**Le logiciel Hadoop Open Source**

1. **Généralités**
2. **Qu'est ce qu'Hadoop?**

Il s'agit d'un [framework](http://www.journaldunet.com/web-tech/dictionnaire-du-webmastering/1203355-framework-definition-traduction/) **Open Source** conçu pour réaliser des traitements sur des volumes de données massifs, de l'ordre de plusieurs petaoctets (soit plusieurs milliers de To). Il s'inscrit donc typiquement sur le terrain du Big Data.

Il est Géré sous l'égide de la fondation [**Apache**](http://www.journaldunet.com/web-tech/dictionnaire-du-webmastering/1203241-apache-definition/). C’est une plateforme permettant d'établir un dialogue entre plusieurs machines d'un cluster.

**Hadoop se base** alors **sur les travaux de Google au niveau du GFS** (Google's distributed filesystem) et de MapReduce pour l'analyse des données d'un système GFS.

Il fournit un framework en Java, pour assurer le développement de l'exécution de tâches map/reduce.

Hadoop et le HDFS forment aujourd'hui la nouvelle infrastructure technologique de l'ère numérique.

1. **Historique**

* Hadoop a été créé en 2004 par Doug Cutting pour les besoins du projet Apache Nutch, un moteur de recherche open source. L'objectif était de le rendre capable d'indexer et de rechercher dans des collections de la taille du Web. Doug Cutting eut l'idée de créer un framework de gestion de fichiers distribués. Hadoop était né.
* **Yahoo!** en est devenu ensuite le principal contributeur.
* En 2006, Hadoop devient un sous-projet d'Apache Lucene
* En 2008, Hadoop devient un projet indépendant de la fondation Apache.
* 2011: Hadoop désormais utilisé par de nombreuses autres entreprises et des universités, et le cluster Yahoo comporte **42000 machines** et des **centaines de peta-octets** d'espace de stockage.

1. **Au-delà de Yahoo et Facebook, qui utilise cette solution?**

* C'est le cas d'autres réseaux sociaux comme **Twitter** et LinkedIn, avec des besoins d'exécution d'informations en gros volumes.
* Sur le front du e-commerce, **eBay** exploite aussi le framework.
* Il est aussi intégré par un grand nombre de services de Cloud Computing. On compte parmi eux **Amazon** qui permet de faire tourner Hadoop sur ses services de plate-forme Amazon EC2 et de stockage Amazon S3. Le géant de l'e-commerce a d'ailleurs rendu son propre système de fichiers (S3 filesystem) compatible avec Hadoop.
* Récemment, **Microsoft** a également annoncé la disponibilité d'une pré-version d'Hadoop pour son Cloud **Azure**.

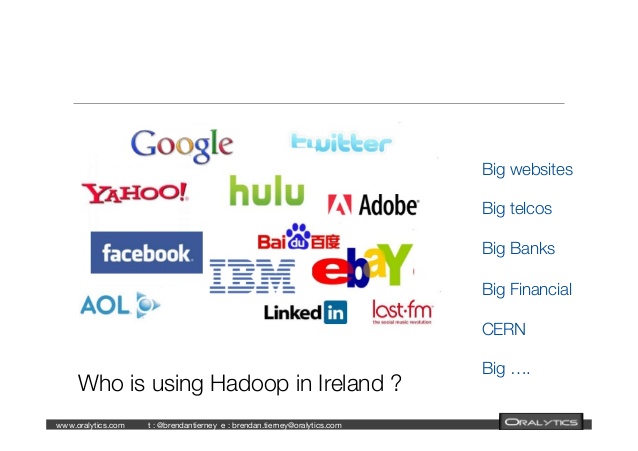


Fig. 1 Qui utilise Hadoop

1. **Contexte de création d'Hadoop et ses objectifs**

Actuellement, le volume des données en circulation connaît une démultiplication permanente: 5 exaoctets de données sont désormais produits tous les deux jours, soit le même volume que l'ensemble des données produites de l'aube de la civilisation à 2003.

L'échelle de cette croissance de données surpasse la capacité raisonnable des technologies traditionnelles, précisément les systèmes de gestion de bases de données relationnelles (SGBDR), ou même la configuration matérielle typique supportant les accès à ces données. Plus encore, les données canalisées vers le réseau Internet créent de la pression pour la capture cohérente et rapide de ces données.

Le nombre de sites Internet dans le monde aujourd'hui est estimé à plus de 1 milliard avec une croissante de 5,1 % (estimation faite par Netcraft), sachant qu'au-delà de 5 secondes d'attente, l'utilisateur considère que la requête a échoué et passe à autre chose. Résoudre ce problème ne peut se faire ni avec les approches de centralisation des données dans un seul serveur comme on l'a fait dans le passé, et ni avec les approches centralisées de traitement in-memory. Il est dès lors indispensable de penser à d'autres approches, et c'est là où Hadoop entre en scène.

Donc, L’objectif de Hadoop est de résoudre les deux principaux problèmes de la manipulation de grandes quantités de données:

* **stocker ces données (limitation de la taille des disques durs)**
* **rechercher rapidement dans ces données (limitation des puissances de calcul)**

1. **Approche conceptuelle d'Hadoop**

Pour répondre à ces challenges, l'idée de Google est de développer une approche conceptuelle qui consiste à

* Paralléliser le traitement de ces données sur plusieurs nœuds d'un cluster d'ordinateurs.
* Distribuer le stockage des données

Ainsi, la croissance des données est gérée en augmentant simplement les nœuds dans le cluster. Cette approche conceptuelle a été adoptée par le marché et est à la base de toutes les technologies Big Data actuellement.

**5-1 Au niveau du traitement**

Google décide de découper **le problème d'indexation des pages web** en sous-tâches ou sous-problèmes qui seront distribués dans le cluster pour exécution. Pour ce faire, Google décide de construire un **index inversé par mot-clé** contenu dans chaque page web. Pour faire simple, un index inversé correspond à la page d'index d'un livre; il est constitué des mots-clés, avec pour chaque mot-clé ses différentes localisations dans tout le document. Ainsi, chaque fois que vous recherchez un mot, vous pouvez vous servir de l'index pour identifier la ou les pages où le mot que vous cherchez se situe, cela vous évite de fouiller tout le document page par page pour retrouver le mot. Aujourd'hui, la recherche Internet fonctionne selon ce principe. **Le moteur de recherche constitue une base d'index inversés pour chaque mot**.

Pour construire l'index inversé, Google passe par un raisonnement en trois phases consécutives:

* La première phase appelée par Google la phase Map, consiste à assigner à chaque nœud du cluster la tâche d'attribuer à chaque mot de la page web un indice correspondant à la page dans laquelle il est. Cet indice peut être le titre de la page, le numéro de la page, bref n'importe quel élément qui permet d'identifier la page de façon unique parmi toutes les pages qui constituent tout le site web. Cette tâche s'exécute parallèlement (en même temps) dans tout le cluster.
* La deuxième étape, appelée par Google le Shuffle, consiste pour chaque nœud à **trier** par ordre alphabétique les mots auxquels il a affecté un index. Cette étape est intermédiaire et permet de faciliter le travail effectué par la troisième phase.
* La troisième et dernière phase, appelée par Google le Reduce, consiste pour chaque mot dans l'ensemble des nœuds du cluster, à regrouper l'ensemble de ses indices. Ainsi, on obtient l'index inversé.

**5-2 Au niveau du stockage**

Google décide de ne pas centraliser le stockage de toutes les pages Internet vers un seul serveur pour la construction des index inversés. Comme la tâche de construction des index est partagée entre les nœuds du cluster, les fichiers contenant les mots (les pages web) doivent être divisés et chaque morceau de fichier doit être stocké de façon redondante sur le disque dur des nœuds du cluster de sorte que si un nœud tombe en panne tout au long du traitement, cette panne n'affecte pas les autres tâches.

Techniquement, le stockage de fichiers sur un disque dur se fait à l'aide de ce que l'on appelle un système de fichiers (File System). Le système de fichiers est unique par ordinateur, ce qui pose problème dans un cluster où l'ensemble des nœuds doit être vu comme un seul ordinateur. Pour résoudre ce problème et gérer la redondance des données sur plusieurs disques durs, Google met sur pied un nouveau type de système de fichiers appelé « système de fichiers distribués » (Distributed File System - DFS), qui est installé sur le cluster. La figure suivante résume le travail de construction d'index inversés par le raisonnement en trois phases de Google.

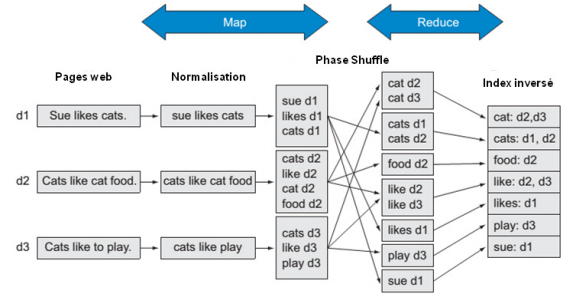
[](http://juvenal-chokogoue.developpez.com/tutoriels/hadoop-fonctionnement/images/Approche%20conceptuelle%20Hadoop.PNG)

Fig. 2 paradigme en trois phases de Google

1. **La structure d’Hadoop : construction et éléments de base**

Lorsque l’on évoque Hadoop, c’est généralement de l’ensemble de l’écosystème du logiciel dont il est question. En plus des composants du noyau, (le Core Hadoop), il existe de nombreuses extensions aux dénominations originales, comme Pig, Chukwa, Oozie ou encore ZooKeeper, qui permettent au framework de travailler des quantités très importantes de données. Ces projets, élaborés les uns à partir des autres, sont soutenus par l’Apache Software Foundation.

Le noyau, ou **Core Hadoop**, constitue la base fondamentale de l’écosystème de Hadoop. Les éléments qui le composent sont présents dans la version 1 du module de base **Hadoop Common**, et les deux couches de base: **Hadoop Distributed File System (HDFS)** et **MapReduce Engine**.

### Hadoop Common: Le module Hadoop Common dispose donc d’une palette ****de fonctions de base**** très vaste. Les données d’archives Java (JAR) en font partie et sont nécessaires pour démarrer Hadoop. Des librairies pour la sérialisation des données tout comme le module Hadoop Common des codes sources sont disponibles pour la documentation du projet et des sous-projets.

* **HDFS**: c’est un système de fichiers distribué conçu pour gérer la distribution des données sur les différents nœuds du cluster
* **MapReduce**: c’est un framework en Java permettant de gérer le découpage des traitements en plusieurs tâches et la parallélisation de ces tâches sur un grand nombre de nœuds de manière fiable et tolérant aux pannes et l’exécution des requêtes sur les données stockées par le cluster. À partir de la version 2.3, le système de gestion en clusters **YARN** (également appelé MapReduce 2.0) le remplace. Ceci exclut l’algorithme MapReduce du système de gestion, qui fonctionne systématiquement à partir de YARN.

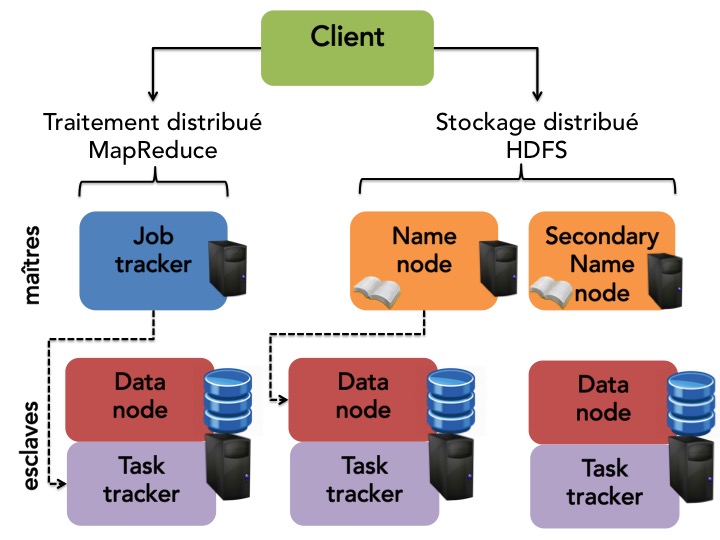


Fig. 3 structure de Hadoop

### Hadoop Distributed File System (HDFS)

Hadoop est basé sur le système de fichiers Google (GFS) pour ses propres services de Cloud et fournit un système de fichiers distribué conçu pour gérer la répartition du stockage des données utilisateurs sur les différents nœuds de manière fiable et tolérante aux pannes, du stockage et de la rédondance des fichiers sur le cluster.

Le système de fichiers distribués Hadoop (HDFS) est :

* Distribué: les données sont réparties sur les machines du cluster.
* Répliqué: en cas de panne, aucune donnée n'est perdue.
* Optimisé: pour la localisation des données et des traitements.

En effet:

* Les fichiers sont physiquement découpés en blocs d'octets de grande taille (par défaut 64 Mo) pour optimiser les temps de transfert et d'accès. La longueur des blocs de données, tout comme leur **degré de rédondance** sont configurables.
* Ces blocs sont ensuite **répartis** sur plusieurs machines, permettant ainsi de **traiter un même fichier en parallèle**. Cela permet aussi de ne pas être limité par la capacité de stockage d'une seule machine pour au contraire tirer parti de tout l'espace disponible du cluster de machines.
* Enfin, pour garantir une tolérance aux pannes, les blocs de chaque fichier sont **repliqués**, de manière intelligente, sur plusieurs machines.

**Note:** La taille d’un bloc de données est de **64 Mo** dans HDFS contre 512 octets à 4 Ko dans la plupart des systèmes de fichiers traditionnels. Cette taille de bloc s’explique par le fait que Hadoop doit analyser de grandes quantités de données en local.

HDFS utilise une **architecture de stockage *maître-esclave*** dans laquelle le maître consiste en un **seul NameNode** et un ou **plusieurs DataNodes esclaves**.

* Le nœud maître unique appelé **name node,** unique sur le cluster, contient et stocke tous les noms et blocs des fichiers et leurs caractéristiques de manière centralisée ainsi que leur localisation dans le cluster. On peut donc le voir comme un gros annuaire.
* Une autre machine, appelée **secondary name node** sert de namenode de secours en cas de défaillance du nœud maître et il a donc pour rôle de faire des sauvegardes régulières de l'annuaire.
* Les autres nœuds, les **data nodes,** les esclaves, les DataNode, plusieurs par cluster, stockent le contenu des fichiers eux-même, fragmentés en blocs (64KB par défaut). Les DataNodes prennent en charge les **opérations de lecture et d'écriture** avec le système de fichiers. Ils prennent également en charge la création, la suppression et la réplication de blocs en fonction des instructions fournies par NameNode.

Chaque DataNode envoie régulièrement un signe de vie au NameNode, on appelle cela le **heartbeat** (battement de cœur). Si ce signal ne se manifeste pas, le NameNode décrète l’esclave comme étant « mort » et veille à l’aide des copies de fichiers et des autres nœuds qu’il y ait suffisamment de blocs de données dans le cluster qui soient disponibles.

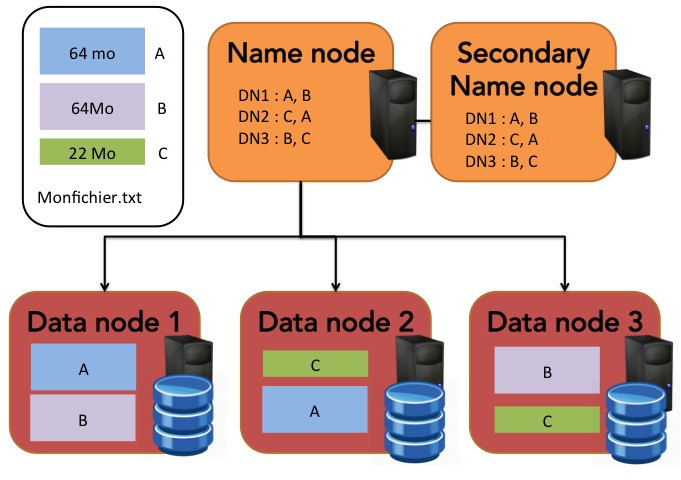


Fig. 4 architecture maitre-esclave de HDFS

HDFS fournit un shell comme tout autre système de fichiers et une liste de commandes est disponible pour interagir avec le système de fichiers.

Pour le passage de Hadoop 1 à Hadoop 2, HDFS a été étendu à différents systèmes de sauvegarde: **NameNode HA** (High Availability) qui complète le programme avec un système en cas de panne NameNode, afin que des composants de remplacement soient utilisés automatiquement. Une fonction de **copie Snapshot** permet de plus que le système soit restauré au statut précédent. L’extension **Federation** permet également que divers NameNodes puissent opérer au sein d’un même cluster.

**Lecture d'un fichier**

Pour lire un fichier:

* Le client contacte le NameNode du cluster, indiquant le fichier qu'il souhaite obtenir; c’est à dire le *NameNode* est interrogé pour localiser l'ensemble des blocs de données.
* Le NameNode lui indique la taille, en blocs, du fichier, et pour chaque bloc une liste des addresses des DataNodes susceptibles de lui fournir; c'est-à-dire les *DataNode* qui disposent de la plus grande [bande passante](https://fr.wikipedia.org/wiki/Bande_passante).
* Le client contacte les DataNodes en question pour obtenir les blocs, qu'il reconstitue sous la forme du fichier.
* En cas de DataNode inaccessible/autre erreur pour un bloc, le client contacte un DataNode alternatif de la liste pour l'obtenir.

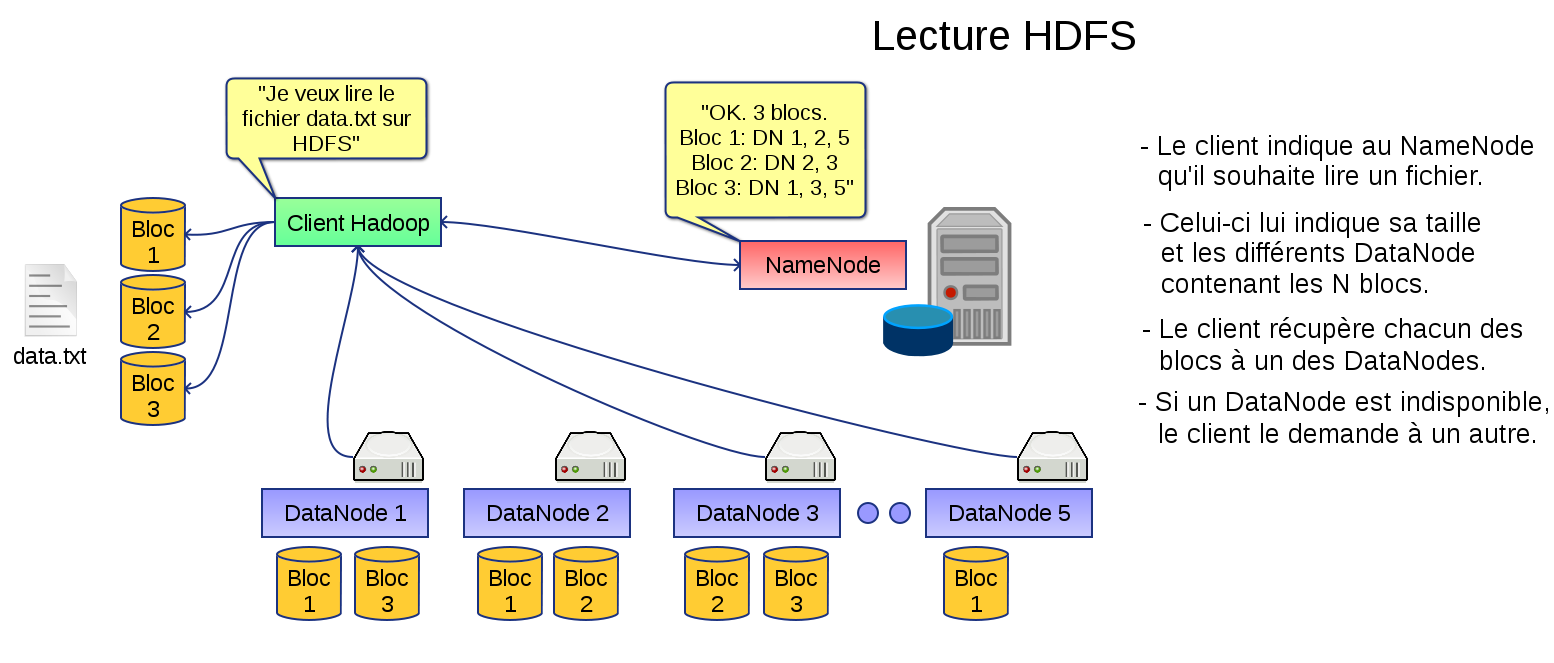


Fig. 5 lecture d’un fichier

**Écriture d'un fichier**

Pour écrire un fichier:

* Le client contacte le NameNode du cluster, indiquant la taille du fichier et son nom.
* Le NameNode confirme la demande et indique au client de fragmenter le fichier en blocs, et d'envoyer tel ou tel bloc à tel ou tel DataNode.
* Le client envoie les fragments aux DataNode.
* Les DataNodes assurent ensuite la réplication des blocs.

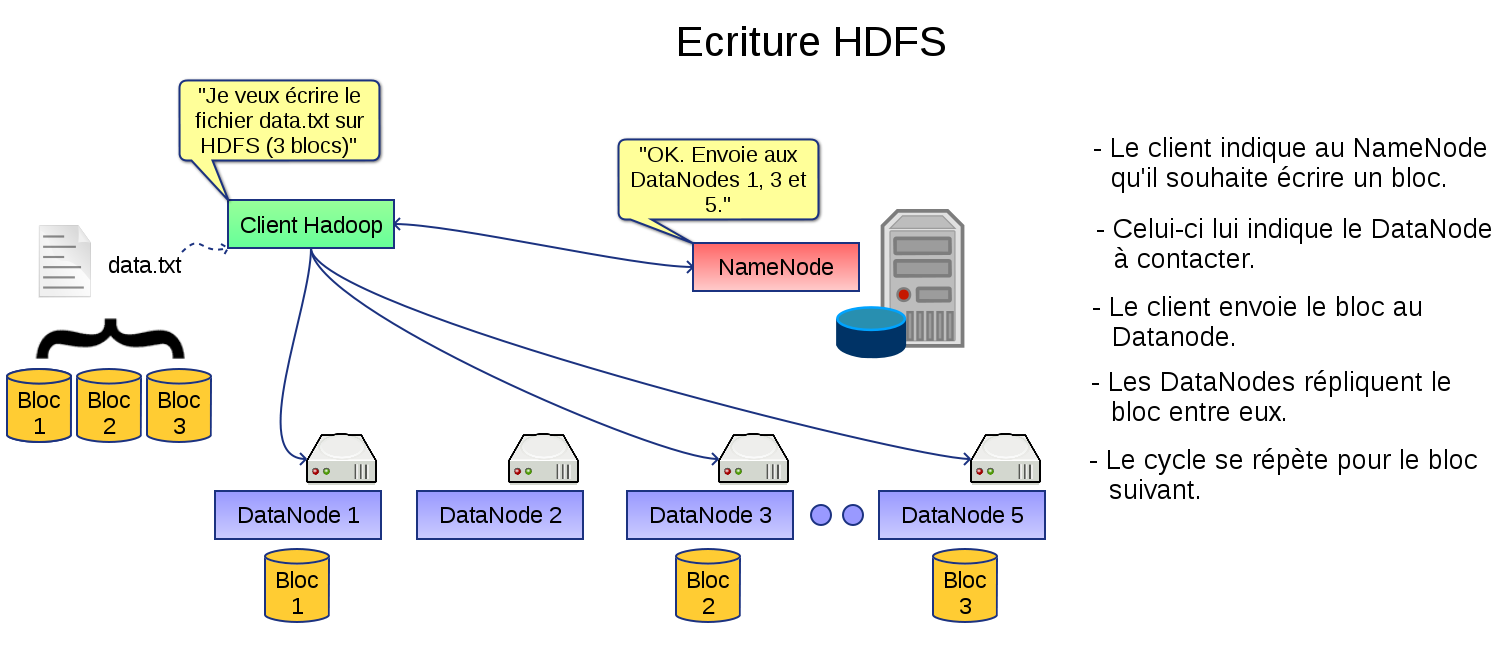


Fig. 6 écriture d’un fichier

### MapReduce Engine

Un autre composant de base du Core Hadoop est l’agorithme de Google **MapReduce**, qui est implémenté dans la version 1 du framework. Le devoir principal du MapReduce Engine consiste à **gérer les ressources et** à **guider le processus de calcul** (Job scheduling/monitoring). Il s’occupe de la distribution de tâches en tirant parti de la localité entre données et traitements sur le même nœud de façon à minimiser l’impact des transferts de données entre les nœuds du cluster sur la performance.

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

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

Le cluster exécute le job MapReduce en divisant le programme MapReduce en deux phases:

* les tâches Map
* les tâches Reduce.

Ces phases permettent aux fichiers d’être travaillés directement où ils sont stockés (data locality). Cela accélère le temps de calcul et minimise la consommation excessive de la bande passante du réseau.

* Dans la phase « Map », le nœud maître divise le problème posé en sous-problèmes et les distribue entre nœuds de traitement. Ces nœuds peuvent en cascade distribuer à nouveau les tâches qui leur ont été assignées. Les réponses sont ensuite remontées de nœuds en nœuds jusqu’au nœud maître ayant assigné les travaux à l’origine.
* Dans la phase "Reduce", le nœud maître collationne les réponses emontant des nœuds de traitement, sous forme clés/valeurs, et les combine dans un ensemble plus petit afin de fournir la réponse à la question posée à l’origine.
* Au cours de la Reduce Phase qui s’ensuit, les résultats intermédiaires du MapReduce Engine sont collectés et délivrent un résultat global

Il est à noter que les traitements Mapreduce s’opèrent sur des données structurées sous la forme (clé, valeur) et que des mécanismes d’optimisation assurent que les traitements sont distribués de telle sorte qu’ils s’opèrent au plus proche des données (c’est-à-dire idéalement sur les neuds qui hébergent les données concernées).

La tâche de réduction est toujours effectuée après la tâche map. Toujours dans notre exemple, il s'agira du calcul final du nombre total de mots en aditionant tout les chiffres 1 obtenu lors de la première opération.

Eventuellement une étape intermédiaire pourra permettre de réaliser des tris, qui permettra notamment de réaliser un comptage par catégorie des mots dans notre exemple. Un orchestrateur de tâches (un scheduler) a été dessiné pour gérer les traitements par lôt en fonction des capacités disponibles à un instant T au sein de la ferme de serveurs.

**Le framework MapReduce se compose de deux types de processus** qui contrôlent l'exécution du job: Un JobTracker maître unique et un TaskTracker esclave par nœud de cluster (on a un ensemble de tasktrackers).

1. Le jobtracker c'est le processus central qui est démarré sur le nœud de référence (le Name Node). Le JobTracker est un point de défaillance unique pour le service Hadoop MapReduce, ce qui signifie que si JobTracker tombe en panne, tous les travaux en cours sont arrêtés. Le serveur JobTracker est en communication avec HDFS; il sait où sont les données d'entrée du programme map/reduce et où doivent être stockées les données de sortie. Il peut ainsi optimiser la distribution des tâches selon les données associées.

Le JobTracker, unique sur le cluster:

* Reçoit les tâches map/reduce à exécuter (sous la forme d'une **archive Java .jar**)
* Reçoit l'emplacement des fichiers d'entrée et de sortie dans le système de fichiers distribué.
* Organise et coordonne tous les jobs qui s'exécutent sur le cluster en distribuant le logiciel/configuration aux esclaves.
* Planifie les tâches à exécuter sur les tasktrackers (les esclaves) et les surveille.
* Fournit des informations de statut et de diagnostic au job-client.
* Gère les ressources du cluster
* Suit la consommation/disponibilité des ressources
* Réexécute les tâches échouées.

1. Les esclaves tasktrackers: ce sont les processus qui traitent le programme MapReduce de l'analyste, ils sont démarrés au niveau des nœuds de données.

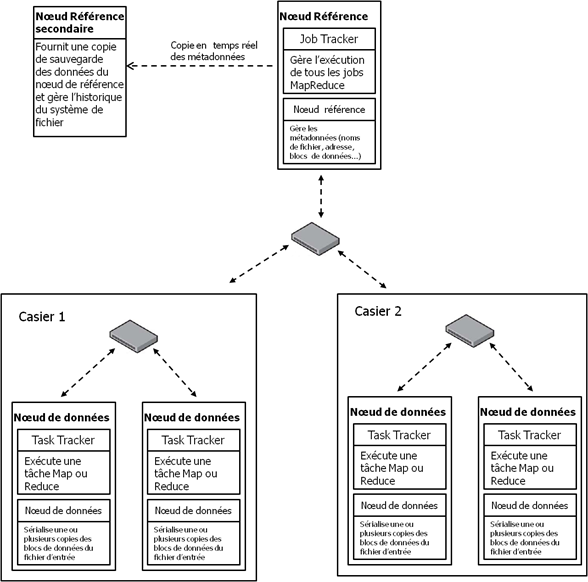
Les **TaskTracker** veillent par la suite à ce que les divers processus partiels soient traités de manière parallèle.

Les TaskTracker, plusieurs par cluster.

* Exécutent les tâches Map ou Reduce
* Envoient des rapports d'avancement au jobtracker, qui garde une copie du progrès général de chaque job.
* Si une tâche échoue, le jobtracker peut la replanifier sur un tasktracker différent.
* Les esclaves TaskTracker exécutent les tâches selon les instructions du maître et fournissent périodiquement des informations sur l'état de la tâche au maître.
* Exécutent le travail map/reduce lui-même selon l'implémentation de MapReduce (sous la forme de tâches map et reduce ponctuelles avec les données d'entrée associées).
* Stockent la sortie de la fonction de réduction dans les fichiers de sortie du système de fichiers.
* Chacun des TaskTrackers constitue une unité de calcul du cluster.

En fait, le jobtracker désigne le nœud de référence et le processus Master qui y est démarré, tandis que le Tasktracker désigne un nœud de données et le processus Worker qui y est démarré.

La figure ci-après illustre le rapport entre le jobtracker et les tasktrackers dans le cluster.

[](http://juvenal-chokogoue.developpez.com/tutoriels/hadoop-fonctionnement/images/Cluster-Hadoop.png)Figure 7 : cluster Hadoop, avec 5 nœuds. Le nœud de référence secondaire ne compte pas comme un nœud du cluster. Les nœuds de données sont des tasktrackers tandis que le nœud de référence est un jobtracker.

1. **Fonctionnement d'Hadoop**

Pour exécuter un programme map/reduce, on devra donc:

* Écrire les données d'entrée sur HDFS.
* Soumettre le programme au JobTracker du cluster.
* Récupérer les données de sortie depuis HDFS.

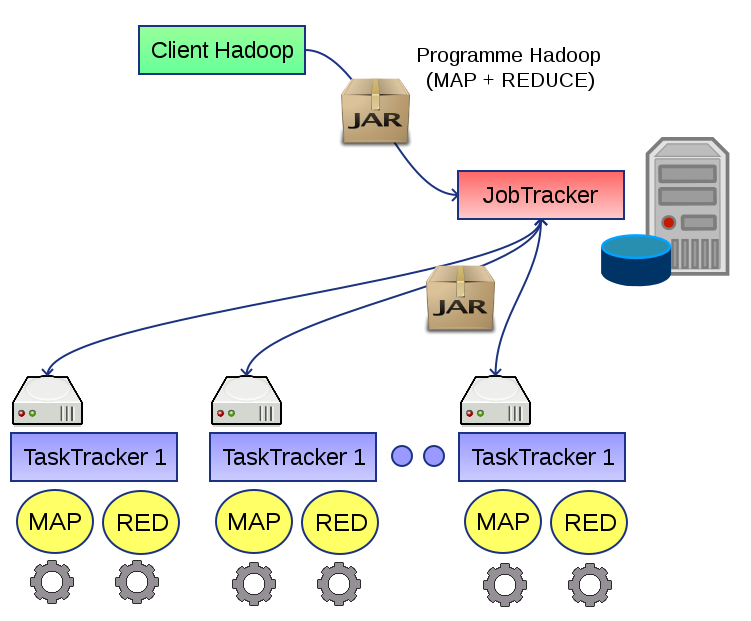
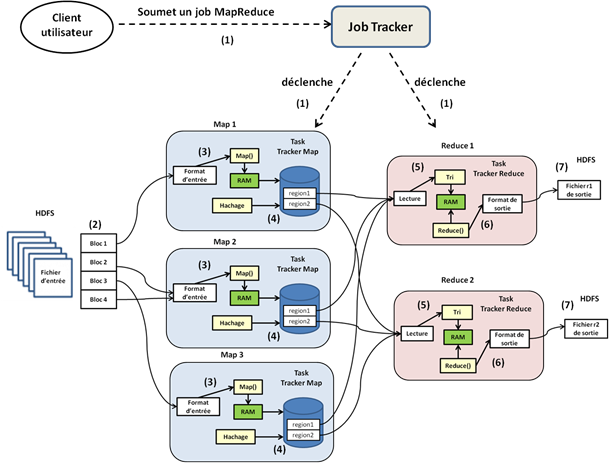


Fig. 8 Fonctionnement d'Hadoop

Le traitement MapReduce écrit par l'utilisateur qui s'appelle job MapReduce s'exécute en sept étapes:

* 1. Au départ, l'utilisateur configure le Job MapReduce: il écrit la fonction Map, la fonction Reduce, spécifie le nombre de tâches Reduce que nous appellerons **r**, le format de lecture du fichier d'entrée, le format de sortie des **r** fichiers Reduce, éventuellement la taille des blocs du fichier d'entrée et le facteur de réplication . Une fois que tout cela est fait et qu'il déclenche l'exécution du job, le jobtracker démarre les **r** tasktrackers qui vont effectuer les **r** tâches Reduce que l'utilisateur a spécifiées ;
  2. Le HDFS découpe le fichier d'entrée en **M blocs** de taille fixe, généralement **64 Mo** par bloc (sauf si l'utilisateur a spécifié une taille de bloc différente à la première étape). Ensuite, le HDFS réplique ces blocs selon le facteur de réplication définie par l'utilisateur (trois par défaut) et les distribue de façon rédondante dans des nœuds différents dans le cluster. Le fait de diviser le fichier d'entrée en blocs de taille fixe permet de répartir de façon équilibrée la charge de traitement parallèle entre les nœuds du cluster, ce qui permet au traitement de s'achever à peu près au même moment dans l'ensemble des nœuds du cluster ;
  3. Par défaut, le jobtracker déclenche M tasktrackers sur les M nœuds de données dans lesquels ont été répartis les M blocs du fichier d'entrée, pour exécuter les tâches Map, soit un tasktracker Map pour chaque bloc de fichier.Chaque tasktracker lit le contenu du bloc de ficher par rapport au format d'entrée spécifié par l'utilisateur, le transforme par le processus de hachage défini dans la fonction Map en paires de clés/valeurs. Ce processus de hachage s'effectue en mémoire locale du nœud ;
  4. Périodiquement, dans chaque nœud, les paires de clés/valeurs sont sérialisées dans un fichier sur le disque dur local du nœud. Ensuite ce fichier est partitionné en **r** régions (correspondant aux **r** tâches Reduce spécifiées par l'utilisateur) par une fonction de hachage qui va assigner à chaque région une clé qui correspond à la tâche Reduce à laquelle elle a été assignée. Les informations sur la localisation de ces régions sont transmises au jobtracker, qui fait suivre ces informations aux r tasktrackers qui vont effectuer les tâches Reduce;
  5. Lorsque les **r** tasktrackers Reduce sont notifiés des informations de localisation, ils utilisent des appels de procédures distantes (protocole RPC) pour lire depuis le disque dur des nœuds sur lesquels les tâches Map se sont exécutées, les régions des fichiers Map leur correspondant. Ensuite, ils les trient par clé. Notez au passage que le tri s'effectue en mode batch dans la mémoire du tasktracker Reduce. Si les données sont trop volumineuses, alors cette étape peut augmenter de façon significative le temps total d'exécution du job;
  6. Les tasktrackers Reduce itèrent à travers toutes les données triées et pour chaque clé unique rencontrée, ils la passent avec sa valeur à la fonction Reduce écrite par l'utilisateur. Les résultats du traitement de la fonction Reduce sont alors sérialisés dans le fichier ri (avec i l'indice de la tâche Reduce) selon le format de sortie spécifié par l'utilisateur. Cette fois-ci, les fichiers ne sont pas sérialisés dans le disque dur du nœud tasktracker, mais dans le HDFS, ceci pour des raisons de résilience (tolérance aux pannes);
  7. Le job s'achève là, à ce stade, les r fichiers Reduce sont disponibles et Hadoop applique en fonction de la demande de l'utilisateur, soit un « Print Écran », soit leur chargement dans un SGBD, soit alors leur passage comme fichiers d'entrée à un autre job MapReduce.

La figure suivante récapitule en visuel ces sept étapes.

[](http://juvenal-chokogoue.developpez.com/tutoriels/hadoop-fonctionnement/images/jobMapReduce.png)Figure 9 : étapes d'exécution d'un job MapReduce dans un cluster Hadoop. La couleur jaune traduit les traitements, la verte représente la RAM, le blanc représente les opérations d'accès à la donnée et les cylindres bleus les fichiers Map

**YARN/MapReduce 2.0**

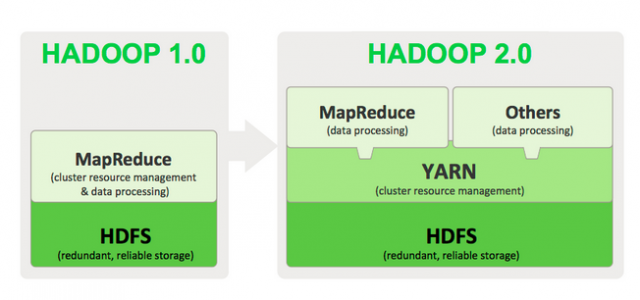
Avant de parler de Yarn, essayons de faire un petit bilan sur ce que nous avons vu !

Nous avons vu que l'algorithme MapReduce permet d'implémenter de nombreux types de traitement en vue de leur parallélisation. Pour autant, tous les problèmes rentrent-ils dans le moule MapReduce, très rigide?

* Non pas forcement et très souvent, si c'est le cas, cela demande beaucoup d'efforts de transformer un algorithme en MapReduce.
* De même, pour traiter des problèmes complexes, les deux étapes MAP et REDUCE ne suffisent pas, il est très souvent nécessaire d'enchaîner des séquences de MapReduce ce qui est très coûteux car cela nécessite de démarrer un job MapReduce à chaque fois.
* Enfin, si on s'interesse un peu plus à l'architecture d'Hadoop, on remarque que le job tracker a une double responsabilité :
  1. Il doit gérer les ressources du cluster.
  2. Il doit ordonnancer les jobs.
* Que se passe-t-il si le Job tracker est défaillant?

Pour répondre à ces différents problèmes, plusieurs améliorations ont été apportées à Hadoop (version 2.x). Notamment, l'architecture d'Hadoop a été modifiée pour introduire YARN : Yet Another Ressource Negociator, **un framework permettant d'exécuter n'importe quel type d'application distribuée sur un cluster Hadoop**, pas uniquement les applications MapReduce.

YARN propose en effet de séparer la gestion des ressources du cluster et la gestion des jobs MapReduce, permettant ainsi de généraliser cette gestion des ressources à d'autres applications. L'idée principale est de considérer que les nœuds ont des ressources (mémoire et CPU) qui seront allouées aux applications quand elles le demandent.

Fig. 10 D'Hadoop 1.X à Hadoop 2.X avec YARN.

Les modifications de base à l’architecture de Hadoop concernent avant tout les deux trackers du MapReduce-Engine, qui n’existent plus en tant que composants uniques dans la version 2 de Hadoop. À la place, le module YARN possède trois nouvelles entités: le ResourceManager, le NodeManager et l’ApplicationMaster.

* Le **resource manager** qui est le chef d'orchestre des ressources du cluster. Il ordonnance les requêtes clients et Driver le cluster par l'intermédiaire de **node managers** qui s'exécutent sur chaque nœud de calcul. Il a donc pour rôle de contrôler toutes les ressources du cluster et l'état des machines qui le constituent. Donc le resourceManager gère le cluster en maximisant l'utilisation de ressources par l'intermédiaire du ResourceScheduler.
* Le NodeManager agit sur chaque nœud du cluster d’ordinateurs. Celui-ci prend en compte la position des esclaves dans l’infrastructure d’Hadoop 2 et procède en tant que destinataire de commandes du ResourceManager. Si un NodeManager est démarré dans un nœud du cluster, celui-ci véhicule l’information au ResourceManager et envoie un signe de vie périodique (*heartbeat*). Chaque NodeManager est responsable des ressources de son propre nœud et en met une partie à disposition du cluster.

Le ResourceScheduler détermine comment les ressources disponibles du cluster seront partagées et dirige la manière dont les ressources sont utilisées dans le cluster.

* L'**application master (AM)** qui est un processus s'exécutant sur toutes les machines esclaves et qui gère, en discussion avec le **resource manager**, les ressources nécessaires au travail soumis.

De même, les fonctionnalités du **task tracker** sont aussi réparties sur une même machine entre:

* Des **containers** qui sont des abstractions de ressources sur un nœud dédiées soit à l'exécution de tâches comme Map et Reduce, soit à l'exécution d'un **application master**.
* Un **node manager** qui héberge des **containers** et gére donc les ressources du nœud. Il est en communication via un heartbeat avec le ressource manager.

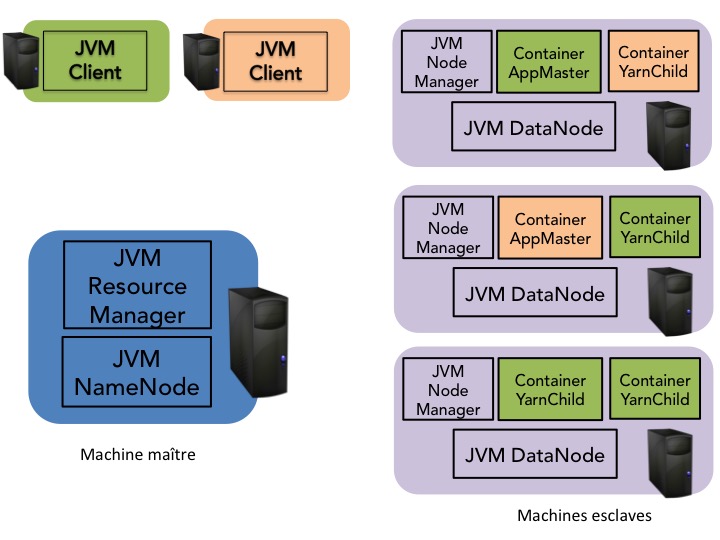


Fig. 11 Architecture maître-esclave de Hadoop avec YARN

Si une application Big Data est exécutée sur Hadoop, **trois acteurs sont impliqués**:

* un client
* le ResourceManager
* un ou plusieurs NodeManagers

Le schéma de soumission et d'exécution d'un job dans cette nouvelle architecture est donc le suivant:

1. le client Hadoop du ResourceManager a pour tâche de démarrer l’application Big Data dans le cluster Hadoop et il copie ses données sur HDFS.
2. Le client soumet le travail à effectuer au **resource manager** sous la forme d'une archive.jar et des noms des fichiers d'entrée et de sortie.
3. Le **resource manager** alloue alors un container pour l'**application master** sur un **node manager**. En d’autres termes: le ResourceManager réserve les ressources du cluster pour l’application et contacte le NodeManager.
4. L'**application master** demande au **resource manager** un ou plusieurs containers avec des préférences de localisation dépendant de la localité des données d'entrée du travail.
5. Le **resource manager** alloue alors un ou plusieurs containers (child) à l'**application master**.
6. L'**application master** choisit parmi la liste des tâches (par exemple Map et Reduce) et démarre une instance de la tâche choisie dans un des **containers** qui lui a été alloué. Il collabore alors avec le **node manager** pour utiliser les ressources acquises. Il communique aussi souvent avec le **resource manager** (message heartbeat) pour la tolérance aux pannes.

Le schéma ci-dessous illustre le schéma simplifié de soumission et d'exécution d'un travail dans Hadoop 2.X avec YARN.

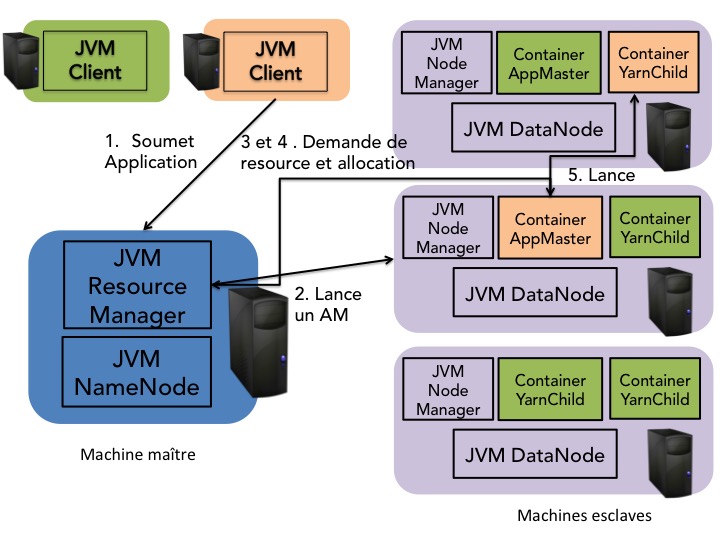
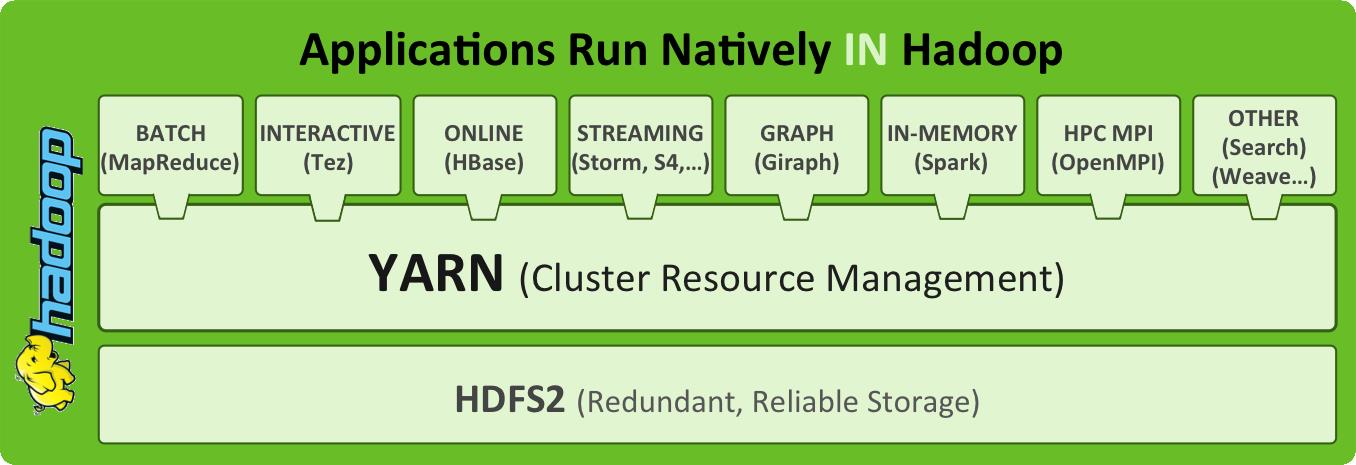


Fig 12 Schéma simplifié de l'exécution d'un travail dans Hadoop 2.X avec YARN.

Malin ! Et le gros avantage est que tout cela ne se limite donc plus à MapReduce et avec ce principe, de nombreuses applications peuvent s'exécuter de manière native sur un même cluster Hadoop parmi lesquelles:

* [Spark](https://spark.apache.org/) : une solution pour le traitemen et l'analyse de données massives.
* [Giraph](https://giraph.apache.org/): une solution pour faire des calculs sur graphes.
* [HBase](https://hbase.apache.org/) : une base de données NoSQL reposant sur HDFS. Nous détaillerons aussi ce type de base de données dans un prochain cours.
* [Tez](https://tez.apache.org/) : un cadre pour l'écriture et l'exécution de traitements modélisés sous la forme de graphes dirigés acycliques (DAG) et qui facilite l'enchaînement des traitements.
* ...

Fig 13 Quelques applications pouvant s'exécuter de manière native sur

Avec la sortie de la **version 2.3 de Hadoop**, le MapReduce-Engine a été retravaillé. Le résultat a donné la méthode de gestion des clusters **YARN/MapReduce 2.0**, dont le management des ressources et la gestion des tâches (Job Scheduling/Monitoring) de MapReduce ont été couplés. Ainsi, le framework offre de nombreuses possibilités en matière de nouveaux modèles de travail et **une large palette d’applications Hadoop pour le Big Data**.

1. **Avantages de Hadoop**

Le framework Hadoop :

* Simple à utiliser et à déployer
* Portable (clusters hétérogènes) et très extensible.
* Le principal atout des clusters Hadoop est leur adéquation à l’analyse de gros volumes de données. Le [Big Data](http://www.lemagit.fr/definition/Big-Data-Gestion) est le plus souvent non structuré et largement distribué.
* Permet à l'utilisateur d'écrire et de tester rapidement des systèmes distribués. Il est efficace, et il distribue automatiquement les données et fonctionne à travers les machines et, à son tour, utilise le parallélisme des cœurs du processeur.
* Hadoop ne s'appuie pas sur le matériel pour assurer la tolérance aux pannes, mais la librairie Hadoop elle-même a été conçue pour détecter et gérer les défaillances au niveau de la couche d'application.
* Les serveurs peuvent être ajoutés ou supprimés dynamiquement du cluster et Hadoop continue de fonctionner sans interruption.
* Un autre gros avantage de Hadoop est que, en plus d'être open source, il est compatible sur toutes les plateformes puisqu'il est basé sur Java.
* De plus, le Big Data est surtout utile s’il est analysé en temps réel ou dans un délai aussi proche que possible du temps réel. Les capacités de traitement en parallèle de Hadoop participent à la rapidité de l’analyse, mais à mesure que le volume de données à analyser augmente, la puissance de traitement du cluster peut devenir insuffisante. Heureusement, l’ajout de noeuds permet de faire évoluer le framework.
* Les clusters Hadoop peuvent s’avérer une solution économique. On peut ainsi construire un cluster puissant sans dépenser une fortune en serveurs.
* Un autre avantage de ces clusters est leur résistance aux pannes. Quand un élément de donnée est envoyé à un noeud pour analyse, les données sont également copiées vers d’autres noeuds: en cas de défaillance sur un noeud, d’autres copies existent ailleurs dans le cluster et il est toujours possible de les analyser.

1. **Inconvénients de Hadoop**

Malgré ses nombreux avantages, la solution des clusters Hadoop ne convient pas aux besoins d’analyse de données de toutes les organisations.

* Pour les petits volumes de données, ses bénéfices seront inexistants, même s’il est besoin d’une analyse poussée de ces dernières.
* Un autre inconvénient des clusters Hadoop réside dans le principe même de la solution de clustering, qui suppose que les données puissent être fractionnées et analysées par des processus parallèles exécutés sur des noeuds de cluster distincts. Si l’analyse ne peut pas se dérouler dans un environnement de traitement parallèle, un cluster Hadoop n’est tout simplement pas le bon outil pour ce travail.
* Mais le plus grand obstacle à l’utilisation d’un cluster Hadoop reste probablement la courbe d’apprentissage considérable associée à la construction, au fonctionnement et à la prise en charge de la solution.
* Optimisé pour des lectures concurrentes; sensiblement moins performant pour des écritures concurrentes.

1. **Modes de fonctionnement Hadoop**

Une fois que vous avez téléchargé Hadoop, vous pouvez utiliser votre cluster Hadoop dans l'un des trois modes suivants:

**Mode local / autonome:** Après avoir téléchargé Hadoop dans votre système, par défaut, il est configuré en mode standard et peut être exécuté en tant que **processus java unique**.

**Pseudo Distributed Mode:** Il s'agit d'une simulation distribuée sur une seule machine. **Chaque démon Hadoop**, tel que hdfs, yarn, MapReduce etc., fonctionnera comme un **processus java séparé**. Ce mode est utile pour le développement.

**Mode entièrement distribué:** Ce mode est entièrement distribué avec au moins deux machines en tant que cluster.

1. **Les commandes HDFS les plus utilisées**

|  |  |
| --- | --- |
| Commande | Explication et exemples |
| hadoop fs -mkdir | **Créer un dossier dans HDFS**  hadoop fs -mkdir –p /user/monDossier |
| hadoop fs –ls | **Lister le contenu d’un dossier**  hadoop fs -ls /user/monDossier |
| hadoop fs -lsr | **Liste recursive du contenu d’un dossier**  hadoop fs -ls /user/monDossier |
| hadoop fs -put | **Charger un ou plusieurs fichiers du local à HDFS**  hadoop fs -put /home/monFichier.txt /user/monDossier |
| hadoop fs -get | **Exporter un ou plusieurs fichiers de HDFS au local**  hadoop fs -get /user/monDossier/monFichier.txt /home |
| hadoop fs -cp | **Copier un ou plusieurs fichiers dans HDFS**  hadoop fs -cp /user/monDossier1/monFichier.txt /user/monDossier2 |
| hadoop fs -mv | **Déplacer un ou plusieurs fichiers dans HDFS**  hadoop fs -mv /user/monDossier1/monFichier.txt /user/monDossier2 |
| hadoop fs -copyFromLocal | **Charger un ou plusieurs fichiers du local à HDFS :** Cette commande est similaire à –put  hadoop fs -copyFromLocal /home/monFichier.txt /user/monDossier |
| hadoop fs -copyToLocal | **Exporter un ou plusieurs fichiers de HDFS au local**: Cette commande est similaire à -get  hadoop fs -copyToLocal /user/monDossier/monFichier.txt /home |
| hadoop fs -cat <Path[Filename]> | **Afficher le contenu d’un fichier**  hadoop fs -cat /user/monFichier.txt |
| hadoop fs -tail <Path[Filename]> | **Afficher les dernières lignes d’un fichier**  hadoop fs -tail /user/monFichier.txt |
| hadoop fs -rm | **Supprimer un fichier dans HDFS**  hadoop fs -rm /user/monFichier.txt |
| hadoop fs -rmr | **Suppression récursive dans HDFS**  hadoop fs -rmr /user/ |
| hadoop fs -help | voir la description d’une commande  hadoop fs -help stat |
| hadoop fs -touchz | **Créer un fichier vide** |
| hadoop fs -chmod | **Modification des permissions** |
| hadoop fs –chown | **Modification du propriétaire** |
| hadoop fs -chgrp | **Modification du groupe primaire** |

1. **Conclusion**

Hadoop n'utilise aucun principe foncièrement nouveau; il offre en revanche une très forte simplicité et souplesse de déploiement inconnues jusqu'à présent pour l'exécution facile de tâches parallèles variées.

Grâce à Hadoop, même des structures limitées en taille/ressources peuvent facilement avoir accès à de fortes capacités de calcul: déploiement à bas coût de clusters en interne ou location de temps d'exécution via les services de cloud computing.

* + - 1. **Hadoop - Configuration de l'environnement**

Hadoop est supporté par la plateforme GNU / Linux et ses goûts. Par conséquent, nous devons installer un système d'exploitation Linux pour configurer l'environnement Hadoop. Dans le cas où vous avez un système d'exploitation autre que Linux, vous pouvez installer un logiciel Virtualbox et avoir Linux dans la Virtualbox ou installer Cygwin (une collection de logiciels open source permettant à différentes versions de Windows d'émuler un système Unix.)

**Étapes pour les applications Hadoop**

**Partie I Hadoop standalone mode**

Étape 1: Téléchargez VirtualBox pour Linux.

Étape 2: Télécharger Ubuntu LTS en cours d'exécution sur la machine virtuelle / 64 bits

Étape 3: Téléchargez Hadoop-1.2.1

Étape 4: téléchargez java JDK et JRE

Étape 5: edit le fichier /etc/profile

Etape 6: Dire à Linux où sont stockés les fichiers java

Étape 7: téléchargez netbeans (.sh)

Étape 8: tester si la configuration réussit

**Étape 1: Téléchargez VirtualBox pour Linux.**

Une version de téléchargement gratuite de VirtualBox pour Linux peut être téléchargée à partir d'Oracle VirtualBox.

**Note:** Remarque: Si vous ouvrez VirtualBox pour la première fois et que vous recevez l'erreur suivante:

“hardware acceleration is not available, this means that the Bios does not support virtualization”

* Entrer au BIOS (dans windows 10: settings/update & security/recovery/restart now/troubleshoot/advanced options/ UEFI firmware settings/ restart now/ setup utility)
* System configuration/virtualization technology/ enable/Save changes

**Note:** Si vous ouvrez VirtualBox pour la première fois et que vous recevez l'erreur:

“Asking all remaining processes to terminate”

* Press CTRL + ALT + T

**Etape 2:** **Télécharger Ubuntu LTS** en cours d'exécution sur la machine virtuelle/64 bits puis **l'installer**

* + 1. Téléchargez l'image Ubuntu **.ISO** (LTS 64 bits), puis:
* ouvrez virtualbox/new (pour le nom entrez Ubuntu, pour le type entrez linux, pour la version entrez votre version)
* Allouer de la mémoire (2048) / Créer un disque dur virtuel / VDI / allocate dynamically 20GB
  + 1. Installer Ubuntu LTS

• Démarrer Ubuntu

* Dans settings storage, cliquer **Empty**/ cliquer dans icon pour choisir startup disk
  1. selectionner “iso Ubuntu” fichier à partir de download repertoire
  2. Choisir install Ubuntu
  3. installer third party software
  4. erase disk et installer Ubuntu console puis suivre les instructions.

**Etape 3:** **Télécharger Hadoop-1.2.1 (Hadoop-1.2.1.tar.gz)**

(<http://www-eu.apache.org/dist/hadoop/common>)

* Créer un répertoire “hadoop” dans le répertoire home
* Ouvrir le terminal et copier Hadoop-1.2.1.tar.gz du répertoire downloads au répertoire hadoop
* Changer répertoire à Hadoop et tar le fichier.

|  |
| --- |
| * 1. **cd hadoop**   2. **tar xzf hadoop\*** |

**Etape 4:** **Télécharger java JDK et JRE**

* Télécharger JDK et JRE à partir de java SE environment; ceux qui se terminent par tar.gz pour Linux 64 bit. Choisir “save file”.
* Créer un répertoire “java” où vous voulez installer les fichiers java et les copier à ce repertoire.

|  |
| --- |
| * 1. **sudo mkdir -p /usr/local/java**   2. **cd Downloads**   3. **sudo cp –r jre-\*.tar.gz /usr/local/java**   4. **sudo cp –r jdk-\*.tar.gz /usr/local/java** |

* Changer répertoire à /usr/local/java et tar ces fichiers.

|  |
| --- |
| * 1. **cd /usr/local/java**   2. **sudo tar xvzf jdk\*.tar.gz**   3. **sudo tar xvzf jre\*.tar.gz** |

**Etape 5:** Edit le fichier profile et ajouter à la fin les variables java\_home et Hadoop\_install et les ajouter aux paths.

|  |
| --- |
| * **$ sudo nano /etc/profile**   Add the following lines at the end of the file |
| * **JAVA\_HOME=/usr/local/java/jdk-9.0.4** * **PATH=$PATH:$JAVA\_HOME/bin** * **JRE\_HOME=/usr/local/java/jre-9.0.4** * **PATH=$PATH:$JRE\_HOME/bin** * **HADOOP\_INSTALL=/home/zeinab/hadoop/hadoop-1.2.1** * **PATH=$PATH:$HADOOP\_INSTALL/bin** * **export JAVA\_HOME** * **export JRE\_HOME** * **export PATH** |

**Etape 6:** Dire à Linux où sont stockés les fichiers java

|  |
| --- |
| * **sudo update-alternatives --install “/usr/bin/javac” “javac” “/usr/local/java/jdk-9.0.4/bin/javac” 1** * **sudo update-alternatives --install “/usr/bin/java” “java” “/usr/local/java/jre-9.0.4/bin/java” 1** * **sudo update-alternatives --install “/usr/bin/javaws” “javaws” “/usr/local/java/jre-9.0.4/bin/javaws” 1** * **sudo update-alternatives --set java “/usr/local/java/jdk-9.0.4/bin/javac”** * **sudo update-alternatives --set javac“/usr/local/java/jre-9.0.4/bin/java”** * **sudo update-alternatives --set javaws “/usr/local/java/jre-9.0.4/bin/javaws”** |

* refresh le fichier profile

|  |
| --- |
| * 1. **. /etc/profile**. |

* Ensuite, les commandes suivantes doivent fonctionner

|  |
| --- |
| * **java –version** * **sudo echo $JAVA\_HOME** |

**Etape 7:** Télécharger netbeans (netbeans-8.2-javaee-linux.sh) puis l’installer

Pour installer NetBeans:

* Créer un répertoire netbeans sous ton “home directory”
* Copier le fichier “netbeans-8.2-javaee-linux.sh” à partir du répertoire “downloads” au répertoire “netbeans”
* Ajouter la permission “x” sur ce fichier.
* Exécuter le fichier

|  |
| --- |
| **$ chmod 755 netbeans-8.2-javaee-linux.sh**  **$ sudo sh netbeans-8.2-javaee-linux.sh**  Ou  $ cd Downloads  $ **./netbeans-8.2-javaee-linux.sh** |

**Note:** Si vous n’arrivez pas à l’installer, exécuter la commande suivante:

|  |
| --- |
| **$ sudo apt install netbeans** |

**Note:** si vous recevez le message: “an instance of the program cannot access specified user directory”

|  |
| --- |
| **$ sudo chown –R zeinab /home/zeinab/.netbeans**  **$ sudo chown –R zeinab ~/.cache/netbeans/**  **$ sudo chgrp –R zeinab ~/.cache/netbeans/** |

**Note:** pour ouvrir netbeans à partir de la ligne de commande, exécuter la commande suivante:

|  |
| --- |
| **$ sudo /bin/sh “/usr/local/netbeans-8.2/bin/netbeans”** |

**Note:** si votre installation n’est pas propre et en uninstall netbeans vous recevez le message “cannot create folder in /root/.nbi”

|  |
| --- |
| **$ cd /usr/local/netbeans8.2**  **$ su**  **$ sh uninstall.sh**  Changer home directory et supprimer les répertoires: \*\*netbeans8.\*, .nbi, .netbeans\*\* |

**Etape 8:** À ce stade, nous avons fait une configuration Hadoop pour autonome (**standalone)**. Par défaut, Hadoop est configuré pour s'exécuter en mode non distribué, en tant que processus Java unique (**single Java process)**. Pour tester si la configuration a réussi

|  |
| --- |
| * **$ cd $HADOOP\_INSTALL** (où vous avez installer Hadoop) * **$ sudo mkdir input** * Copier le répertoire conf pour l’utiliser comme input par l’une des commandes suivantes * **$ sudo cp conf/\*.xml input ou $~/hadoop/hadoop-1.2.1/bin/hadoop fs –put conf input** * run an example:   **$ bin/hadoop jar hadoop-examples-\*.jar grep input output 'dfs[a-z.]+'**  (finds and display every match of the given regular expression. Output is written to the given output directory) |

Hadoop fonctionne correctement si les fichiers **\_success** and **part-r-00000** sont présent quand nous exécutons la commande:

|  |
| --- |
| * **$ sudo ls output/\*.** |

**Pour exécuter votre version de WordCount écrite sous NetBeans:**

* changer répertoire là où vous avez enregistrer votre programme (program path)
* télécharger à partir de apache les fichiers suivants et **unzip** les fichiers avec l’extension .jar.zip :
  + hadoop-core-1.1.2.jar.zip
  + commons-logging-1.2.1.1.jar
  + commons-configuration-1.2.jar
  + commons-lang-2.1.jar
  + Jackson-mapper-asl-1.8.8-sources.jar
  + Commons-httpclient=3.1.jar
  + Jackson-mapper-asl-1.9.13.jar
  + Jackson-core-asl-1.9.13.jar
* ajouter les fichiers .jar à la librairie du projet

|  |
| --- |
| * **right click le nom du projet/properties/librairies/add jar files** |

* compiler le programme en chosissant du menu principal: **run/clean and build**

alors NetBeans crée le repertoire **“dist/lib**” où il ajoute les fichiers .jar, les librairies et crée le fichier “**WordCount.jar**” dans le repertoire **dist**

* exécuter le programme.

|  |
| --- |
| /home/zeinab/NetBeansProjects/WordCount/src**$ ~/hadoop/hadoop-1.2.1/bin/hadoop jar dist/WordCount.jar /input /output** |

**Partie II Hadoop pseudo-distributed mode**

Même 7 étapes que le mode standard.

Etape 8: Accès SSH (ssh localhost)

Étape 9: Modifier les fichiers de configuration (core-site.xml, mapred-site.xml, mapred-site.xml, hadoop-env.sh)

Étape 10: Formatez le namenode

Étape 11: Démarrer les processus

Étape 12: Créez les fichiers d'entrée et les copiez dans le système de fichiers distribué

Etape 13: Exécuter quelques exemples fournis ou votre propre application

**Étape 8:** Accès SSH

* En outre, nous installons ssh

|  |
| --- |
| **$ sudo apt-get install ssh**  **$ sudo apt-get install rsync**  (rsync est un utilitaire pour transférer et synchroniser efficacement des fichiers sur des systèmes informatiques). |

* Générer une clé ssh pour l’utilisateur «zeinab» et l'enregistrer dans le fichier **id\_rsa** in the folder ~/.ssh

|  |
| --- |
| **$ su – zeinab**  **$ ssh-keygen -t rsa -P ‘’ –f ~/.ssh/id\_rsa** |

* Activez l'accès SSH à votre machine locale avec cette clé nouvellement créée

|  |
| --- |
| **$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys** |

* Configuration sans passsh pour ssh
  + Testez la configuration SSH en vous connectant à votre machine locale avec l'utilisateur zeinab. Cette étape est également nécessaire pour enregistrer l'empreinte de la clé hôte de votre machine locale dans le fichier **known\_hosts** de l’utilisateur zeinab

|  |
| --- |
| **$ssh localhost** |

**Note:** Si la connexion SSH échoue, les conseils généraux suivants peuvent aider:

* Activez le débogage avec **ssh -vvv localhost** et examinez l'erreur en détail.
* Vérifiez la configuration du serveur SSH dans /etc/ssh/sshd\_config, en particulier les options PubkeyAuthentication (qui doit être définie sur yes) et AllowUsers (si cette option est active, ajoutez l'utilisateur zeinab). Si vous avez apporté des modifications au fichier de configuration du serveur SSH, vous pouvez forcer un réchargement de la configuration avec sudo /etc/init.d/ssh reload

**Étape 9:** **Modifier les fichiers de configuration** suivants qui existent dans le répertoire hadoop/Hadoop-1.2.1/conf

|  |  |
| --- | --- |
| **core-site.xml** | **mapred-site.xml** |
| <configuration>  <property>  <name>fs.default.name</name>  <value>hdfs://localhost:9000</value>  </property>  </configuration> | <configuration>  <property>  <name>mapred.job.tracker</name>  <value> localhost:9001</value>  </property>  </configuration> |
| **hdfs-site.xml** | **hadoop-env.sh** |
| <configuration>  <property>  <name>dfs.replication</name>  <value>1</value>  </property>  </configuration> | Set JAVA\_HOME :  export JAVA\_HOME=/usr/local/java/jdk-9.0.40 |

**Étape 10:** **Formatez le namenode** (formatez le système de fichiers qui initialise simplement le répertoire spécifié par la variable dfs.name.dir)

|  |
| --- |
| **$ bin/hadoop namenode –format** |

**Étape 11:** **Démarrer les processus**

|  |
| --- |
| **$ bin/start-all.sh**  les processus pour namenode, datanode, nom de noeud secondaire. Jobtracker, tasktracker doit commencer.  pour vérifier si les processus Hadoop attendus sont en cours d'exécution, exécutez la commande  **$ jps**  Vous pouvez également vérifier avec netstat si Hadoop écoute sur les ports configurés  **$ sudo netstat -plten | grep java** |

Parcourez l'interface Web pour le NameNode et le JobTracker; Par défaut, ils sont disponibles à:

* **NameNode -** [**http://localhost:50070/**](http://localhost:50070/)
* **JobTracker -** [**http://localhost:50030/**](http://localhost:50030/)
* **TaskTracker – http://localhost:50060**

Ces interfaces Web fournissent des informations concises sur ce qui se passe dans votre cluster Hadoop.

**Étape 12: Copiez les fichiers d'entrée dans le système de fichiers distribué**

|  |
| --- |
| **$ bin/hadoop fs –put conf /input**  Ici le répertoire conf est copié au repertoire /input du système de fichiers |

**Etape 11:** Exécuter quelques exemples fournis

|  |
| --- |
| $ bin/hadoop jar hadoop-example-\*.jar grep input output ‘dfs[a-z.]+’ |

**Note:** si vous recevez un message d’erreur “output directory /output already exists”

|  |
| --- |
| **$ bin/hadoop fs –rmr /output** |

**Etape 12: Examiner les fichiers de sortie du système de fichiers distribué**

|  |
| --- |
| sur système de fichiers distribué  **$ bin/hadoop fs –cat output/\***  Ou dans le système de fichiers local  **$ bin/hadoop fs –get /output output**  **$ cat output/\*** |

**Étape 13: arrêter les daemons**

|  |
| --- |
| **$ bin/hadoop stop-all.sh** |

**Note:** Si vous laissez le fichier core-site.xml par défaut, Hadoop résout toutes les adresses IP et tous les noms de machines du cluster en tant que 127.0.0.1 localhost

**Partie III Distributed mode**

Les 7 premières étapes sont similaires au mode pseudo-distributed mode.

Etape 8: Configuration du réseau

* + - Ajouter un nouveau Adapter (host-only)
    - Sur toutes les machines, modifier les fichiers: /etc/network/interfaces, /etc/hosts, /etc/hostname)
    - Sur la machine maître modifier les fichiers: masters et slaves
    - s’assurer que toutes les machines sont connectées (ping)

Etape 9: Accès SSH (ssh de la machine locale à toutes les machines du cluster)

Étape 10: sur toutesc les machinesm modifier les fichiers de configuration (core-site.xml, mapred-site.xml, mapred-site.xml, hadoop-env.sh)

Étape 11: Formatez le namenode

Étape 12: Démarrer les processus

Étape 13: Créez les fichiers d'entrée et les copiez dans le système de fichiers distribué

Etape 14: Exécuter quelques exemples fournis ou votre propre application

**Etape 6: Configuration du réseau (Networking)**

Sur toutes les machines, appliquer le suivant:

* Ajouter toutes les machines dans le **même réseau**. (192.168.56.x)
* Choisir du menu principal: **Machine/Settings/network**
* Sélectionner: **Adapter2/enable/host-only network**
* Exécuter la commande **ifconfig –a** pour voir l’addresse statique associée à l’adapter 2 puis la commande **route –n** pour voir le gateway.
* **System settings/network/wired/options/ipv4 settings/manual/add**
  1. Remplir ip address, netmask, gateway et DNS servers.
  2. Save
  3. Presser pour avoir “off”, attendez un moment, puis presser pour avoir “on” (il faut voir les settings qu’on a ajouté)

**Note:** pour faire enable network

|  |
| --- |
| $ sudo dhclient eth0 |

**Note:** si en cliquant **options** de la connection, vous recevez le message “could not find a connection with UUID null” executez les commandes suivantes:

|  |
| --- |
| **$ sudo systemctl stop NetworkMananger.service**  **$ sudo systemctl start NetworkMananger.service** |

**Note:** si en cliquant une connection, on voit “unmanaged”

|  |
| --- |
| **$ sudo nano /etc/NetworkManager/NetworkManager.conf**  Changer managed=false à **managed=true**  **$ sudo service network-manager restart** |

* Modifier les fichiers suivants:
  + **/etc/network/interfaces**
  + **/etc/hosts**
  + **/etc/hostname**
    1. Pour chaque machine, ajouter les lignes suivantes dans le fichier **/etc/network/interfaces**.

|  |
| --- |
| * **auto eth1** * **iface eth1 net static** * **address 192.168.56.X** (l’adresse qu’on a choisi pour cette machine) * **netmask 255.255.255.0** * **broadcast 192.168.56.255** |

Enregistrer le fichier puis exécuter les commandes suivantes:

|  |
| --- |
| $ **sudo** **service networking restart** pour redémarrer le service networking  $ **sudo** **ifconfig eth1 down**  $ **sudo ifconfig eth1 up**  $ **ifconfig** pour voir la nouvelle adresse |

**Note:** Si l’adresse n’est pas modifiée, alors exécuter la commande:

|  |
| --- |
| **sudo /etc/init.d/networking restart**  ou   * $ **sudo ip addr flush eth1** * **$ sudo systemctl restart networking.service**   Puis vérifier qu’elle est correcte:  $ ip add |

* Mettre à jour le fichier **/etc/hostname**
  1. Pour la machine master modifier le nom de la machine à “Hmaster”
  2. Pour les machines slaves modifier le nom de chaque machine à “Hslave1” pour la première et “Hslave2” pour la deuxième
* Sur **toutes les machines**, ajouter les lignes suivantes dans le fichier **/etc/hosts**

192.168.56.103 Hmaster

192.168.56.104 Hslave1

192.168.56.105 Hslave2

* **Seulement** sur la machine **master**
  + Dans le fichier “**masters**” du repertoire “~/Hadoop/Hadoop-x.x.x/conf”, ajouter le nom de la machine master (Hmaster) à la place de “localhost”. Ce fichier définit sur quelles machines Hadoop va démarrer *secondaryNameNodes* sur notre multi-node cluster. Dans notre cas c’est la machine “”Hmaster.
  + dans le fichier “**slaves**” du repertoire “~/Hadoop/Hadoop-x.x.x/conf”, ajouter les noms des machines esclaves (Hslave1, Hslave2) à la place de “localhost”
* pour s’assurer que toutes les machines sont connectées, sur chacune d’elles faire un **ping** vers les autres machines

|  |
| --- |
| **$ ping 192.168.56.103**  **$ ping 192.168.56.104**  **$ ping 192.168.56.105** |

**Etape 7: Accès SSH**

Hadoop requiert un accès SSH pour gérer ses nœuds, c'est-à-dire les machines distantes et votre machine locale.

L'utilisateur “zeinab” du maître (zeinab@Hmaster) doit pouvoir se connecter: à son propre compte utilisateur sur le maître - c'est-à-dire zeinab@Hmaster et aussi au compte utilisateur “zeinab” sur les esclaves (zeinab@Hslave1 zeinab@Hslave1) via un login SSH sans mot de passe.

* Télécharger SSH puis l’installer

|  |
| --- |
| **$ sudo apt-get install ssh**  **$ sudo apt-get install rsync** |

* Générer une clé ssh pour l’utilisateur zeinab et l'enregistrer dans le fichier id\_rsa in the folder ~/.ssh

|  |
| --- |
| **$ su – zeinab**  **$ ssh-keygen -t rsa -P ‘’” –f ~/.ssh/id\_rsa** |

* Distribuer la clé publique de “zeinab@Hmaster”

il suffit d'ajouter la clé publique SSH de “zeinab@Hmaster” au fichier authorized\_keys de zeinab@Hslave1.

|  |
| --- |
| **zeinab@master:~$ sudo cat ~/.ssh/id\_rsa.pub >> ~authorized\_keys** |

Cette commande nous demande le mot de passe de l’utilisateur zeinab sur la machine esclave, puis copie la clé publique SSH pour vous, en créant le répertoire correct et en fixant les permissions.

* Configuration sans passsh pour ssh
  + Testez la configuration SSH en vous connectant à votre machine locale avec l'utilisateur zeinab. Cette étape est également nécessaire pour enregistrer l'empreinte de la clé hôte de votre machine locale dans le fichier **known\_hosts** de l’utilisateur zeinab. De même connecter avec l’utilisateur zeinab du master aux machines esclaves.

|  |
| --- |
| zeinab@master:~$ **ssh master**  zeinab@master:~$ **ssh Hslave1** la ligne de commande sera : zeinab@Hslave1~$  zeinab@master:~$ **ssh Hslave2** la ligne de commande sera : zeinab@Hslave2~$ |

**Note:** si vous recevez l’erreur: “RSA host key for pong has changed and you have requested strict checking. Host key verification failed”.

|  |
| --- |
| ssh-keygen -R <host> ssh-keygen -R 192.168.56.104 |

Ce qui supprime la key du fichier known\_hosts. La sortie de la commande nous indique le numéro de la ligne à supprimer. Si on recoit:

“sed '4d' -i /var/lib/sss/pubconf/known\_hosts. The 4d is on the account of Offending RSA ...known\_hosts:4”

Donc il faut supprimer la ligne 4 du fichier **Known\_hosts**

**Note:** pour prendre une session root utiiser: **sudo –i**

**Note:** pour utiliser gedit avec superuser permissions: **gksudo gedit /etc/pm/config.d/config**

**III- Application Java:** Le Programme Word Count avec MapReduce

Dans Hadoop, MapReduce est un calcul qui décompose de gros travaux de manipulation en tâches individuelles pouvant être exécutées en parallèle sur un cluster de serveurs. Les résultats des tâches peuvent être regroupés pour calculer les résultats finaux.

|  |
| --- |
| import java.io.IOException;  import java.util.StringTokenizer;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class WordCount {  public static class Map extends Mapper<Object, Text, Text, IntWritable>{    //hadoop supported data types  private final static IntWritable **one** = new IntWritable(1);  private Text **word** = new Text();  //map method that performs the tokenizer job and framing the initial key value pairs  public void map(Object key, Text value, Context context)  throws IOException, InterruptedException {  //taking one line at a time and tokenizing the same  StringTokenizer itr = new StringTokenizer(value.toString());  //iterating through all the words available in that line and forming the key value pair  while (itr.hasMoreTokens()) {  word.set(itr.nextToken());  //sending to output collector which in turn passes the same to reducer  context.write(word, one);  }  }  }  public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable> {  private IntWritable **result** = new IntWritable();  //reduce method accepts the Key Value pairs from mappers, do the aggregation  //based on keys and produce the final out put  public void reduce(Text key, Iterable<IntWritable> values, Context context)  throws IOException, InterruptedException {  int sum = 0;  /\*iterates through all the values available with a key and add them together  /\*and give the final result as the key and sum of its values\*/  for (IntWritable val : values) {  sum += val.get();  }  result.set(sum);  context.write(key, result);  }  }  public static void main(String[] args) throws Exception {  //creating a JobConf object and assigning a job name for identification purposes  Configuration conf = new Configuration();  Job job = Job.getInstance(conf, "word count");  job.setJarByClass(WordCount.class);  //Providing the mapper and reducer class names  job.setMapperClass(Map.class);  job.setCombinerClass(Reduce.class);  job.setReducerClass(Reduce.class);  //Setting Job object with the Data Type of output Key and Value  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(IntWritable.class);  //the hdfs input and output directory to be fetched from the command line  FileInputFormat.addInputPath(job, new Path(args[0]));  FileOutputFormat.setOutputPath(job, new Path(args[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

Quelles sont les exigences minimales?

1. Entrer des fichiers texte - n'importe quel fichier texte

2. Les classes de mappeur, de réducteur et de Driver pour traiter les fichiers d'entrée

Comment ça marche

L'opération de comptage de mots se déroule en deux étapes: une phase de mappage et une phase de réduction. Dans la phase de mappage, le test est d'abord segmenté en mots, puis nous formons une paire de valeurs clés avec ces mots, la clé étant le mot lui-même et la valeur '1'. Par exemple, considérez la phrase :

“tring tring the phone rings”

En phase de carte, la phrase serait divisée en mots et formerait la paire de valeurs de clé initiale

<tring,1>

<tring,1>

<the,1>

<phone,1>

<rings,1>

Dans la phase de réduction, les clés sont regroupées et les valeurs des clés similaires sont ajoutées. Donc, ici, il n'y a qu'une paire de clés similaires 'tring' les valeurs de ces clés seraient ajoutées de sorte que les paires de valeur de clé de sortie seraient

<tring,2>

<the,1>

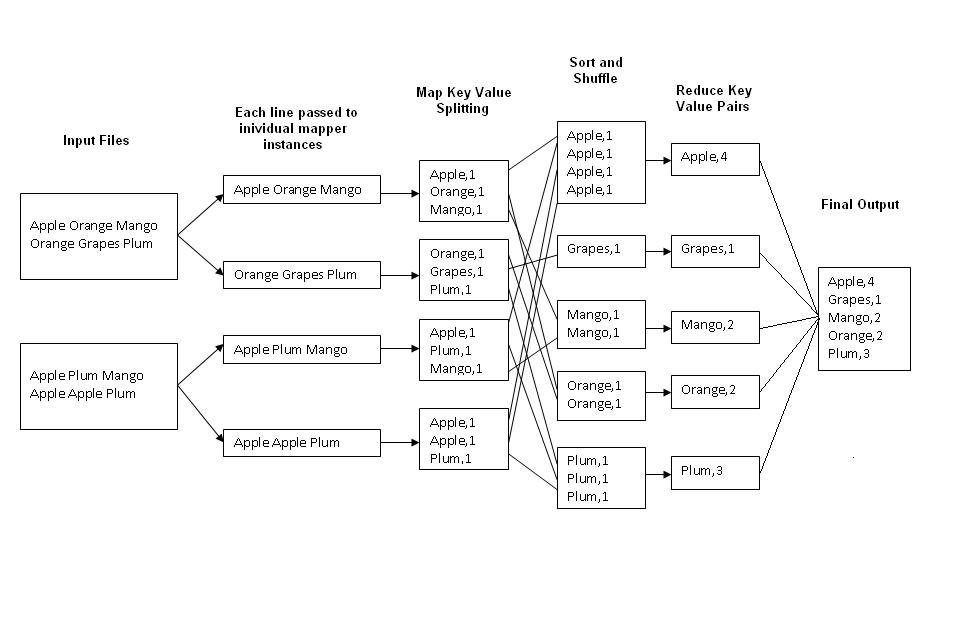
<phone,1>

<rings,1>

Cela donnerait le nombre d'occurrence de chaque mot dans l'entrée. Ainsi, réduire forme une phase d'agrégation pour les clés.

Le point à noter ici est que d'abord la classe de mappeur s'exécute complètement sur l'ensemble de données en séparant les mots et en formant les paires de valeurs de clés initiales. Ce n'est qu'après que tout ce processus est terminé que le réducteur démarre. Dites si nous avons un total de 10 lignes dans nos fichiers d'entrée combinés ensemble, d'abord les 10 lignes sont tokenized et les paires de valeurs clés sont formées en parallèle, seulement après que l'agrégation / réducteur commencerait son opération.

La figure ci-dessous apporterait plus de lumière à votre compréhension

Maintenant, en ce qui concerne le côté pratique de la mise en œuvre, nous avons besoin de notre fichier d'entrée et de notre carte pour réduire le volume du programme. Dans une carte commune réduire le processus deux méthodes font le travail clé à savoir la carte et réduire, la méthode principale déclencherait la carte et réduirait les méthodes. Pour plus de commodité et de lisibilité, il est préférable d'inclure les méthodes map, reduce et main dans trois fichiers de classe différents. Nous regardons les 3 fichiers dont nous avons besoin pour effectuer le travail de comptage de mots

La fonctionnalité de la méthode Map est la suivante

1. Créez une variable IntWritable 'one' avec la valeur 1

2. Convertissez la ligne d'entrée dans le type de texte en chaîne

3. Utilisez un tokenizer pour diviser la ligne en mots

4. Itérer à travers chaque mot et une paire de valeurs clés:

a- Assignez chaque travail du tokenizer (de type String) à un 'mot' Text

b. Former des paires de valeurs clés pour chaque mot comme <word, one> et le pousser vers le collecteur de sortie

La fonctionnalité de la méthode de réduction est la suivante

* 1. Initialiser une variable 'sum' comme 0
  2. Itérer à travers toutes les valeurs par rapport à une clé et les additionner.
  3. Pousser vers le collecteur de sortie la clé et la somme obtenue en tant que valeur

<https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html>

<https://www.google.com.lb/search?q=google+translate&oq=google+trans&aqs=chrome.0.0j69i57j0l4.4496j0j7&sourceid=chrome&ie=UTF-8>