Synthèse Big Data

Table des matières

Introduction au Big Data

L'ère de l'information et du Big Data

De nos jours, on a de plus en plus de données, Il faut dès lors trouver un moyen de les traiter. De nouvelles méthodes apparaissent afin de nous y aider (Machine Learning, …).

En soit l'humain d'aujourd'hui est beaucoup plus social (du aux réseaux sociaux) et plus mobile (Toujours connecté à l'aide de son smartphone notamment). De plus nos données sont stockées un peu n'importe où dû notamment à l'utilisation des clouds.

L'émergence des réseaux sociaux

On remarque qu'au fil du temps sont apparus divers réseaux sociaux et également l'ampleur que ceux-ci ont acquis dans notre quotidien. Leur évolution est sans limite et a pris une telle ampleur que de nombreuses marques (93 %), sociétés, … l'utilisent au niveau de leur marketing.

Social Business

Le social business au contraire de son nom n'a aucune vocation sociale. En soit le social business permet à une entreprise classique de continuer à fournir son service mais d'une nouvelle manière. On va tenter d'inclure les parties prenantes au sein du processus (clients, employés, fournisseurs, …). On tentera de créer une communauté autour du produit comme c'est par exemple le cas avec le crowdfunding.

On remarquera d'ailleurs qu'il existe deux types d'échange :

- Mono directionnel : l'entreprise produit, le client quant à lui achète.
- Multi directionnel : il s'agit de l'échange mono directionnel auquel on ajoute la présence du client, on va notamment lui demander son avis, des suggestions (On va faire en sorte qu'il ait ce qu'il aimerait avoir).

Le client joue dans ce type de business un rôle de client actif (l'entreprise va s'impliquer dans l'entreprise).

« Les PDG veulent que les clients participent davantage… jusqu'à contribuer au développement de leur stratégie d'entreprise ».

Consom'acteur

Dans le temps, un consommateur était surtout passif. De nos jours les gens souhaitent être de plus en plus actif, ils achètent notamment en fonction de la valeur du produit et portant particulièrement importance à la réputation de la marque. En soit-ils agissent de plus en plus en tant que citoyen responsable qu'en tant que mouton.

Le consommateur peut de nos jours favoriser nos activités dans le cas où celles-ci soient en accord avec ces valeurs. Mais il peut aussi dans le cas où nos valeurs ne sont pas en adéquation avec les siennes, boycotter nos activités.

On remarquera plusieurs catégories de consommateurs :

- Influencé : ce consommateur ci va sur base de ce qu'il a pu lire ailleurs (sur internet, …) et sur base de l'avis d'autres gens aiguillé son choix d'achat.

- Influenceur : va donner son avis sur un produit et partagé celui ici avec d'autres personnes (Par exemple : dénoncer un produit non fonctionnel ou encore donner un avis positif). On remarquera deux types d'avis : les avis positifs et les avis négatifs. Mais il faudra dès lors faire attention aux faux avis potentiels ainsi qu'aux avis truqués (il faut également avoir conscience qu'il y a une plus grande tendance à avoir des avis négatifs).

Klout.com est une entreprise qui analyse les réseaux sociaux (ce que les gens publient, leurs avis) et donnent un score à chaque personne, il s'agira du score d'influence de la personne (La puissance de ces avis, la fréquence de ceux-ci). A la suite de la récolte de ces données, elle va ensuite les envoyer aux entreprises.

Analyse des réseaux sociaux

Il s'agit en soit d'une des disciplines du Big Data même si elle peut prendre énormément de formes différentes. Le but va être de tirer le maximum d'info (ou du moins les informations pertinentes), comme par exemple : les ressentis, les thèmes clés, les tendances, les sites et contributeurs influents, …

Cela va notamment permettre de définir de nouvelles stratégies de marques ou commerciales.

Le Mobile

L'importance du mobile dans nos quotidiens est sans contestes. Les mobiles sont même plus répandus que les téléphones fixes (dû à leur coût moindre à l'installation, de sa mobilité, de son coté pratique ou encore de sont coût moindre en infrastructure).

On remarquera également son utilisation dans des opérations commerciales (M-Commerce, Paiements mobiles, …). Mais cela est très dur à quantifié. Le mobile impacte même les transactions en magasin. Cela ouvre donc forcément de nombreuses opportunités même si celle-ci ne sont pas forcément bonnes.

Internet of things (IOT)

De nos jours l'internet of things commence à devenir de plus en plus important et est en expansion croissante. De nombreux objets de notre quotidien commencent à être connectés (Montres connectées, voiture connectée, frigo connecté, …).

Combiner

Il s'agit de l'utilisation de données venant de différentes sources afin de fournir des solutions pouvant intéresser. La combinaison d'informations est l'étape logique suivant la collecte de ces dernières.

On remarquera notamment les smart cities ou en bon français les villes connectées, dans lesquelles on a notamment installé des capteurs qui sont disponibles à tout le monde afin que ceux qui en voient une utilité puissent s'en servir.

Cloud

Le cloud est de nos jours présents dans de nombreux aspects de notre vie, les systèmes de sécurité, nos balances, nos montres, nos messageries, voici de nombreux systèmes utilisant le cloud.

On distinguera d'ailleurs deux types de cloud :

- Les clouds publics : qui sont comme leur nom l'indique accessible à tous.
- Les clouds privés : les machines nous sont dès lors dédiées et nous appartiennent.

Changement de paradigme

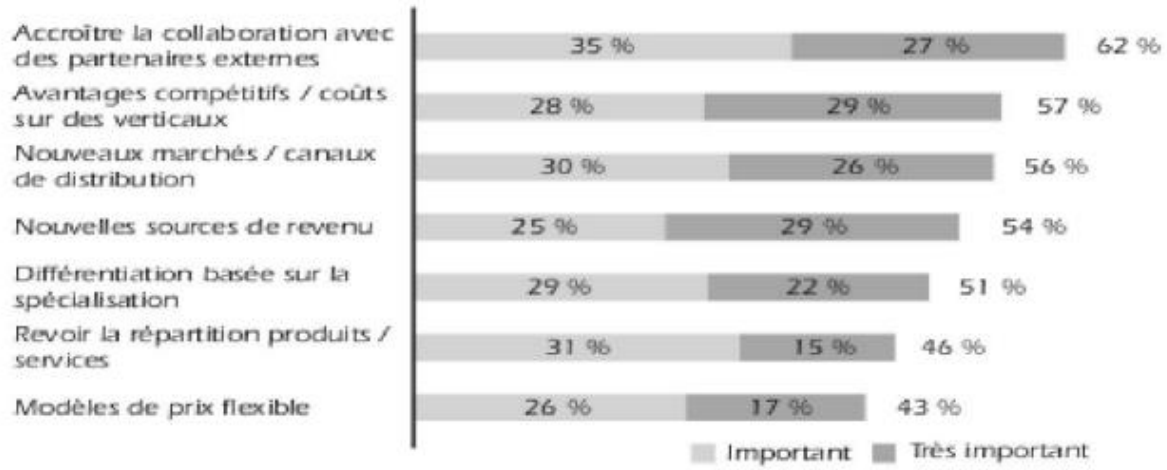
De nos jours la fluidité de l'information est une chose très importante. L'information doit pouvoir circuler facilement entre les différentes entreprises, leurs partenaires, leurs employés, leurs administrations, leurs clients, …

On va dès lors devoir utiliser le code pour fluidifier ce flux d'informations. Un point important est la sécurité et le sérieux du fournisseur de ce flux.

Contextualisation

En soit le cloud permet la collecte, le stockage et l'analyse des données. La combinaison du cloud et du Big data offre surtout des services contextuels et qui donc vont varier en fonction du contexte.

L'open data est le fait de mettre les données accessibles à tout le monde, de les rendre publiques.



Source : "The Power of cloud – IBV PoV and Executive Report", 2012

Dès le moment on nous sommes les personnes qui générons ces données, pourquoi certaines sociétés pourraient / devraient garder ces données privées. Le souci dans tout cela est que malgré l'anonymisation des données, on pourrait dresser un profil très précis d'une personne en particulier ou même d'un échantillon de personnes en regroupant / combinant ces données.

Les 3V

Il s'agit du pilier du Big Data. En soit le Big Data n'est pas l'analyse de gros volume de données traditionnels. Le Big Data est la combinaison de ces données (volumineuses ou non) afin de décider d'une action.

Dû à cette augmentation du nombre de données, les bases de données traditionnelles doivent être revues car elle n'arrive pas à traiter cet amas de données. De nouveaux outils sont donc à notre disposition (Hadoop, …).

La business intelligence (BI) quant à elle effectue un travail de recherche de causalité. On va de trouver les causes de la situation actuelle.

Volume

Anciennement on interrogeait un échantillon de la population afin d'effectuer des statistiques (on procédait à un échantillonnage). Mais certains problèmes s'en dégageaient : la façon de poser la question pouvait influencer les réponses, on pouvait réfléchir avant de donner notre réponse et de plus il s'agissait des données dites mais non pensées.

Le Big Data quant à lui travaille sur l'entièreté de la population. On peut donc espérer que cela est plus fiable. De plus un mauvais choix de traitement peut être rattrapé en recommençant le traitement. Le plus gros avantage est qu'il travaille directement sur l'avis des gens (lorsqu'ils sont à chaud).

On va tenter un travail de prédiction plutôt qu'un travail de causalité comme le fait la BI.

« Plus on a de données sur un phénomène et son environnement, mieux on l'appréhende. »

Vélocité

Afin que les résultats reflètent avec précision la situation actuelle il faut que les données utilisées soient les plus fraiches possibles et qu'elles soient donc toujours d'actualité.

Par exemple si nous prenons le real time biding, la vente de certains emplacements présent sur notre site : celui qui paie le plus cher pour cet emplacement de publicité aura la possibilité d'y afficher sa publicité. Mais pour cela il faut être rapide.

Les résultats doivent donc être le plus instantané possible.

Variété

Afin de permettre divers types de résultats, tout deviendra donnée. Ainsi qu'il s'agisse de texte, de son, d'images on va tenter de récupérer cela afin d'y apposer un traitement.

On remarquera dès lors différents types de données :

- Structurées : il ne s'agit pas forcément de structure comme nous la voyons. Par exemple : un fichier xml possédant un fichier contraignant, …
- Semi-structurée : il s'agit du type de données le plus répandu. Par exemple : fichier xml (sans fichier contraignant, fichier de log, …)
- Non structurée : image, film, …

Un élément qui prend de l'ampleur dû à cette différence des types de données est l'intelligence artificielle, celle-ci va nous aider à traiter ces diverses données et donc de le faire plus rapidement et avec plus d'efficacité.

NoSQL

SGBD relationnelles

Même si les bases de données relationnelles offrent de nombreux avantages : l'utilisation de tables et de tuples pour structurer l'information, les liens par clés permettant de faire facilement des sélections et des jointures (à l'aide de SQL), elles sont basées sur un système à scalabilité verticale et ne permettent donc malheureusement pas de faire de la scalabilité horizontale.

Pionniers du NoSQL

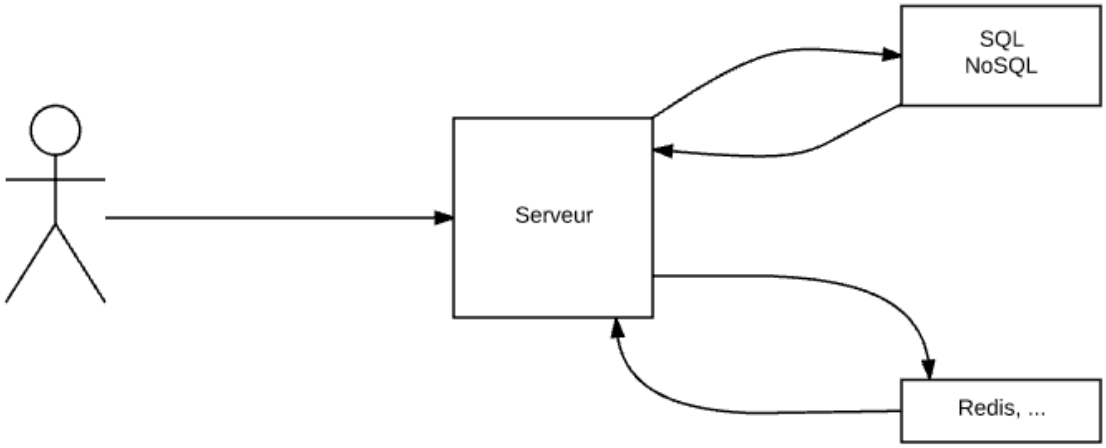
Dû à ce besoin de scalabilité horizontale et l'impossibilité de mettre cela en œuvre avec une base de données relationnelles classique est « né » le NoSQL qui permettait d'apporter une autre solution. Il faudra tout de même noter que NoSQL ne veut pas dire plus de SQL non, il s'agit plus de bases de données n'utilisant pas seulement du SQL (Not Only SQL).

Base de données orienté clefs – valeurs

Il s'agit de bases de données tentant de stocker de petits formats de données (Chaines de caractères, tableaux associatifs, listes, ensembles, …). Celle-ci sera stockée en RAM afin d'être très rapide.

Ce type de bases de données ne permet ni la scalabilité horizontale, ni de stocker de grandes quantités de données. De plus en cas de panne, une partie de nos données se verra perdue. En contrepartie, ce type de bases de données a de très bonnes performances et est donc parfaitement adapté par exemple pour les sessions.

En fait les bases de données orienté clefs – valeurs (comme Redis) sont surtout utiles si combinés avec une autre base de données plus conséquentes. Ainsi celle orienté clefs – valeurs pourra contenir les informations les plus importantes demandant un traitement rapide tandis que l'autre se chargera de contenir l'entièreté des informations.



En conclusion, les bases de données orientées clefs – valeurs ont surtout permis d'apporter comme solution de meilleures performances comparer aux anciennes bases de données relationnelles un peu lente.

Un exemple de ce type de bases de données, est Redis, une base de données écrite en C possédant de très hautes performances et se basant sur l'utilisation de la RAM, de l'évènementiel et un principe de maitre – esclave (master -slave) dans lequel un maitre va rediriger les requêtes vers les esclaves (le maitre n'effectuera donc aucun traitement).

Bases de données orientées documents

On tentera de stocker au sein de notre base de données des collections, ou des documents. Mais un document pourrait très bien ne pas avoir de schéma et donc on pourrait avoir plusieurs types de documents différents dans une même collection. En soit on remarquera qu'il n'y a plus de type de

tuples mais bien de valeurs. Ce type de bases de données est « Schema Less ». Chaque objet sera généralement un JSON (ou un BJSON, du Binary JSON).

L'avantage de ce type de bases de données est que l'on peut y stocker ce que l'on veut ajouter à cela le fait qu'elles utilisent du JSON (ou BJSON), il y a beaucoup moins de conversions, et donc beaucoup moins de pertes et également de meilleures performances. On parlera notamment de Full Stack JS (Notre application sera entièrement en JavaScript du Front end à la base de données) ainsi que de Sharding (découpe de l'application en plusieurs morceaux de données).

Les désavantages sont cependant cette absence de relations entre les documents et donc pas de jointures. Dû à cela une application demandant de nombreuses relations entre les données ne sera pas adaptée avec ce type de base de données. En soit une base de données orienté documents peut s'apparenter à une dénormalisation (Et il nous faudra dès lors reporter chacune des modifications dans le cadre d'une occurrence).

Exemple de telles bases de données : MongoDb, Couchdb

Bases de données orientées colonnes

Cela pourrait ± s'apparenter à une table possédant des colonnes dynamiques. La plupart des requêtes pouvant s'appliquer sur ce type de bases de données seront des requêtes simplistes et nécessiteront un index.

Les colonnes ne seront pas forcément gardées sur le disque (on ne gardera que les données que l'on a besoin). On tentera donc d'avoir un gain de place. Ce type de bases de données est prévu (à la base) pour une scalabilité horizontale.

Bases de données orientées graphes

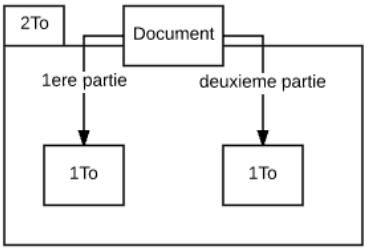
Ce type de bases de données sera surtout constituées de nœuds et d'arcs afin de former un graphe. Ce type de bases de données est surtout pratique lors de la modélisation des réseaux sociaux. L'utilisation de l'algorithme de Dijkstra au sein de ce type de bases de données est donc relativement importante.

Exemple : Neo4j, Orient DB.

Hadoop

Avant de nous attaquer regardons ensemble les différentes manières dont nous pourrions améliorer nos systèmes de stockages actuels :

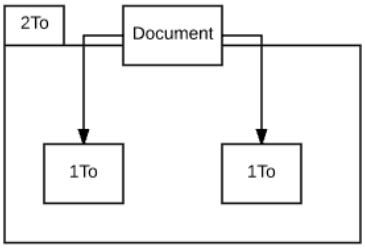
1. RAID 0



On va effectuer diviser le document en deux afin que de diviser la tâche en deux. Cela va permettre de meilleures performances en écriture / lecture mais en contrepartie si l'un des disques dur lâche, on perdra automatiquement les données de l'autre car elles deviendront illisibles.

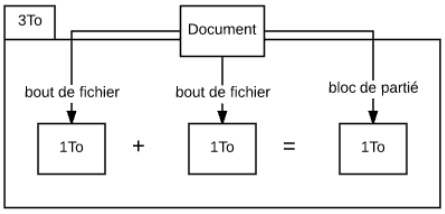
2. RAID 1

Il s'agit en soit du « RAID sécurité » on va écrire un document en double. On va donc avoir une réplication parfaite des deux disques durs. En soit il s'agit d'un système de backup automatisé.



3. RAID 5

On va sauvegarder une partie d'un document au sein d'un disque et l'autre partie au sein d'un autre mais on va également garder un bloc de parité qui permettra de savoir dans le cas où l'un des disques lâche, le contenu du disque défectueux.



En soit on remarquera deux types de RAID : les RAID orienté matériel qui sont plus rapide mais ne connaissent pas le contenu des disques et les raids logiciels qui sont plus flexibles et qui travaille avec l'OS et au niveau des partitions. Le choix de tel ou tel type de RAID dépend des cas d'utilisation.

Introduction

Dû à la production de plus en plus massive de données et à cette combinaison des 3V du Big Data les systèmes actuellement ms en place ne permettaient pas une gestion efficace des fichiers.

C'est donc pour cela que voient le jour des systèmes distribués comme le HDFS (Hadoop Distributed File System) ainsi que le mapReduce. Hadoop permet notamment d'utiliser un environnement distribué (on pourra donc séparer le travail sur différentes machines) et de traiter de grandes quantités de données

Dû à sa nature il faudra faire attention Hadoop n'a de l'intérêt que dans un système de grande taille et n'est donc pas du tout adapté pour les petits fichiers.

HDFS

Il s'agit d'un système de fichiers (un fichier étant un ensemble de blocs), reliant le nom du fichier à une liste de blocs. Ce système de fichiers possédera également des permissions ainsi que des répertoires comme un système de fichiers classique (Unix).

Les grandes différences de l'HDFS est qu'il n'est pas lié au noyau et donc est portable. En soit il s'agit d'une application Java (et donc est « virtuel ») et pourra créer un nouveau système de fichiers sur n'importe quelle machine, il s'agit donc d'une application externe de montage.

De plus celui-ci est distribué (et ne possède donc pas de limite de taille) et on pourra donc stocker les données sur plusieurs serveurs. A cela vient s'ajouter sa taille de blocs plus élevées que sur un système de fichiers « classique » celle-ci dépassera les 64Mo on n'y stockera donc que des gros fichiers. A l'aide de cela on gaspillera moins de place.

HDFS permettra également la réplication des blocs. Dû au fait que chaque bloc est distribué, ceux-ci peuvent donc se retrouver sur n'importe lequel des serveurs et pourraient donc être répliqués sur des serveurs différents. (Le facteur par défaut de réplication au sein de HDFS est 3).

HDFS – Namenode

Il s'agit du service central (en soit du maître). C'est lui qui possède la connaissance de l'état du système de fichiers (Tant les métas données que les répertoires ou les droits). Il possède également la connaissance des datanodes de notre système. Il connait donc l'entièreté du système.

Il a également un rôle de load balancing. Il assurera le rôle de chef d'orchestre et devra répartir équitablement les tâches.

Lors du démarrage d'un namenode celui-ci chargera la position des blocs. Ceux-ci ne seront d'ailleurs accessibles qu'en lecture seule. De plus ce procédé se passera uniquement en mémoire ce qui est donc très couteux (150 bytes par fichiers).

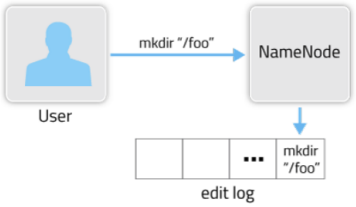
Le problème avec cela est le fait que le namenode devient dès lors un « single point of failure ». Une solution à cela est l'utilisation d'un deuxième namenode. Pour cela ce namenode devra être un réplica du premier et devra donc vérifier constamment l'état du premier. De plus il ne sera pas toujours facile pour le deuxième namenode de savoir quand prendre le relais.

HDFS – Datanode

Il s'agit en soit de ceux qui travaillent. C'est lui qui possédera les blocs de données à proprement parler. De plus ces derniers peuvent dialoguer entre eux et ne doivent donc pas forcément passer par le namenode (Cela permet au namenode de travailler le moins possible). Les datanodes se connectent au namenode dès le démarrage.

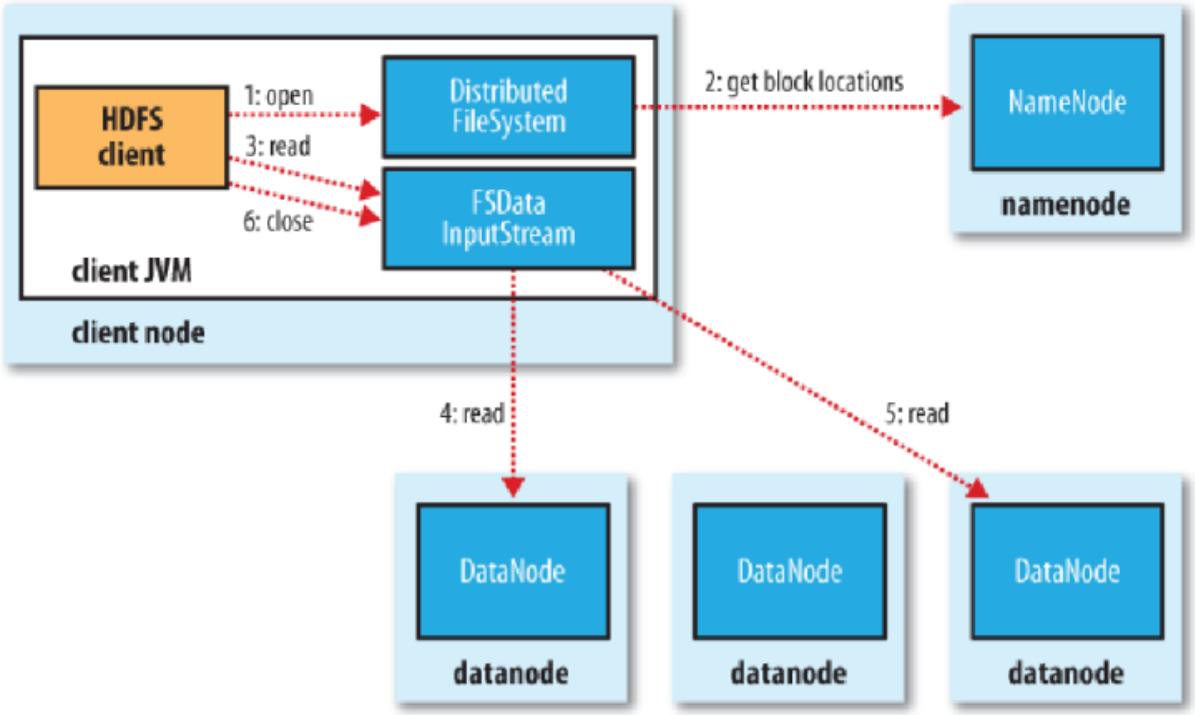
HDFS – Checkpointing

Dès que l'on fait une modification, celle-ci va se mettre dans une nouvelle photo de notre système de fichiers[1]. On va ensuite remettre les deux ensembles (Le fichier log reprenant les modifications et l'ancienne photo de notre système de fichiers) afin de recréer cette image de notre système de fichiers.



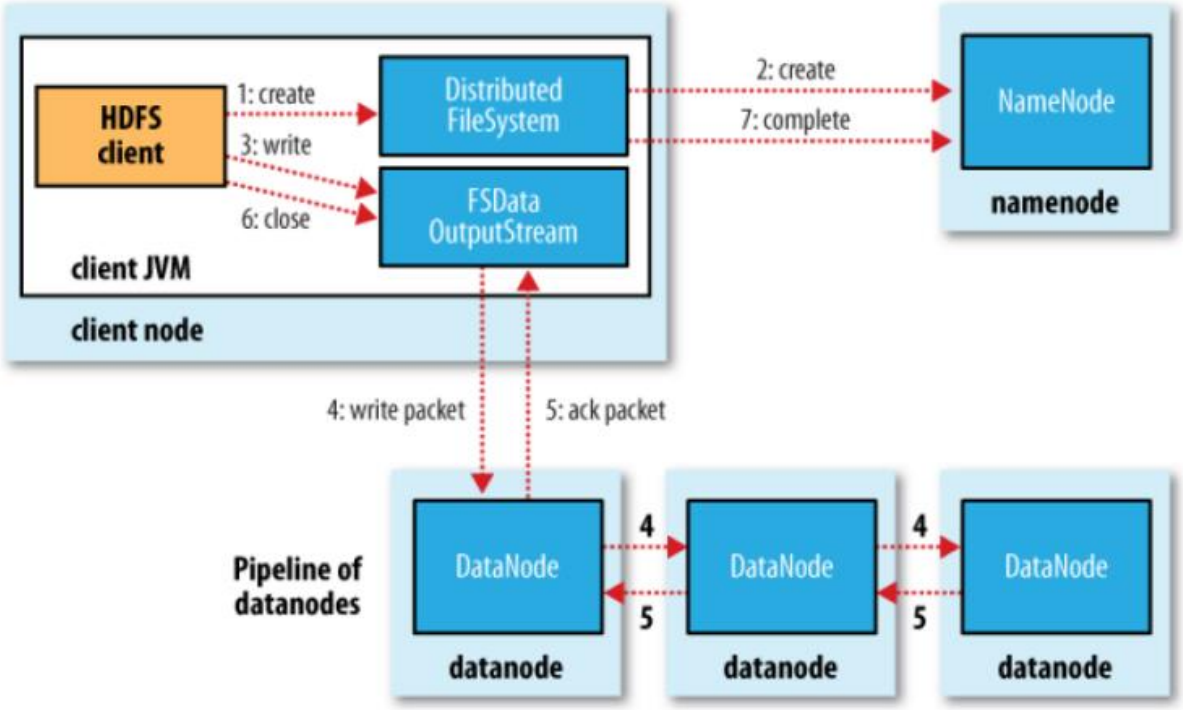
Il faudra dès lors faire attention aux photos du système de fichiers présents en mémoire (qui sont à jour) et ceux présent sur le disque (qui eux ne le sont pas).

[1] Entièreté des informations du système de fichiers




```
FileSystem fileSystem = FileSystem.get(conf);
Path path = new Path("/path/to/file.ext");
if (!fileSystem.exists(path)) {
   System.out.println("File does not exists");
   return;
}
FSDataInputStream in = fileSystem.open(path);
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
   System.out.prinln((char)numBytes));// code to manipulate
the data which is read}
   in.close();
   out.close();
   fileSystem.close();
}
```

HDFS – Écriture




```
FileSystem fileSystem = FileSystem.get(conf);
// Check if the file already exists
Path path = new Path("/path/to/file.ext");
if (fileSystem.exists(path)) {
    System.out.println("File " + dest + " already exists");
    return;
}
// Create a new file and write data to it.
FSDataOutputStream out = fileSystem.create(path);
InputStream in = new BufferedInputStream(new FileInputStream(new File(source)));

byte[] b = new byte[1024];
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
    out.write(b, 0, numBytes);
}
// Close all the file descripters
in.close();
out.close();
fileSystem.close();
```

MapReduce

Il s'agit d'une manière de programmer, d'un Framework d'implémentation. On tentera de programmer le plus proche possible des données (on aménera le calcul aux données). À l'aide de cela on va séparer la masse de travail en parallélissant celui-ci et en le distribuant.

Pour cela on va utiliser des « jobs » qui vont prendre des données en entrées, un programme mapReduce ainsi que des paramètres d'exécution (Il pourra s'agir par exemple du temps avant le démarrage, du nombre de nœuds sur lesquels faire mes calculs, …).

Au sein de Hadoop ceux-ci seront divisés en deux tâches : une pour map et une pour reduce.

MapReduce – Map

On va effectuer une map avec les données reçues en entrée, ou plutôt on va créer une liste de clefs – valeurs. En d'autres mots notre valeur sera notre liste de deonnées.

Exemple :

On prend en input deux phrases :

- Hello World, Bye World
- Hello Hadoop, GoodBye Hadoop

Et on recevra en sortie :

<Hello 1>, <World 1>, <Bye 1>, <World 1>

<Hello 1>, <Hadoop 1>, <GoodBye 1>, <Hadoop 1>

```
public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

MapReduce – Combine

On va combiner les clefs identiques afin de supprimer les doublons.

Exemple :

On aura en input une liste constituée de :

<Hello 1>, <World 1>, <Bye 1>, <World 1>

<Hello 1>, <Hadoop 1>, <GoodBye 1>, <Hadoop 1>

Et on recevra en sortie :

< Bye 1>, < Hello 1>, < World 2>

< Goodbye 1>, < Hadoop 2>, < Hello, 1>

MapReduce – Reduce

On va récupérer une liste clefs – valeurs et n'en faire qu'une liste de valeurs. Il va permettre également de réduire le nombre de listes disponibles afin de n'en former qu'une.

Exemple on prend en input :

< Bye, 1> < Hello, 1> < World, 2>

< Goodbye, 1> < Hadoop, 2> < Hello, 1>

Et on recevra en output une seule liste contenant tous les éléments distincts :

< Bye, 1> < Goodbye, 1> < Hadoop, 2> < Hello, 2> < World, 2>

```
public static class Reduce extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

MapReduce – Hadoop

On va notamment avoir trois parties importantes :

- Split

  On va séparer les différentes données au sein des différents nœuds.

- Map

  On est le plus proche possible de la donnée. Le stockage se fera par ailleurs sur le nœud.

- Reduce

  On va tout récupérer sur un seul nœud, il s'agira du nœud central, c'est lui qui s'occupera d'assembler le tout et qui récupérera les données calculées.