# Chapitre 1 :

# Introduction au Big Data

## 1.1. Introduction au Big Data

Le terme Anglais « Big Data » a été proposé par John Mashey, alors expert scientifique chez Silicon Graphics. Les big data, (grosses données, megadonnées, ou données massives), désignent 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.

La notion de Big Data est un concept qui s'est popularisé en 2012 pour traduire le fait que les entreprises sont confrontées à des volumes de données à traiter de plus en plus considérables et présentant un fort enjeux commercial et marketing.

Ces Grosses Données en deviennent difficiles à travailler avec des outils classiques de gestion de base de données. Il s'agit donc d'un ensemble de **technologies**, **d'architecture**, d'**outils** et de **procédures** permettant à une organisation de **très rapidement capter**, **traiter** et **analyser** de **larges quantités** et **contenus hétérogènes** et **changeants**, et d'en extraire les **informations pertinentes** à un **coût accessible**.

## 1.2. Les frontières du Big Data

"Big Data" est un concept très difficile à définir avec précision, puisque la notion même de "big"en termes de volumétrie des données varie d'une entreprise à l'autre. En Règle générale, on considère du BigData quand le traitement devient trop long pour une seule machine.

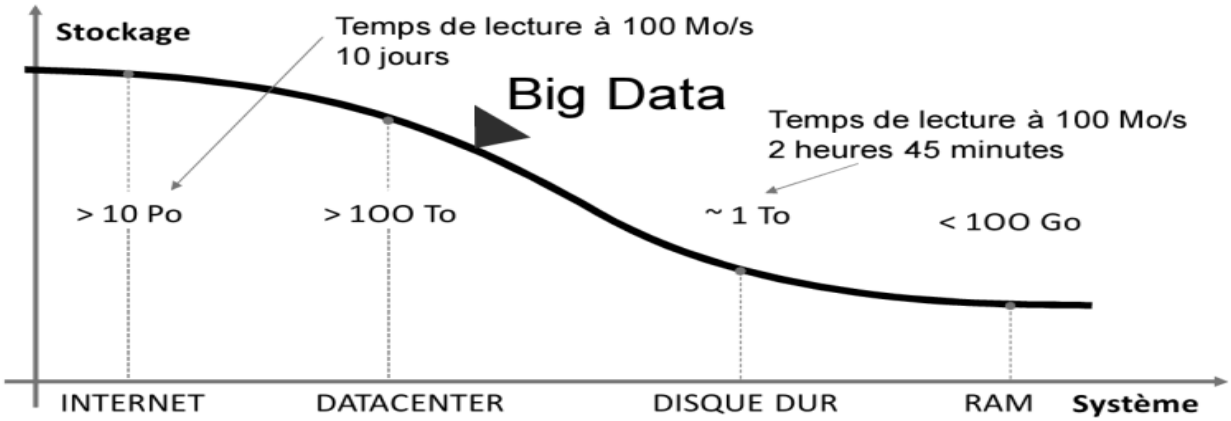



Fig. 1.1. Les frontières du Big Data

Retenons que :

1 Megaoctet = $10^6$ octets

1 Gigaoctet = $10^9$ octets

1 Teraoctet = $10^{12}$ octets

1 Petaoctet = $10^{15}$ octets

1 Exaoctet = $10^{18}$ octets

1Zttaoctet = $10^{21}$ octets

### 1.3. Spécificités du Big Data

Comme l'expression l'indique, le « Big » Data se caractérise par la taille ou la volumétrie des informations. Mais d'autres attributs, notamment la **vitesse** et le **type** de données, sont aussi à considérer. En ce qui concerne le **type**, le Big Data est souvent rattaché à du contenu non structuré ou semi-structuré, ce qui peut représenter un défi pour les environnements classiques de stockage relationnel et de calcul.

#### 1.3.1. Volume

Internet étant le principal acteur dans l'avènement du Big Data, dans une minute Internet : on a 30h Vidéo téléchargées, 204 Millions d'emails échangés et 300 milles Tweets envoyés, …

#### 1.3.2. Variété

Dans un contexte Big Data les données sont sous forme structurée (bases de données structurée, feuilles de calcul venant de tableur,…) et non structurée (textes, sons, images, vidéos, données de capteurs, fichiers journaux, medias sociaux, signaux,…) qui doivent faire l'objet d'une analyse collective. On estime à 80% les données non structurées pour 20% des données structurées.

#### 1.3.3. Vitesse

Fait référence à la vitesse à laquelle de nouvelles données sont générées et la vitesse à laquelle les données sont traitées par le système pour être bien analysées.

#### 1.3.4. Véracité

La véracité fait référence à la qualité de la fiabilité et la confiance des données (données bruités, imprécises, prédictives,…)

#### 1.3.5. Valeur

La démarche Big Data n'a de sens que pour atteindre des objectifs stratégiques de création de valeur pour les clients et pour l'entreprise; dans tous les domaines d'activité: commerce, industrie, services…En effet, le succès d'un projet Big Data n'a d'intérêt aux utilisateurs que s'il apporte de la valeur ajoutée et de nouvelles connaissances.

### 1.4. Les solutions Big Data

La plupart des outils et des frameworks de Big Data sont construits en gardant à l'esprit les caractéristiques suivantes:

**La distribution des données:** Le grand ensemble de données est divisé en morceaux ou en petits blocs et réparti sur un nombre N de nœuds ou de machines. Ainsi les données sont réparties sur plusieurs nœuds et sont prêtes au traitement parallèle. Dans le monde du Big Data, ce type de

distribution des données est réalisé à l'aide d'un Système de Fichiers Distribués-DFS (Distributed File System).

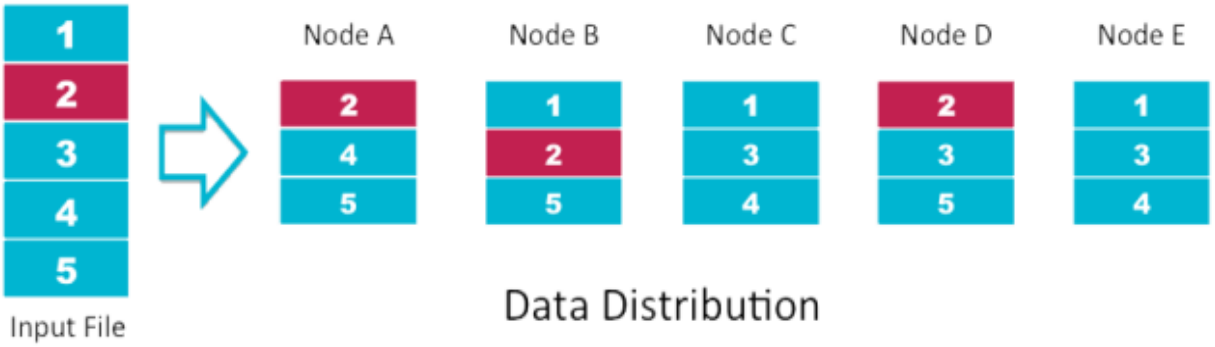



Fig.1.2. Distribution des données

**Le traitement en parallèle:** Les données distribuées obtiennent la puissance de N nombre de serveurs et de machines où les données résident. Ces serveurs travaillent en parallèle pour le traitement et l'analyse. Après le traitement, les données sont fusionnées pour le résultat final recherché.

**La tolérance aux pannes:** En général, nous gardons la réplique d'un seul bloc (ou chunk) de données plus qu'une fois. Par conséquent, même si l'un des serveurs ou des machines est complètement en panne, nous pouvons obtenir nos données à partir d'une autre machine ou d'un autre «data center».

**L'utilisation de matériel standard:** La plupart des outils et des frameworks Big Data ont besoin de matériel standard pour leur travail. Donc nous n'avons pas besoin de matériel spécialisé avec un conteneur spécial des données «RAID»[1]. Cela réduit le coût de l'infrastructure totale.

**Flexibilité, évolutivité et scalabilité:** Il est assez facile d'ajouter de plus en plus de nœuds dans le cluster quand la demande pour l'espace augmente. De plus, la façon dont les architectures de ces frameworks sont faites, convient très bien au scénario de Big Data.

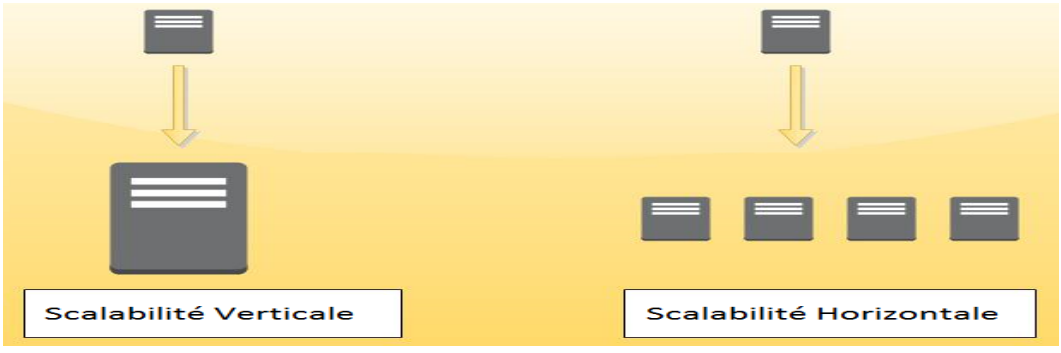



Fig.1.3. Scalabilité verticale vs horizontale

## 1.5. Perspectives et domaines d'application

Les perspectives d'utilisation de ces données sont énormes, notamment pour l'analyse d'opinions politiques, de tendance industrielles, la génomique, la lutte contre la criminalité et la fraude, les méthodes de marketing publicitaire et de vente etc…

### 1.6. Les acteurs du Big Data

Les grands acteurs du web tel que **Google**, **Yahoo**, **Facebook**, **Twitter**, **LinkedIn**…ont été les premiers à être confrontés à des volumétries de données extrêmement importantes et ont été à l'origine des premières innovations en la matière portées principalement sur deux types de technologies:

1. Les plateformes de développement et de traitement des données (GFS, Hadoop, Spark,…)
2. Les bases de données (NoSQL)

#### 1.6.1. Le cas Google

Pour stocker son index grandissant, Google a mis en place un nouveau système propriétaire: GFS (Google File Système) en 2003. C'est un algorithme inventé par Google, afin de distribuer des traitements sur un ensemble de machines avec le système GFS. Google possède aujourd'hui plus de 10000000 de serveurs interconnectés dans le monde.

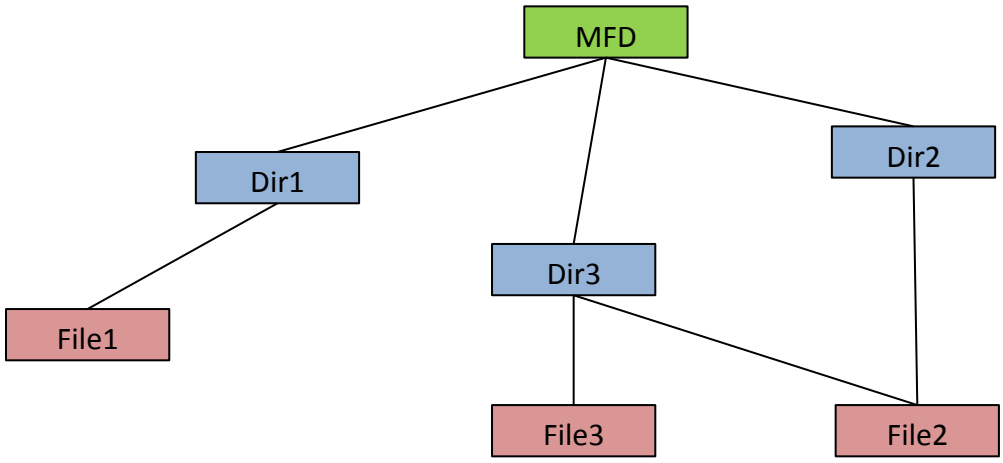



Fig .1.4. Une arborescence GFS

# Chapitre 2 : Hadoop

## 2.1. Généralités

En 2004, Google a développé un paradigme de programmation appelé MapReduce. Yahoo! a mis en place Hadoop comme implémentation de MapReduce en 2005 et a fini par le lancer en tant que projet open source en 2007.

**Hadoop** est une plateforme (framework) open source conçue pour réaliser d'une façon distribuée des traitements sur des volumes de données massives, de l'ordre de plusieurs pétaoctets. Ainsi, il est destiné à faciliter la création d'applications distribuées et échelonnables (scalables). Il s'inscrit donc typiquement sur le terrain du Big Data.

À l'instar des autres systèmes opérationnels, Hadoop possède les structures de base nécessaires à effectuer des calculs: un système de fichiers, un langage de programmation, une méthode pour distribuer les programmes ainsi générés à un cluster, un mode pour obtenir les résultats et arriver à une consolidation unique de ces derniers, ce qui est le but.

Avec Hadoop, le Big Data est distribué en segments étalés sur une série de nœuds s'exécutant sur des périphériques de base. Au sein de cette structure, les données sont dupliquées à différents endroits afin de récupérer l'intégralité des informations en cas de panne. Il s'occupe de toutes les problématiques liées au calcul distribué, comme l'accès et le partage des données, la tolérance aux pannes, ou encore la répartition des tâches aux machines membres du cluster: le programmeur a simplement à s'occuper du développement logiciel pour l'exécution de la tâche.

Hadoop assure les critères des solutions Big Data à savoir:

**Performance:** support du traitement d'énormes data sets (millions de fichiers, Go à To de données totales) en exploitant le parallélisme et la mémoire au sein des clusters de calcul.

**Economie:** contrôle des coûts en utilisant des matériels de calcul de type standard.

**Evolutivité (scalabilité):** un plus grand cluster devrait donner une meilleure performance.

**Tolérance aux pannes:** la défaillance d'un nœud ne provoque pas l'échec de calcul.

**Parallélisme de données:** le même calcul effectué sur toutes les données.

## 2.2. Composants fondamentaux de Hadoop

Hadoop est constitué de deux grandes parties :

Hadoop Distibuted File System – **HDFS** : destiné pour le stockage distribué des données

Distributed Programing Framework - **MapReduce** : destiné pour le traitement distribué des données.

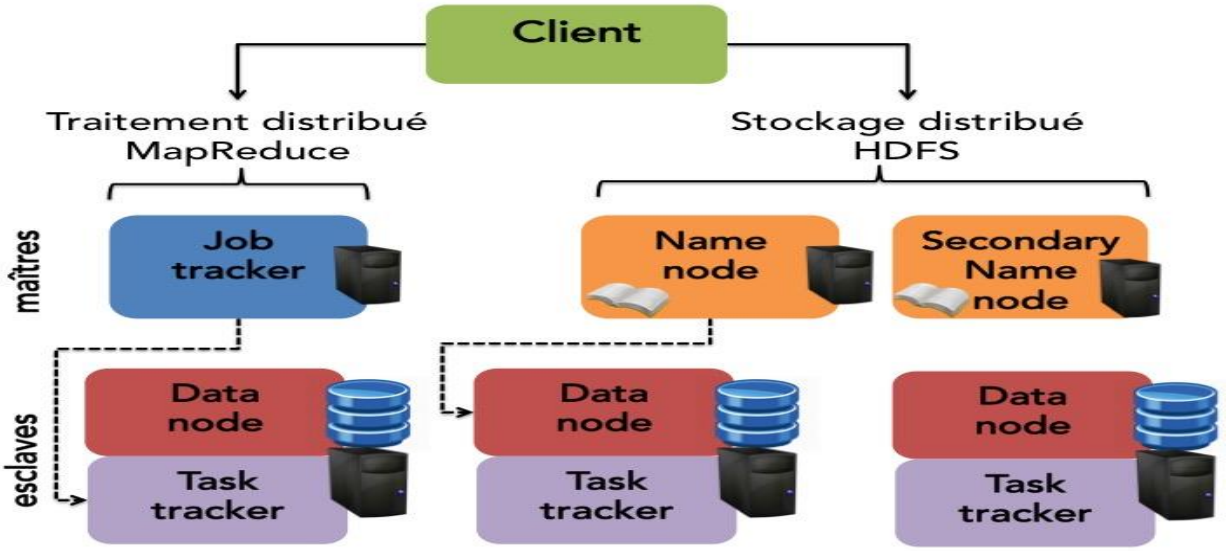



Fig. 2.1. Composants de Hadoop

### 2.3. Le système de fichier (HDFS)

HDFS (pour Hadoop Distributed File System) est un système de fichiers distribué associé à Hadoop. C'est là qu'on stocke les données d'entrée, de sortie, etc.

HDFS permet l'abstraction de l'architecture physique de stockage, afin de manipuler un système de fichiers distribué comme s'il s'agissait d'un disque dur unique. Il reprend de nombreux concepts proposés par des systèmes de fichiers classiques comme ext2 pour Linux ou FAT pour Windows. Nous retrouvons donc la notion de blocs (la plus petite unité que l'unité de stockage peut gérer), les métadonnées qui permettent de retrouver les blocs à partir d'un nom de fichier, les droits ou encore l'arborescence des répertoires. Toutefois, HDFS se démarque d'un système de fichiers classique pour les principales raisons suivantes :

- HDFS n'est pas solidaire du noyau du système d'exploitation. Il assure une portabilité et peut être déployé sur différents systèmes d'exploitation. Un des inconvénients est de devoir solliciter une application externe pour monter une unité de disque HDFS.
- HDFS est un système distribué. Sur un système classique, la taille du disque est généralement considérée comme la limite globale d'utilisation. Dans un système distribué comme HDFS, chaque nœud d'un cluster correspond à un sous-ensemble du volume global de données du cluster. Pour augmenter ce volume global, il suffira d'ajouter de nouveaux nœuds.
- HDFS utilise des tailles de blocs largement supérieures à ceux des systèmes classiques. Par défaut, la taille est fixée à 64 Mo. Il est toutefois possible de monter à 128 Mo, 256 Mo, 512 Mo voire 1 Go.
- HDFS fournit un système de réplication des blocs dont le nombre de réplications est configurable. Pendant la phase d'écriture, chaque bloc correspondant au fichier est répliqué sur plusieurs nœuds. Pour la phase de lecture, si un bloc est indisponible sur un nœud, des copies de ce bloc seront disponibles sur d'autres nœuds.

### 2.4. Architecture HDFS

Une architecture de machines HDFS inclut trois types de composants majeurs :

**NameNode (noeud de nom) :** Un Namenode est un service central (généralement appelé aussi maître) qui s'occupe de gérer l'état du système de fichiers. Il maintient l'arborescence du système de fichiers et les métadonnées de l'ensemble des fichiers et répertoires d'un système Hadoop. Le Namenode a une connaissance des Datanodes dans lesquels les blocs sont stockés.

**Secondary Namenode :** Le Namenode dans l'architecture Hadoop est un point unique de défaillance (Single Point of Failure en anglais). Si ce service est arrêté, il n'y a pas un moyen de pouvoir extraire les blocs d'un fichier donné. Pour répondre à cette problématique, un Namenode secondaire appelé Secondary Namenode a été mis en place dans l'architecture Hadoop version2. Son fonctionnement est relativement simple puisque le Namenode secondaire vérifie périodiquement l'état du Namenode principal et copie les métadonnées. Si le Namenode principal est indisponible, le Namenode secondaire prend sa place.

**Datanode :** Un Datanode contient les blocs de données. En effet, il stocke les blocs de données lui-mêmes. Il y a un DataNode pour chaque machine au sein du cluster. Les Datanodes sont sous les ordres du Namenode et sont surnommés les Workers. Ils sont donc sollicités par les Namenodes lors des opérations de lecture et d'écriture.

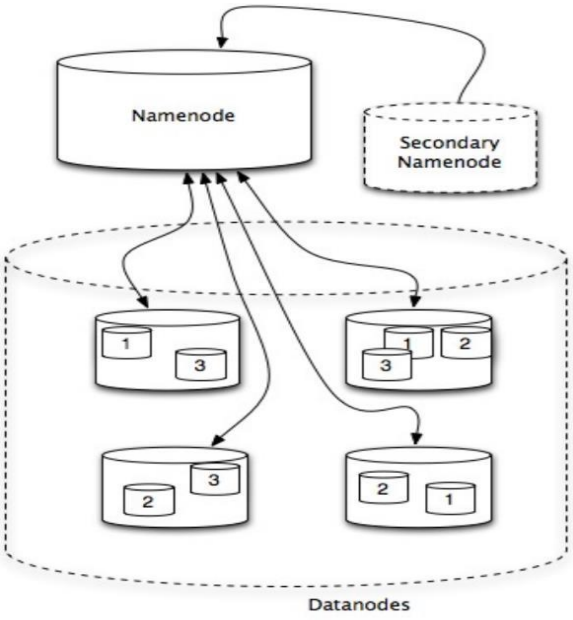



Fig.2.2. Trois types de machines de HDFS

### 2.5. Les commandes de HDFS

La commande permettant de stocker ou extraire des fichiers de HDFS est l'utilitaire console hadoop, avec l'option fs. Il réplique globalement les commandes systèmes standard Linux. On trouve par exemple :

- hadoop fs -put livre.txt /data_input/livre.txt

Permet de stocker le fichier livre.txt sur HDFS dans le répertoire /data_input.

- hadoop fs -get /data_input/livre.txt livre.txt

Permet d'obtenir le fichier /data_input/livre.txt de HDFS et le stocker dans le fichier local livre.txt.

- hadoop fs -mkdir /data_input

Pour créer le répertoire /data_input

- hadoop fs -rm /data_input/livre.txt

Pour supprimer le fichier /data_input/livre.tx

# Chapitre 3 : MapReduce

## 3.1. Généralités

En 2004, Google a développé un paradigme appelé MapReduce, et en 2005, Yahoo! a mis en place Hadoop comme implémentation de MapReduce et a fini par le lancer en tant que projet open source en 2007.

À l'instar des autres systèmes opérationnels, Hadoop possède les structures de base nécessaires à effectuer des calculs: un système de fichiers, un langage de programmation, une méthode pour distribuer les programmes ainsi générés à un cluster, un mode pour obtenir les résultats et arriver à une consolidation unique de ces derniers, ce qui est le but.

## 3.2. Exemple introductif

Supposons qu'une entreprise possède plusieurs magasins qu'elle gère à travers le monde. Pour cela, elle a un très grand livre de comptes qui contient TOUTES les ventes.

**Objectif** : Calculer le total des ventes par magasin pour l'année en cours

Supposons que les lignes du livre aient la forme suivante: Jour Ville produit Prix

| | | | |
|---|---|---|---|
| 2018-01-01 | London | Books | 51.3 |
| 2018-01-01 | Paris | Music | 22.5 |
| 2018-01-02 | Rome | Clothes | 100.1 |
| 2018-01-02 | London | Clothes | 46 |

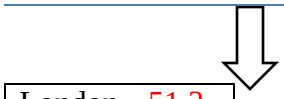
Possibilité:

- Pour chaque entrée, saisir la ville et le prix de vente
- Si on trouve une entrée avec une ville déjà saisie, on les regroupe en faisant la somme des ventes

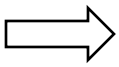
Dans un environnement traditionnel, on fera des *Hash table* sous la forme <clé-valeur>

- Dans ce cas, la clé sera l'adresse du magasin et la valeur le total de ventes

| | | | |
|---|---|---|---|
| 2018-01-01 | London | Books | 51.3 |
| 2018-01-01 | Paris | Music | 22.5 |
| 2018-01-02 | Rome | Clothes | 100.1 |
| 2018-01-02 | London | Clothes | 46 |



| | |
|---|---|
| London | 51.3 |
| Paris | 22.5 |
| Rome | 100.1 |



| | |
|---|---|
| London | 97.3 |
| Paris | 22.5 |
| Rome | 100.1 |

Bien qu'il est fort possible de rencontrer des problèmes de taille mémoire, et que le traitement séquentiel s'avère long en plus du problème classique de l'ajout de nouveaux magasins, le résultat peut s'avérer correct.

| | | | |
|---|---|---|---|
| 2018-01-01 | London | Books | 51.3 |
| 2018-01-01 | Paris | Music | 22.5 |
| 2018-01-02 | Rome | Clothes | 100.1 |
| 2018-01-02 | London | Clothes | 46 |

Clé  Valeur

| | |
|---|---|
| London | 97.3 |
| Paris | 22.5 |
| Rome | 100.1 |

Le Map-Reduce est un moyen plus efficace et plus rapide de traiter ces données, le principe consiste en :

- Au lieu d'avoir une seule personne qui parcourt le livre, on recruterait plusieurs.
- Appeler un premier groupe les Mappers et un autre les Reducers
- Diviser le livre en plusieurs parties, et en donner une à chaque Mapper

Les Mappers peuvent travailler en même temps, chacun sur une partie des données :

**Mappers**

- Pour chaque entrée, saisir la ville et le total de ventes dans une fiche
- Rassembler les fiches d'un même magasin dans une pile

**Reducers**

- Chaque reducer sera responsable d'un ensemble de magasins
- Collectent les fiches associées des différents mappers
- Pour chaque ville, ils parcourent les piles en ordre alphabétique et font la somme des enregistrements

### 3.3 Le modèle de programmation MapReduce

Dans le modèle de programmation MapReduce, le développeur implémente 2 fonctions : la fonction **Map** et la fonction **Reduce**

**Fonction Map :** prend en entrée un ensemble de « Clé, Valeurs » et retourne une liste intermédiaire de «Clé1, Valeur1» :

Map(key,value) -> list(key1,value1)

**Fonction Reduce :** prend en entrée une liste intermédiaire de « Clé1, Valeur1 » et fournit en sortie une ensemble de « Clé1, Valeur2 » :

Reduce(key1,list(value1)) -> value2

L'algorithme MapReduce s'exécute en 5 phases :

1. La phase Initialisation
2. La phase Map
3. La phase regroupement (Shuffle)
4. La phase de Tri
5. La phase Reduce

### 3.4. Fonctionnement de MapReduce

Lorsque l'application MapReduce est lancée, elle crée un composant « ***Master »*** responsable de la distribution des données et de la coordination de l'exécution de différentes unités de travail ou « ***Workers*** ». Le ***Master*** attribue aux ***Workers*** les tâches ***Map*** et ***Reduce***

Un ***Worker*** possède **3 états**:

- ***idle*** : il est *disponible* pour un nouveau traitement
- ***in-progress*** : un *traitement est en cours* d'exécution
- ***completed*** : il a *fini un traitement*, il en *informe* alors le *Master* de la taille, de la localisation de ses fichiers intermédiaires

Le ***Master*** gère la synchronisation, la réorganisation, le tri et le regroupement des données :

lorsqu'un ***Worker*** de type ***Map*** a fini son traitement, le ***Master*** regroupe, trie et renvoie le résultat à un ***Worker*** de type ***Reduce***

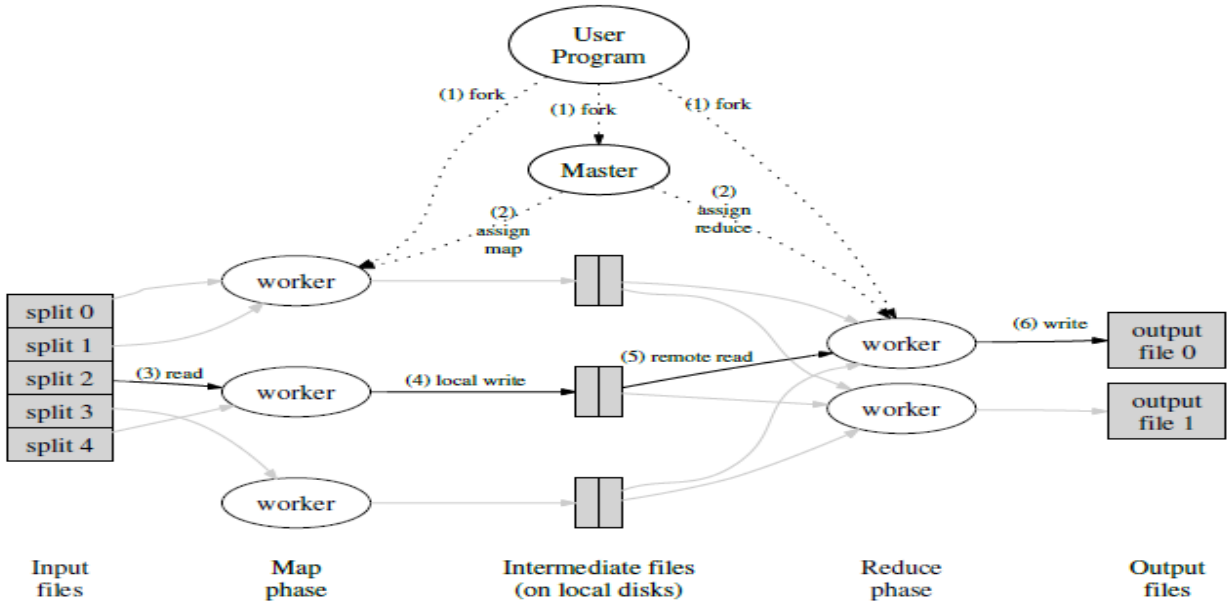



Fig.3.1. Le flux MapReduce

### 3.5. Exemples illustratifs

#### 3.5.1. Le comptage de mots

Imaginons qu'on nous donne un texte écrit en langue Française. On souhaite déterminer pour un travail de recherche quels sont les mots les plus utilisés au sein de ce texte. Les données d'entrée sont constituées du contenu du texte.

Première étape: déterminer une manière de découper (split) les données d'entrée pour que chacune des machines puisse travailler sur une partie du texte. On décide de découper les données d'entrée ligne par ligne. Chacune des lignes du texte sera un fragment de nos données d'entrée.

> Celui qui croyait au ciel
>
> Celui qui n'y croyait pas
>
> Fou qui fait le délicat
>
> Fou qui songe à ses querelles

Il est de coutume de simplifier le travail en enlevant la ponctuation et les caractères accentués ainsi que les majuscules. On obtient les quatre fragments suivants :

> celui qui croyait au ciel

celui qui ny croyait pas

fou qui fait le délicat

fou qui songe a ses querelles

D'abord, on détermine la clé à utiliser pour l'opération MAP, et écrire son code. Comme on s'intéresse aux occurrences des mots dans le texte, la clef qu'on choisit est le mot. L'opération MAP consiste à parcourir le fragment considéré et, pour chacun des mots, émettre le couple clef/valeur: (MOT ; 1). Cela signifie que nous avons rencontré une fois le mot. Le pseudo code de la fonction Map peut être :

```
Pour chaque mot de la ligne
        emettre (mot ; 1)
```

Pour nos fragments, les couples (clé,valeur) ainsi générés sont :

| | |
|---|---|
| celui qui croyait au ciel | (celui ;1) (qui ;1) (croyait ;1) (au ;1) (ciel ;1) |
| celui qui ny croyait pas | (celui ;1) (qui ;1) (ny ;1) (croyait ;1) (pas ;1) |
| fou qui fait le délicat | (fou ;1) (qui ;1) (fait ;1) (le ;1) (délicat ;1) |
| fou qui songe a ses querelles | (fou ;1) (qui ;1) (songe ;1) (a ;1) (ses ;1) (querelles ;1) |

Une fois l'opération « Map » achevée, un mécanisme intermédiaire permet de regrouper les couples par clé commune ; on obtient :

(celui ;1) (celui ;1)

(qui ;1) (qui ;1) (qui ;1) (qui ;1)

(croyait ;1) (croyait ;1)

(fou ;1) (fou ;1)

(au ;1)

(ciel ;1)

etc.

La fonction « Reduce » considère chacun des groupements de clés où elle va opérer à une somme. Le pseudo code peut être :

```
Total=0
Pour chaque Couple dans Groupe
        Total=Total+1
emettre(mot ; total)
```

A la fin de l'opération, on obtient pour chaque mot son nombre d'apparition dans le texte.

(celui ;2)

(qui ;4)

(croyait ;2)

(au ;1)

(ciel ;1)

etc.

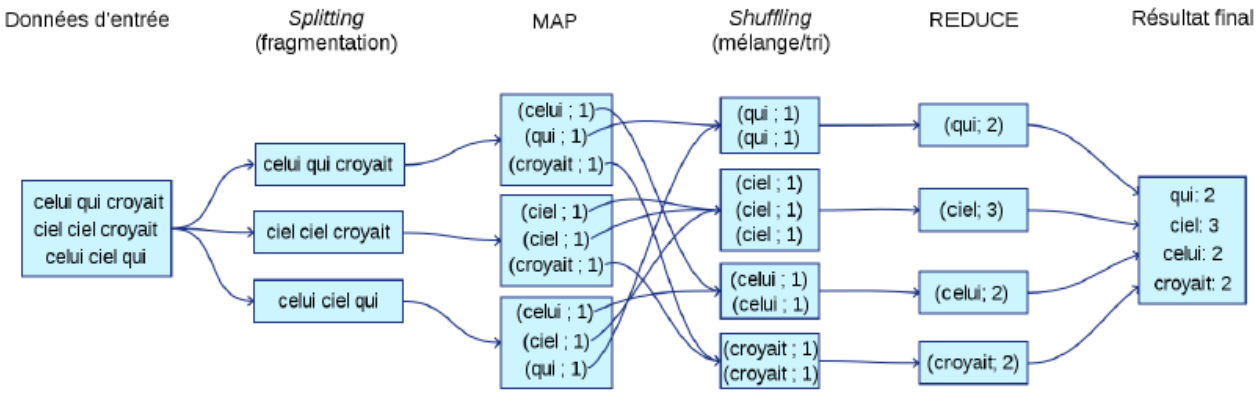



Fig. 3.2. Flux de données MapReduce avec 1 seule tâche Reduce

### 3.5.2. Longueur moyenne des mots

On s'intéresse maintenant à calculer la longueur moyenne des mots dans un texte. On procède alors au découpage comme précédemment au texte d'entrée en lignes. Pour chaque ligne, on calcule le nombre de mots et la longueur de la ligne (ie le nombre de caractères). La longueur moyenne se fait selon la formule :

$$\mu = \frac{1}{n}\sum_{1}^{n} x_i$$

L'entrée de Map est un ensemble de mots {w} d'un fragment du texte, la fonction « Map » calcule le nombre de mots dans le fragment et la longueur totale des mots. La sortie de la fonction « Map » contient deux couples : <"count", #mots> et <"length", longueur totale>.

L'entrée de Reduce est un ensemble {<clé, {valeur}>}, où clé = "count" ou "length" et valeur est un entier. La fonction « Reduce » calcule le nombre total de mots : N = somme de toutes les valeurs "count" et la longueur totale des mots : L = somme de toutes les valeurs "length"

La sortie « Reduce » est <"count", N> et <"length", L>, Le résultat est obtenu en faisant la division μ=L/N.

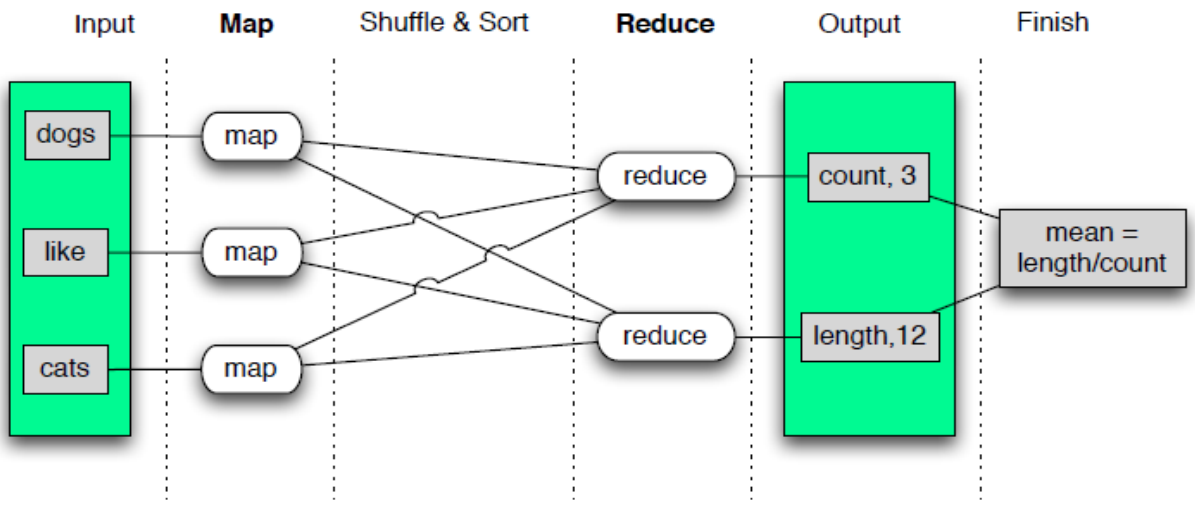



Fig..3.3. Flux de données MapReduce avec 2 tâches Reduce