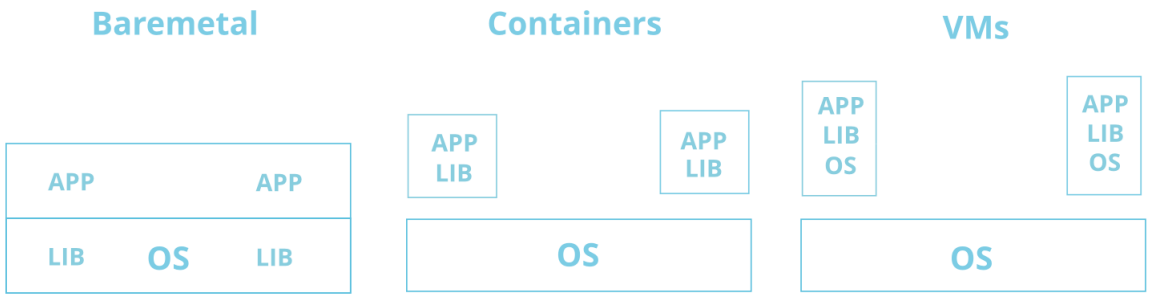


FIGURE 4.1 – Container LXC vs. Machine Virtuelle

On a utilisé LXC au lieu des machines virtuelles donc pour minimiser les ressources allouées aux containers LXC, et afin de maximiser le nombre des containers YARN créés.

4.3 Installation d'Hadoop 2.7.1

Pour construire un cluster Hadoop il faut :

— Installer JDK et configurer ses variables d'environnement (JAVA_HOME)
— Ajouter tous les noeuds au fichier /etc/hosts
— Autoriser l'accès SSH (sans mot de passe démandé) pour tous les noeud à partir hadoopmaster1

1. Installer et utiliser LXC sur CentOS

— Extraire hadoop-2.7.1.tar.gz et configurer ses variables d'environnement (HADOOP_HOME,...)
— Configurer les fichiers : core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml et slaves

4.3.1 Configuration du SSH

Pour démarrer l'ensemble des machines du cluster en une seule commande depuis le master on doit configurer le SSH de telle sorte que les slaves ne demande pas un mot de passe lors de l'accès à distance. Cette configuration consiste à exécuter les commandes suivantes :

```
1  ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
2  cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
3
4  chmod o-w ~/
5  chmod 700 ~/.ssh
6  chmod 600 ~/.ssh/authorized_keys
7
8  ssh-add
9  sudo service ssh restart
10 ssh-copy-id -i ~/.ssh/id_dsa.pub hadoopuser@hadoopslave11
```

Code 4.1 – Configuration SSH

4.3.2 Les fichiers de configuration

Voir annexe A

4.4 Démarrer le Cluster

Pour démarrer le cluster il suffit d'exécuter les commandes suivantes sur le master (hadoopmaster1) :

```
1 hdfs namenode -format
2 start-dfs.sh
3 start-yarn.sh
```

Code 4.2 – Démarrage du cluster

Pour automatiser cette opération on créé des script BASH qui démarrent le cluster en un seul clic :

```
1 sudo lxc-start -n hadoopslave11 -d
2 sudo lxc-wait -n hadoopslave11 -s RUNNING
3
4 sudo lxc-start -n hadoopslave12 -d
5 sudo lxc-wait -n hadoopslave12 -s RUNNING
6
7 sudo lxc-ls --fancy
```

```
8
9  ssh hadoopslave20 << HERE
10 sudo lxc-start -n hadoopslave21 -d ; sudo lxc-wait -n hadoopslave21 -s
       RUNNING ; sudo lxc-ls --fancy
11 HERE
```

Code 4.3 – Exemple lxc-startAll.sh

```
1  scp /data/configs/config2/* /home/hadoopuser/hadoop-2.7.1/etc/hadoop
2
3  scp /data/configs/config2/* hadoopuser@hadoopslave11:/home/hadoopuser/
       hadoop-2.7.1/etc/hadoop
4  scp /data/configs/config2/* hadoopuser@hadoopslave12:/home/hadoopuser/
       hadoop-2.7.1/etc/hadoop
5
6  stop-yarn.sh
7  stop-dfs.sh
8
9  rm -rf /home/hadoopuser/hdfs
10
11 ssh hadoopuser@hadoopslave11 << HERE
12 stop-yarn.sh ; stop-dfs.sh ; rm -rf /home/hadoopuser/hdfs
13 HERE
14
15 ssh hadoopuser@hadoopslave12 << HERE
16 stop-yarn.sh ; stop-dfs.sh ; rm -rf /home/hadoopuser/hdfs
17 HERE
18
19 hdfs dfsadmin -safemode leave
20
21 hdfs namenode -format
22
23 start-dfs.sh
24
25 sleep 30
26
27 hadoop fs -mkdir /input
28 find /data/store/ncdc/gz/ -maxdepth 1 -type f | head -n 10 | xargs -I
       {} hadoop fs -put {} /input
29
30 start-yarn.sh
31
32 sleep 30
```

Code 4.4 – Exemple run-config2.sh

4.5 Calcul du nombre optimal des containers

YARN par défaut considère que tous les noeuds peuvent allouer le même nombre de container (2 dans notre cas), pour optimiser ce nombre on doit trouver le Facteur de Performance de chaque machine en exécutant un job **Pi** sur tous les noeuds séparément, les résultats obtenu sont indiqués dans le tableau ci-dessous :

Noeud	Temps de réponse	Facteur de Performance
hadoopmaster1	43s	1.86 (80 / 43)
hadoopslave11	43s	1.86
hadoopslave12	43s	1.86
hadoopslave20	80s	1 (80 / 80)
hadoopslave21	80s	1

TABLE 4.2 – Facteur de Performance

Noeud	Facteur de Performance	Nombre de container par défaut	Nombre optimisé de container
hadoopmaster1	1.86	1	2 (1 * 1.86)
hadoopslave11	1.86	1	2
hadoopslave12	1.86	1	2
hadoopslave20	1	1	1 (1 * 1)
hadoopslave21	1	1	1

TABLE 4.3 – Nombre optimal des containers

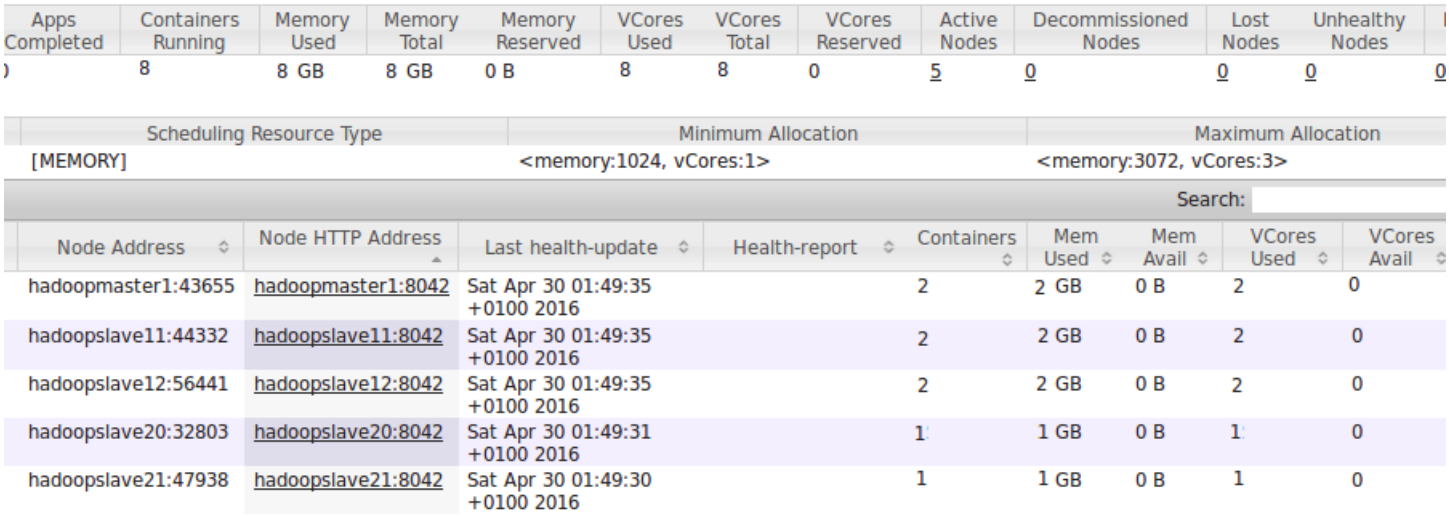
Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
0	5	5 GB	5 GB	0 B	5	5	0	5	0	0	0

Scheduling Resource Type	Minimum Allocation	Maximum Allocation
[MEMORY]	<memory:1024, vCores:1>	<memory:2048, vCores:2>

Search:

Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail
hadoopmaster1:50087	hadoopmaster1:8042	Sat Apr 30 01:45:54 +0100 2016		1	1 GB	0 B	1	0
hadoopslave11:35803	hadoopslave11:8042	Sat Apr 30 01:45:54 +0100 2016		1	1 GB	0 B	1	0
hadoopslave12:49747	hadoopslave12:8042	Sat Apr 30 01:45:54 +0100 2016		1	1 GB	0 B	1	0
hadoopslave20:57437	hadoopslave20:8042	Sat Apr 30 01:45:49 +0100 2016		1	1 GB	0 B	1	0
hadoopslave21:46945	hadoopslave21:8042	Sat Apr 30 01:45:48 +0100 2016		1	1 GB	0 B	1	0

FIGURE 4.2 – Configuration Standard



Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
)	8	8 GB	8 GB	0 B	8	8	0	5	0	0	0

Scheduling Resource Type	Minimum Allocation	Maximum Allocation
[MEMORY]	<memory:1024, vCores:1>	<memory:3072, vCores:3>

Search:

Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail
hadoopmaster1:43655	hadoopmaster1:8042	Sat Apr 30 01:49:35 +0100 2016		2	2 GB	0 B	2	0
hadoopslave11:44332	hadoopslave11:8042	Sat Apr 30 01:49:35 +0100 2016		2	2 GB	0 B	2	0
hadoopslave12:56441	hadoopslave12:8042	Sat Apr 30 01:49:35 +0100 2016		2	2 GB	0 B	2	0
hadoopslave20:32803	hadoopslave20:8042	Sat Apr 30 01:49:31 +0100 2016		1	1 GB	0 B	1	0
hadoopslave21:47938	hadoopslave21:8042	Sat Apr 30 01:49:30 +0100 2016		1	1 GB	0 B	1	0

FIGURE 4.3 – Configuration Optimisée

Pour automatiser le calcul des Facteur de Performance on a créé des scripts BASH qui nous donnent directement le temps de réponse de chaque machine en un seul clic :

```
1 scp /data/configs/hadoopslave20/* hadoopuser@hadoopslave20:/home/
    hadoopuser/hadoop-2.7.1/etc/hadoop
2
3 ssh hadoopuser@hadoopslave20 << HERE
4 stop-dfs.sh ; rm -rf /home/hadoopuser/hdfs ; hdfs dfsadmin -safemode
    leave ; hdfs namenode -format ; start-dfs.sh ; stop-yarn.sh ; start
    -yarn.sh ; sleep 30; hadoop jar /home/hadoopuser/hadoop-2.7.1/share
    /hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar pi 5 1000 |
    grep 'Job Finished in' > /home/hadoopuser/hadoop-2.7.1/
    hadoopslave20
5 HERE
6
7 scp hadoopuser@hadoopslave20:/home/hadoopuser/hadoop-2.7.1/
    hadoopslave20 /data/scripts/result/hadoopslave20
```

Code 4.5 – Exemple run-hadoopslave20.sh

4.6 Application

Pour tester ces configurations, on a développé une application (Meteoop) qui extrait la température maximale des stations météorologique depuis les fichiers GZIP du NCDC.

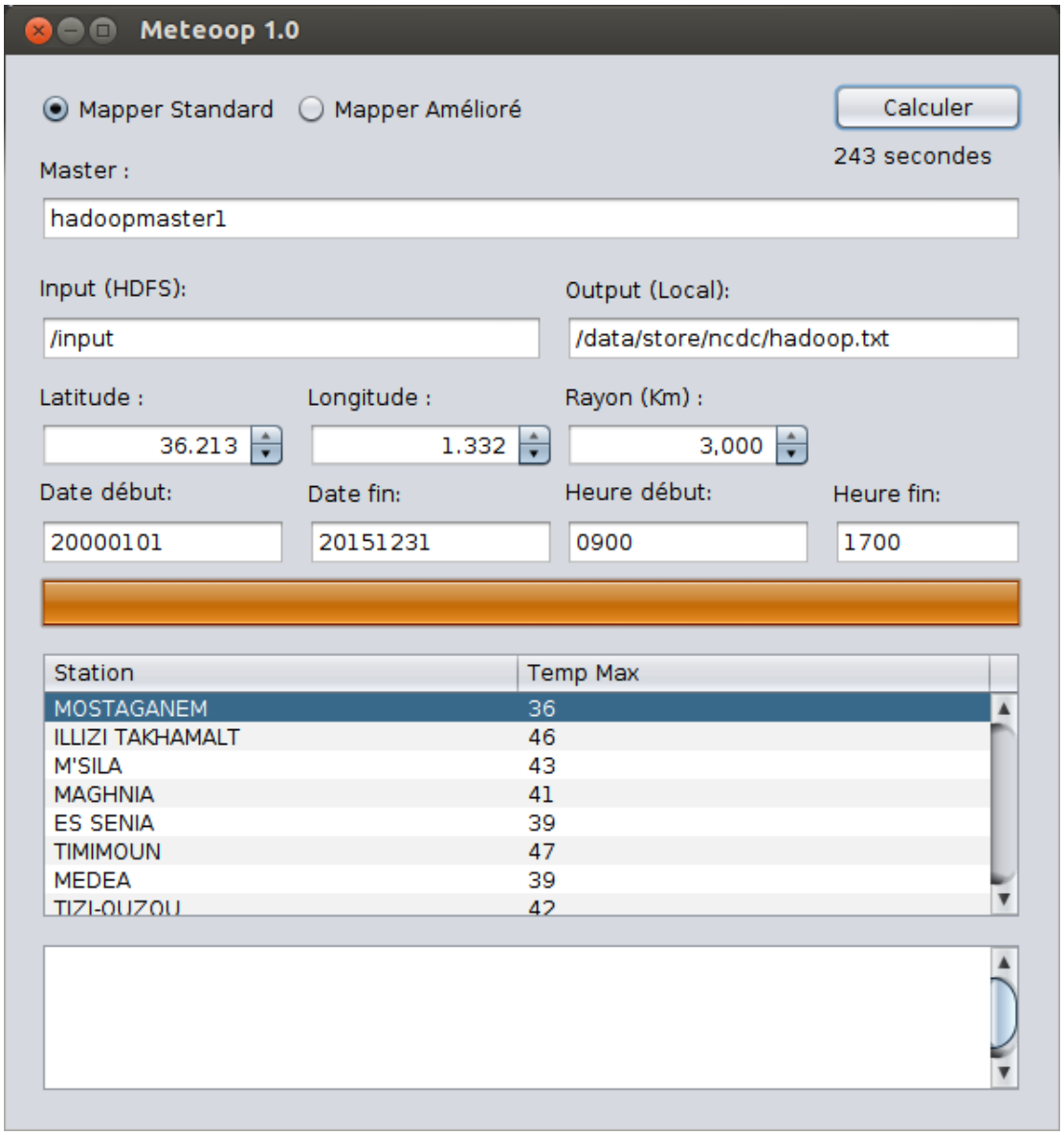


FIGURE 4.4 – Interface de l'application

4.6.1 Présentation du NCDC

Le National Climatic Data Center (NCDC) des États-Unis, fondé en 1951, situé à Asheville en Caroline du Nord est le plus important centre mondial de collecte des données météorologiques. Le centre possède plus de 150 années de données archivées et reçoit chaque jour plus de 224 gigaoctets de nouvelles informations. Le NCDC archive 99 pourcent des données de la National Oceanic and Atmospheric Administration (NOAA), y compris plus de 320 millions d'archives sur papier ; 2,5 millions de microfiches ; et environ 1,2 pétaoctet de données numériques. Le NCDC possède des images prise par les satellites météorologiques depuis 1960.

4.6.2 Les données météo

La base de données météo du NCDC est disponible sous forme des fichiers GZIP. Chaque fichier GZIP contient un fichier texte constitué de plus de 5000 lignes et chaque ligne respecte un format bien défini :

...
05-10 Code USAF de la station
11-15 Code WBAN de la station
16-23 Date sous le format YYYYMMDD
24-27 Heure sous le format HHMM
29-34 Latitude [-90000, +90000] x1000 Angular Degrees
35-41 Longitude [-179999, +180000] x1000 Angular Degrees
...
88-92 Température [-0932, +0618] x10 Degrees Celsius
...

Les fichiers GZIP sont accessibles et téléchargeables depuis un serveur FTP public ftp://ftp.ncdc.noaa.gov/pub/data/noaa/

4.6.3 Code Java d'extraction de la température à partir d'une ligne donnée

```
1  public static KeyValues getTemperature(String line, double RADIUS
2  , double LAT, double LON, String DATE_FROM, String DATE_TO
3  , String TIME_FROM, String TIME_TO) {
4
5          String tempQuality = line.substring(92, 93);
6          if (!tempQuality.matches("[01459]")) {
7                  return null;
8          }
9
10         int temp = Integer.parseInt(line.substring(line.charAt(87)
11                 == '+' ? 88 : 87, 92));
12         if (temp == 9999) {
13                 return null;
14         }
15
16         String date = line.substring(15, 23);
17         int cdatefrom = date.compareTo(DATE_FROM);
18         int cdateto = date.compareTo(DATE_TO);
19         if (cdatefrom < 0 || cdateto > 0) {
20                 return null;
21         }
22
23         String time = line.substring(23, 27);
24         int ctimefrom = time.compareTo(TIME_FROM);
25         int ctimeto = time.compareTo(TIME_TO);
26         if (ctimefrom < 0 || ctimeto > 0) {
27                 return null;
28         }
29
30         double lat = Double.parseDouble(line.substring(28, 34)) / 1000;
```

```
31          double lon = Double.parseDouble(line.substring(34, 41)) / 1000;
32          if (distance(LAT, LON, lat, lon) > RADIUS) {
33                  return null;
34          }
35
36          return new KeyValues(line.substring(4, 15), temp / 10);
37  }
38
39  public static double distance(double lat1, double lon1
40  , double lat2, double lon2) {
41          double theta = lon1 - lon2;
42          double dist = Math.sin(deg2rad(lat1))
43          * Math.sin(deg2rad(lat2))
44          + Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2))
45          * Math.cos(deg2rad(theta));
46          dist = Math.acos(dist);
47          dist = rad2deg(dist);
48          dist = dist * 60 * 1.1515;
49          dist = dist * 1.609344;
50          return (dist);
51  }
52
53  public static double deg2rad(double deg) {
54          return (deg * Math.PI / 180.0);
55  }
56
57  public static double rad2deg(double rad) {
58          return (rad * 180.0 / Math.PI);
59  }
```

Code 4.6 – Extraction de la température

4.6.4 Code Java du Mapper

```
1   public class MaxTemperatureMapper extends
2           Mapper<LongWritable, Text, Text, IntWritable>{
3
4           @Override
5           public void map(LongWritable key, Text value, Mapper.Context
                context)
6           throws IOException, InterruptedException {
7
8                   String line = value.toString();
9
10                  Configuration conf = context.getConfiguration();
11                  double RADIUS = conf.getFloat("RADIUS", 0);
12                  double LAT = conf.getFloat("LAT", 0);
13                  double LON = conf.getFloat("LON", 0);
14                  String DATE_FROM = conf.get("DATE_FROM");
15                  String DATE_TO = conf.get("DATE_TO");
16                  String TIME_FROM = conf.get("TIME_FROM");
17                  String TIME_TO = conf.get("TIME_TO");
18
19                  KeyValues temperature = Utilities.getTemperature(
20                          line
21                          , RADIUS
```

```
22                          , LAT
23                          , LON
24                          , DATE_FROM
25                          , DATE_TO
26                          , TIME_FROM
27                          , TIME_TO);
28
29                  if (temperature != null) {
30                          context.write(new Text(temperature.Key)
31                          , new IntWritable(temperature.Values[0]));
32                  }
33          }
34  }
```

Code 4.7 – La classe du Mapper

4.6.5 Code Java du Reducer

```
1  public class MaxTemperatureReducer
2  extends Reducer<Text, IntWritable, Text, IntWritable> {
3
4          @Override
5          public void reduce(Text key, Iterable<IntWritable> values
6          , Context context) throws IOException, InterruptedException {
7
8                  int maxValue = Integer.MIN_VALUE;
9                  for (IntWritable value : values) {
10                         maxValue = Math.max(maxValue, value.get());
11                 }
12                 context.write(key, new IntWritable(maxValue));
13         }
14 }
```

Code 4.8 – La classe du Reducer

4.6.6 Résultat

Les résultats ont montré que la configuration optimisée a amélioré les performances de 25% en moyenne :

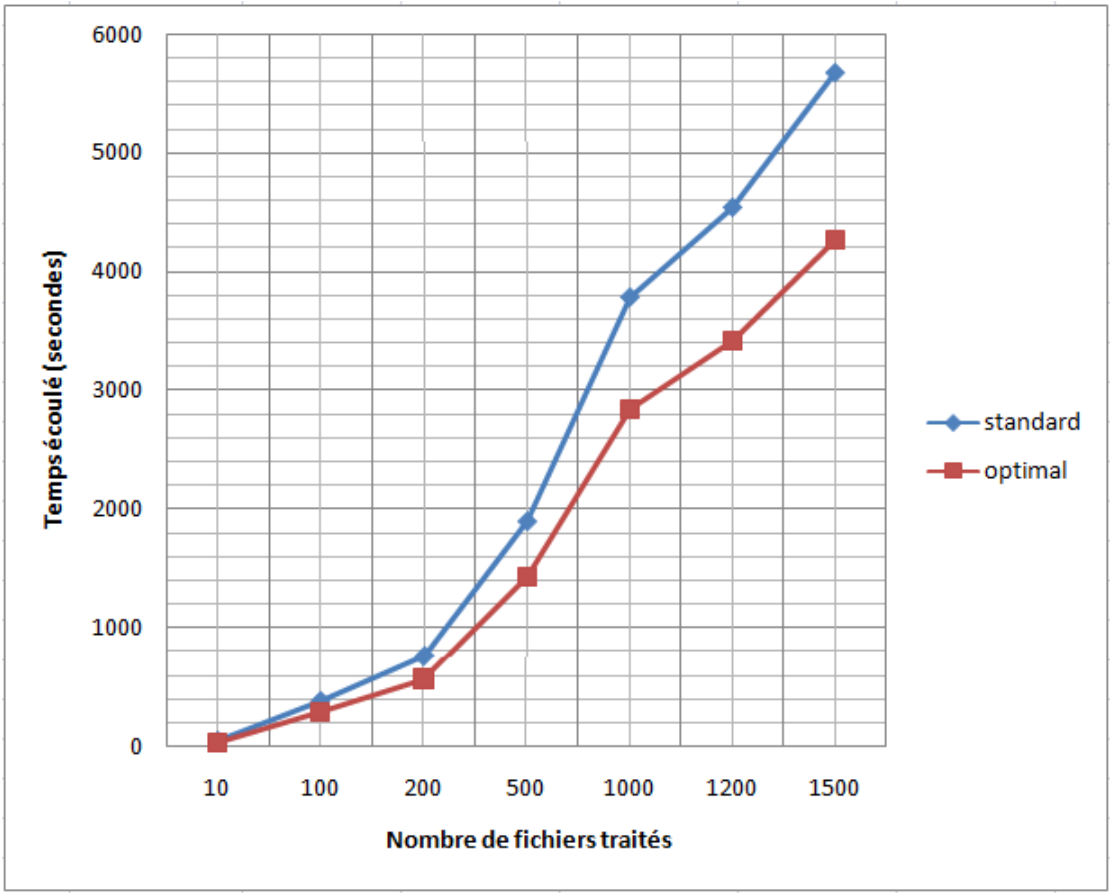


FIGURE 4.5 – Résultat avec la configuration optimisée

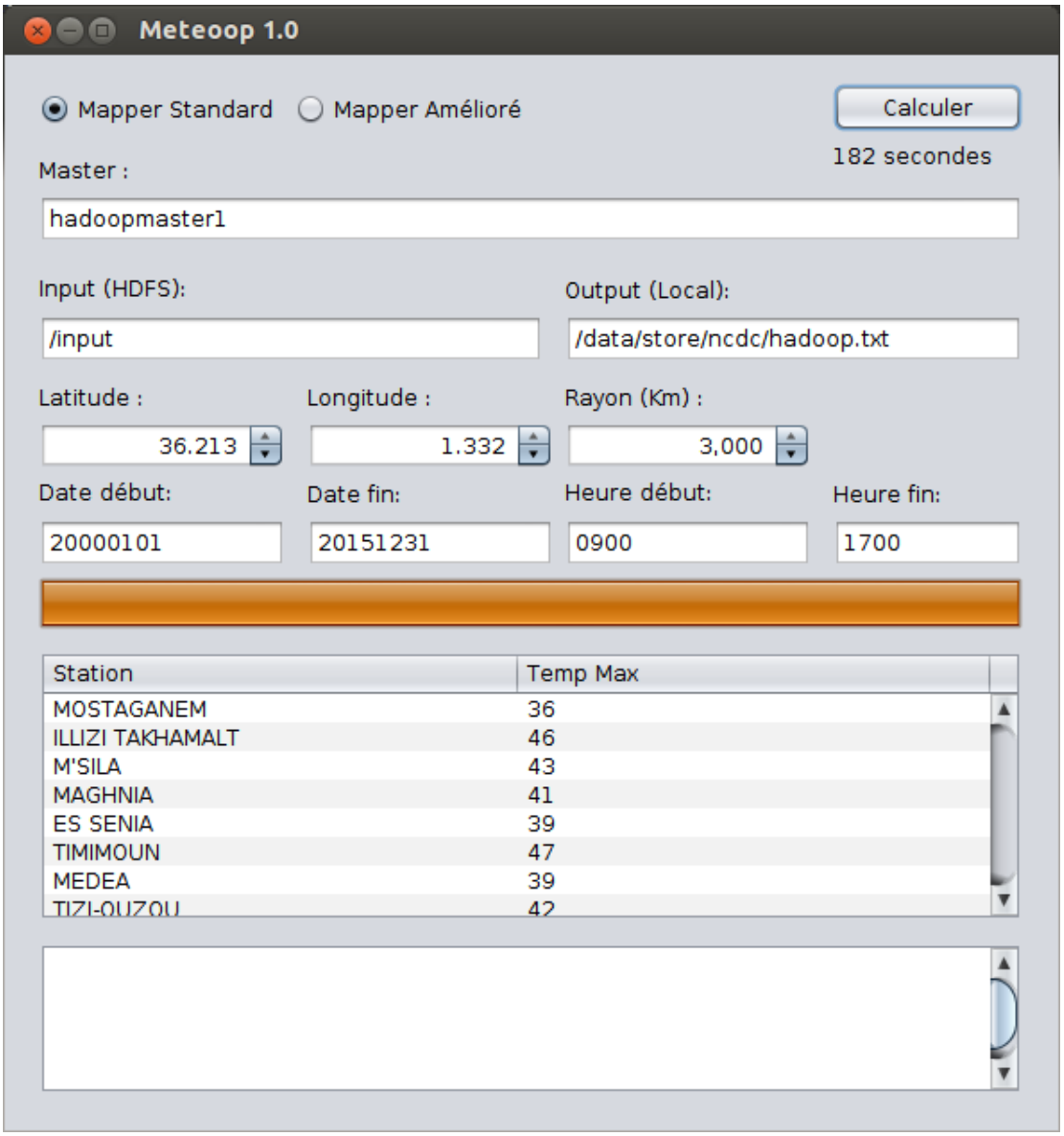


FIGURE 4.6 – Exemple du résultat avec la configuration optimisée

4.7 Optimisation supplémentaire

YARN par défaut crée un mapper pour chaque fichier GZIP (MaxTemperatureMapper) et ça engendre un surcharge sur l'ensemble des machines du cluster dans le cas d'un nombre énorme de fichiers. Pour remédier à ça on a implémenté un Mapper (MaxTemperatureMapperCombine) qui décompresse plusieurs fichier GZIP et les assigner à un seul mapper.

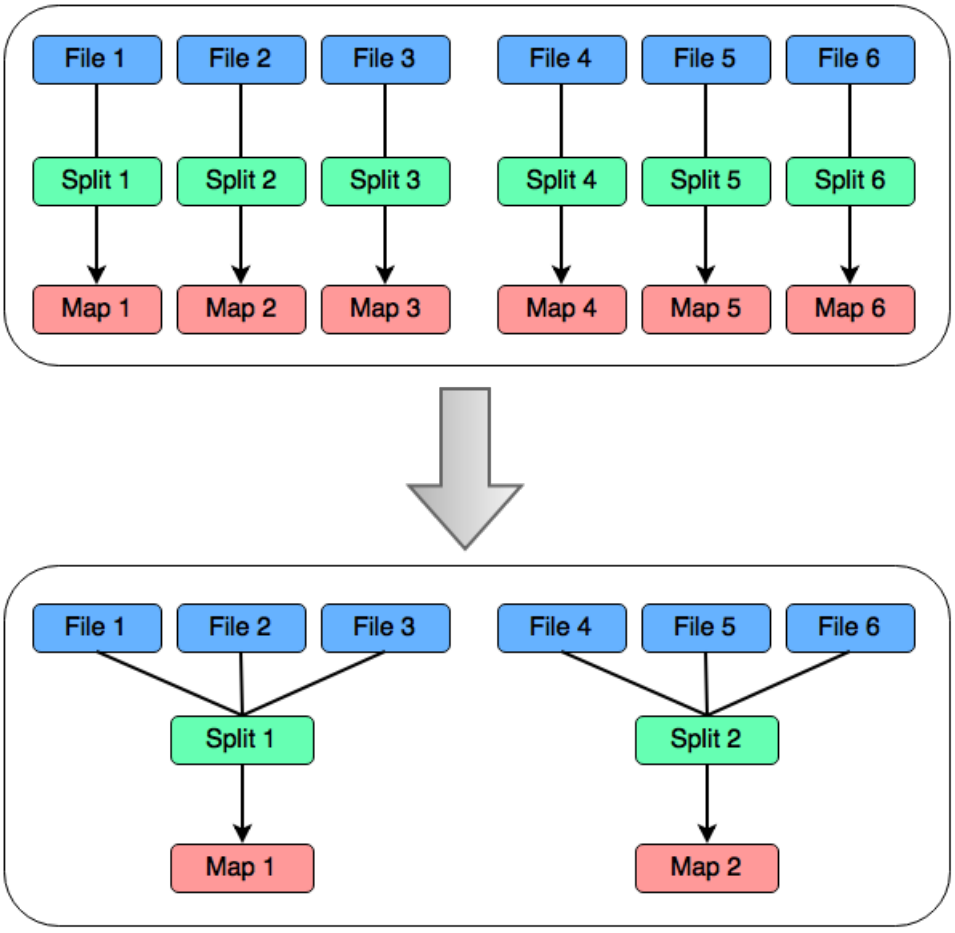


FIGURE 4.7 – MaxTemperatureMapperCombine

Pour cela on a implémenté quatre classes [13] :

— CompressedCombineFileInputFormat
— CompressedCombineFileRecordReader
— CompressedCombineFileWritable
— MaxTemperatureMapperCombine

Voici le code Java du nouveau Mapper :

```
1  public class MaxTemperatureMapperCombine extends
2  Mapper<CompressedCombineFileWritable, Text, Text, IntWritable>{
3
4          @Override
5          public void map(CompressedCombineFileWritable key, Text value
6          , Mapper.Context context) throws IOException,
               InterruptedException {
7
8                  String line = value.toString();
9
10                 Configuration conf = context.getConfiguration();
11                 double RADIUS = conf.getFloat("RADIUS", 0);
12                 double LAT = conf.getFloat("LAT", 0);
13                 double LON = conf.getFloat("LON", 0);
14                 String DATE_FROM = conf.get("DATE_FROM");
15                 String DATE_TO = conf.get("DATE_TO");
```

```
16                  String TIME_FROM = conf.get("TIME_FROM");
17                  String TIME_TO = conf.get("TIME_TO");
18
19                  KeyValues temperature = Utilities.getTemperature(
20                  line
21                  , RADIUS
22                  , LAT
23                  , LON
24                  , DATE_FROM
25                  , DATE_TO
26                  , TIME_FROM
27                  , TIME_TO);
28
29                  if (temperature != null) {
30                          context.write(new Text(temperature.Key)
31                          , new IntWritable(temperature.Values[0]));
32                  }
33          }
34  }
```

Code 4.9 – MaxTemperatureMapperCombine

Les résultats ont montré que le nouveau mapper a amélioré les performances de 45% en moyenne :

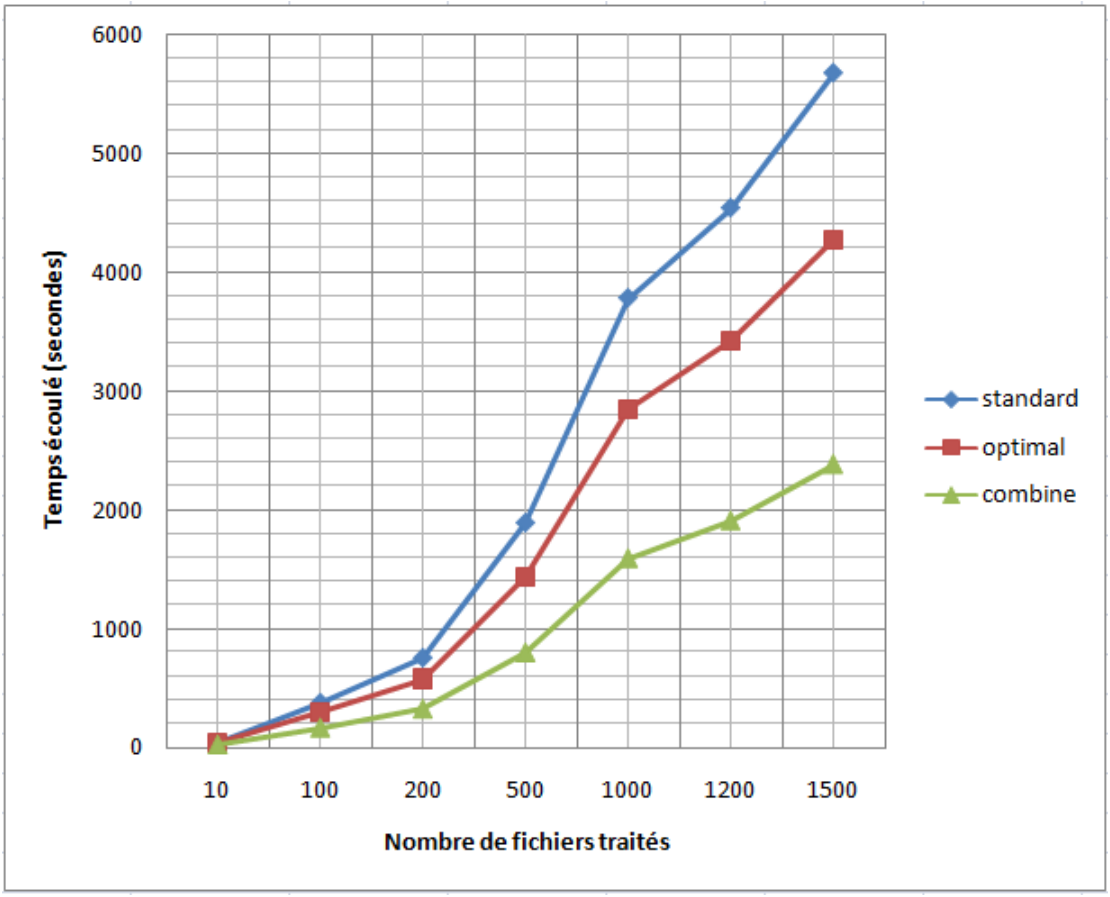


FIGURE 4.8 – Résultat avec le mapper amélioré (combine)

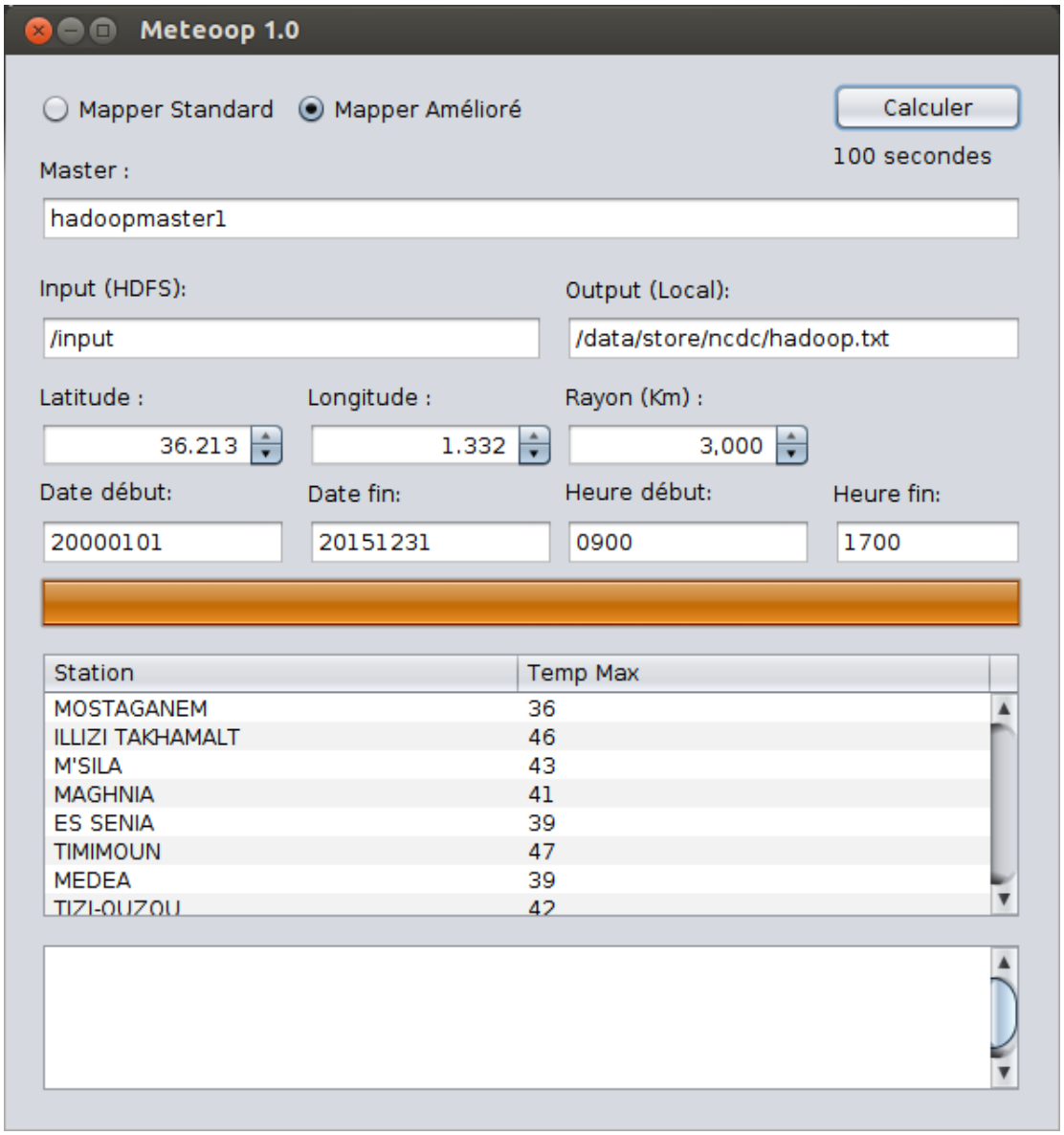


FIGURE 4.9 – Exemple du résultat avec le mapper amélioré

4.8 Conclusion

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

— Une bonne configuration d'Hadoop peut améliorer la performance du traitement de plus de 25%
— Une bonne implémentation du mapper peut améliorer la performance du traitement de plus de 45%

Conclusion générale

Notre travail a fait l'objet d'une étude des nouvelles technologies du Big Data, Nous nous sommes intéressés au framework Hadoop et son optimisation. La partie applicative consistait à déployer et optimiser la configuration d'Hadoop dans un environnement distribué hétérogène de machines.

On a proposé une démarche d'optimisation du nombre de container créé par YARN, les résultats ont montré que la configuration optimisée a amélioré les performances de 25% en moyenne. En plus, on a implémenté un Mapper optimisé pour traiter les petits fichiers compressés. Ce mapper a amélioré les performances de 45% en moyenne.

Ce modeste travail nous a permis de découvrir et bien comprendre l'architecture d'Hadoop et son fonctionnement, comme elle nous a donné l'opportunité de travailler sur plusieurs outils (LXC, bash scripting,...)

Perspectives

Comme améliorations de ce travail d'avenir, nous pouvons envisager :

— Récupérer le Facteur de Performance des noeuds en temps réel
— Automatiser la configuration optimisée des noeuds
— Enrichir l'application avec d'autres statistiques utiles : température (moyenne et minimale), pression, pluie, vent,...

Annexe A

Configuration d'Hadoop

Tous les fichiers de configuration d'Hadoop se trouvent dans le dossier **$HADOOP_HOME/etc/hadoop**

A.1 Configuration du fichier core-site.xml

```
<configuration>
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://hadoopmaster1:9000</value>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/hadoopuser/hadoop-2.7.1/tmp</value>
        </property>
</configuration>
```

Code A.1 – Configuration du fichier core-site.xml

A.2 Configuration du fichier hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.blocksize</name>
                <value>4m</value>
        </property>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.permissions</name>
                <value>false</value>
        </property>
        <property>
                <name>dfs.namenode.name.dir</name>
```

```
16                 <value>/home/hadoopuser/hdfs/namenode</value>
17         </property>
18         <property>
19                 <name>dfs.datanode.data.dir</name>
20                 <value>/home/hadoopuser/hdfs/datanode</value>
21         </property>
22         <property>
23                 <name>dfs.namenode.http-address</name>
24                 <value>hadoopmaster1:50070</value>
25         </property>
26         <property>
27                 <name>dfs.namenode.secondary.http-address</name>
28                 <value>hadoopmaster1:50090</value>
29         </property>
30 </configuration>
```

Code A.2 – Configuration du fichier hdfs-site.xml

A.3 Configuration du fichier mapred-site.xml

```
1  <configuration>
2          <property>
3                  <name>mapreduce.framework.name</name>
4                  <value>yarn</value>
5          </property>
6  </configuration>
```

Code A.3 – Configuration du fichier mapred-site.xml

A.4 Configuration du fichier yarn-site.xml

```
1  <configuration>
2          <property>
3                  <name>yarn.nodemanager.aux-services</name>
4                  <value>mapreduce_shuffle</value>
5          </property>
6          <property>
7                  <name>yarn.nodemanager.aux-services.mapreduce_shuffle.
                       class</name>
8                  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
9          </property>
10         <property>
11                 <name>yarn.resourcemanager.resource-tracker.address</
                       name>
12                 <value>hadoopmaster1:8025</value>
13         </property>
14         <property>
15                 <name>yarn.resourcemanager.scheduler.address</name>
16                 <value>hadoopmaster1:8030</value>
17         </property>
18         <property>
19                 <name>yarn.resourcemanager.address</name>
```

```
20                 <value>hadoopmaster1:8050</value>
21         </property>
22         <property>
23                 <name>yarn.nodemanager.resource.cpu-vcores</name>
24                 <value>2</value>
25         </property>
26         <property>
27                 <name>yarn.nodemanager.resource.memory-mb</name>
28                 <value>2560</value>
29         </property>
30         <property>
31                 <name>yarn.scheduler.minimum-allocation-vcores</name>
32                 <value>1</value>
33         </property>
34         <property>
35                 <name>yarn.scheduler.maximum-allocation-vcores</name>
36                 <value>2</value>
37         </property>
38         <property>
39                 <name>yarn.scheduler.minimum-allocation-mb</name>
40                 <value>512</value>
41         </property>
42         <property>
43                 <name>yarn.scheduler.maximum-allocation-mb</name>
44                 <value>2560</value>
45         </property>
46 </configuration>
```

Code A.4 – Configuration du fichier yarn-site.xml

A.5 Configuration du fichier "slaves"

```
1 hadoopmaster1
2 hadoopslave11
3 hadoopslave12
4 hadoopslave20
5 hadoopslave21
```

Code A.5 – Configuration du fichier "slaves"

Bibliographie

[1] Izzeddine AMEUR. Mapreducerp* : Un framework pour le traitement parallèle de très grande quantité de données. Mémoire de master, Université Hassiba Benbouali de Chlef, 2014.

[2] Abdesalam AMRANE. Big data : Concepts et cas d'utilisation. Rapport, CERIST, 2015.

[3] Apache. Hadoop. hadoop.apache.org.

[4] Mickael BARON. Généralités sur hdfs et mapreduce. Article Web, 2014.

[5] Sid Ahmed Amine BENAOUDA. Implantation du modèle mapreduce dans l'environnement distribué hadoop : Distribution cloudera. Mémoire de master, Université Abou Bekr Belkaid Tlemcen, 2015.

[6] Rudi BRUCHEZ. *Les bases de données NoSQL et le Big Data*. Editions EYROLLES, 2015.

[7] Fabrice DEMARTHON, Denis DELBECQ, and Grégory FLECHET. La déferlante des octets. *CNRS*, (269), 2012.

[8] CLOAREC Erwann. Hadoop : Optimisation et ordonnancement. Master's thesis, Université François-Rabelais, Tours, 2014.

[9] Hortonworks. Hortonworks data platfrom. www.hortonworks.com.

[10] Laurent JOLIA-FERRIER. *Big Data : Concepts et mise en oeuvre de Hadoop*. Editions ENI, 2014.

[11] Djamal LARBI-AISSA. La distribution des données par mappage dans le modèle de programmation mapreduce. Mémoire de master, Université Hassiba Benbouali de Chlef, 2013.

[12] Christophe PARAGEAUD. Big data : La jungle des différentes distributions open source hadoop. Article Web, 2013.

[13] Sujay SOM. Process small, compressed files in hadoop using combinefileinputformat. Article Web, 2014.

[14] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. *IPDPSW IEEE*, 2010.

[15] Yahoo. Developer network. developer.yahoo.com.

[16] Nabil ZEGGARI and Amine KOUICEM. Les technologies du big data. Mémoire de master, ESI (ex. INI), 2013.

Table des figures

Liste des tableaux