

**Introduction au BIG DATA**

Table des matières

[I. Introduction 2](#_Toc515616682)

[II. Histoire 3](#_Toc515616683)

[III. Le Big Data 4](#_Toc515616684)

[1. Les 3 V 4](#_Toc515616685)

[2. Les 5 buts V 4](#_Toc515616686)

[IV. Motivations 5](#_Toc515616687)

[V. Quelques défis 6](#_Toc515616688)

[1. Scalabilité : 7](#_Toc515616689)

[Scalabilité verticale : 7](#_Toc515616690)

[Scalabilité horizontale : 7](#_Toc515616691)

[3. Tolérance aux pannes 7](#_Toc515616692)

[Matériel : 7](#_Toc515616693)

[Indisponibilité de **ressources** 7](#_Toc515616694)

[Répondre à la tolérance aux pannes : 7](#_Toc515616695)

[VI. Rencontre avec Hadoop 9](#_Toc515616696)

[1. Ecosystème Hadoop 10](#_Toc515616697)

[VII. HDFS 11](#_Toc515616698)

[1. Définitions 11](#_Toc515616699)

[4. NameNode : 11](#_Toc515616700)

[5. DataNode : 11](#_Toc515616701)

[6. Architecture 12](#_Toc515616702)

[7. Réplication 12](#_Toc515616703)

[8. Gestion des pannes 13](#_Toc515616704)

[9. Zookeeper et interactions avec les NameNode & DataNode 13](#_Toc515616705)

[10. Ecriture 14](#_Toc515616706)

[11. Lecture 15](#_Toc515616707)

[VIII. MapReduce 16](#_Toc515616708)

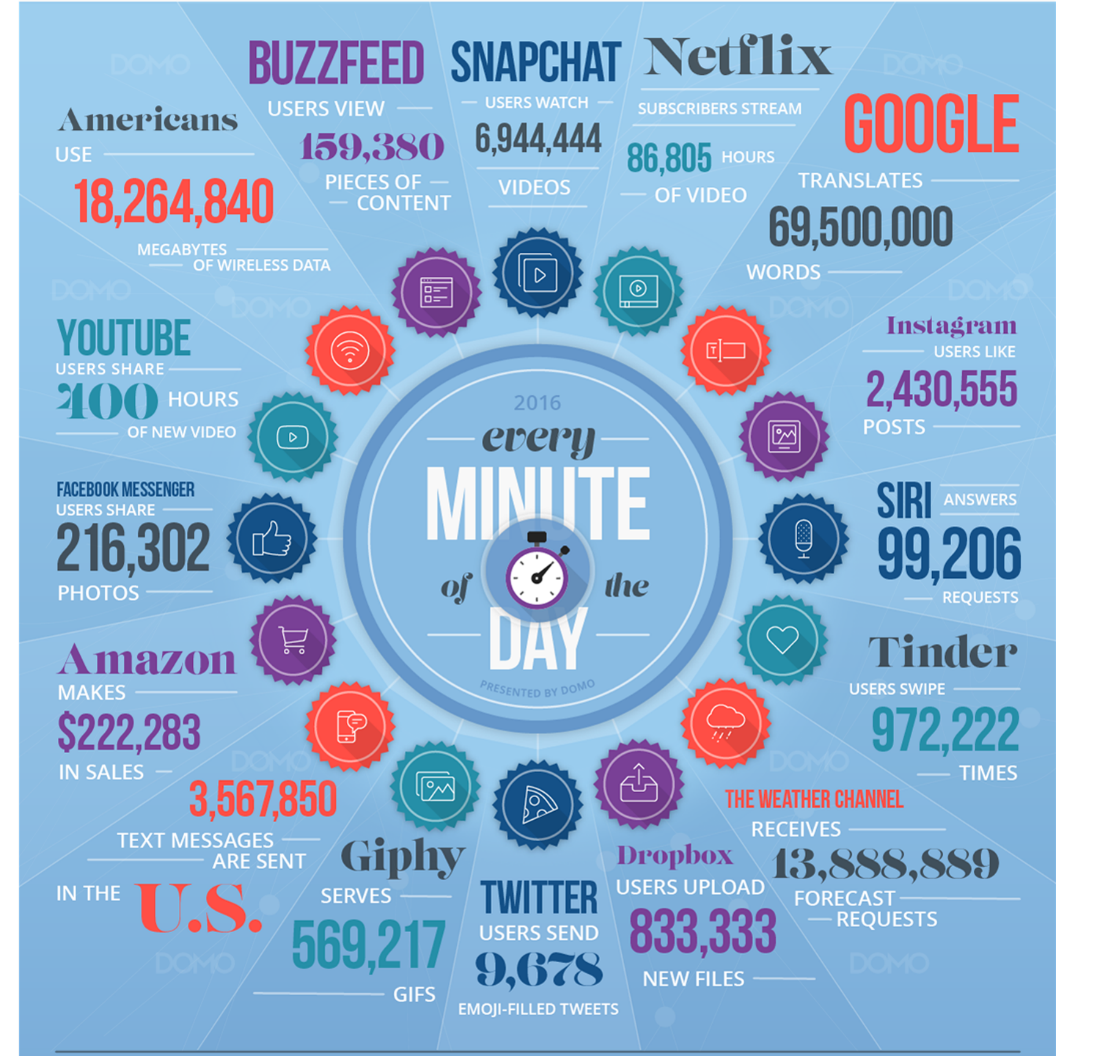
[1. Intuition 16](#_Toc515616709)

[2. En action 17](#_Toc515616710)

# Introduction

Nous vivons une époque où les quantités de données générées sont de plus en plus importantes, notamment via les réseaux sociaux, les capteurs, l'internet des choses (IoT), les applications.

Le graphique suivant montre par exemple en une minute ce qui se passe sur les différentes plateformes utilisées quotidiennement par des millions d'utilisateurs.



Pour en citer quelques exemples : en une minute il y'a 69 millions de mots traduits sur la plateforme de traduction de Google ou encore Amazon génère un chiffre d'affaire de plus de 220 K$.

# Histoire

Au début de l’apparition des moteurs de recherche et notamment Google, il devait être capable d’indexer tout le contenu du web et ainsi répondre aux recherches faites par les utilisateurs.

La plupart des moteurs de recherche été confrontés à plusieurs contraintes :

1. Stocker des volumes importants
2. Analyser les gros volumes de données ainsi stockés

Au sein de Google, les équipes ont réfléchi et eu un certain nombre d’idées :

1. Distribuer le stockage
2. Distribuer les calculs

Et pour cela ils ont développé des outils :

1. Un système des fichiers distribués
2. Et le principe du MapReduce

# Le Big Data

La plupart des acteurs qui travaillent dans la gestion de grosses quantités de données ont attribué le nom « BigData ».

Le terme BigData peut être définie de plusieurs façons, la majorité des acteurs se sont mis d’accord pour une définition avec la lettre V.

## Les 3 V

1. VOLUMETRIE

Une volumétrie importante qui ne peut être traitée par les solutions classiques (Téra octets, Péta octets). De plus cette volumétrie est croissante.

1. VARIABILITE

Des données hétérogènes, JSON, CSV, texte, ... Des formats non encore connus.

1. VELOCITE

Capacité à traiter des données en quasi temps réel. Analyse prédictive/approximative sur un échantillon de données.

## Les 5 buts V

A cela s’ajoutent deux V :

1. VERACITE

Toutes les données n’ont pas la même valeur, le même poids dans l’algorithme de traitement.

1. VALEUR

Les données sont transformées pour créer de la valeur pour les clients et l’entreprise.

La valeur est l’un des éléments les plus importants des cinq V.

# Motivations

Pour illustrer un exemple de gros volumes de données, imaginant que nous souhaitions traiter le contenu de l’encyclopédie en ligne Wikipédia.

On suppose qu’elle occupe un espace disque de 500 Téra bytes (à peu près 20 000 000 de pages web)

Si on souhaite lire l’ensemble des pages, on a un disque a une vitesse de lecture de 40 Mb/s, il nous faudra quatre mois pour tout lire.

Quatre mois c’est un délai non raisonnable pour traiter cette volumétrie.

Il faudra donc multiplier les ressources pour être capable d’aller plus vite, avoir plus de ressources :

1. Avoir plus de CPU permettra d’exécuter les traitements plus rapidement
2. Avoir plus de disque permettra de stocker de gros volumes de données et de paralléliser les traitements

Mais multiplier les ressources nous imposent un certain nombre de défis que nous allons aborder dans le chapitre suivant.

# Quelques défis

Notre système doit être capable de s’adapter à un certain nombre de défis :

1. La scalabilité :

Etre capable de s’adapter à une forte sollicitation et à une forte augmentation des volumes de données.

1. Tolérance aux pannes :

Être capable de gérer toute panne des ressources du système.

1. Facilité d’usage :

Être facile à utiliser.

1. Traitement des données en temps réels

Etre capable de traiter les données en temps réel.

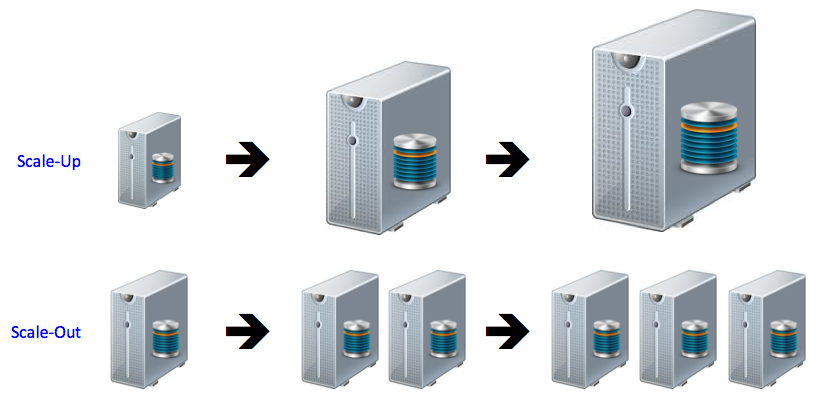
1. Traitement des données variées

Etre capable de traiter des données de formes et sources variées

1. Précision et performance

Etre performant et précis

## Scalabilité :



C’est la capacité à s’adapter à une forte sollicitation et/ou à l’augmentation des volumes de données.

Il y’a deux façons possibles d’adapter un système :

### Scalabilité verticale :

On dispose d’une machine puissante, à laquelle on ajoute de la puissance et de l’espace de stockage au fur à mesure de l’augmentation de la sollicitation.

Les machines sont fiables et robustes mais coutent cher.

### Scalabilité horizontale :

On dispose de machines petites de capacité similaire, aux quelles on ajoute les mêmes machines au fur et à mesure de l’augmentation de la sollicitation.

Les machines sont moins fiables (tombent en panne) mais capables de traiter des tâches intelligemment découpées, ensemble elles forment un **cluster**.

## Tolérance aux pannes

Dans les systèmes massivement parallélisés, les pannes sont très courantes, elles peuvent survenir à tout moment pour des raisons diverses :

### Matériel :

Le **Matériel** qui ne fonctionne pas correctement :

* Panne de disque
* Panne de mémoire
* Refroidissement inadéquat

### Indisponibilité de **ressources**

Cela peut arriver quand le système est très sollicité, toutes les ressources sont occupées à exécuter des traitements

### Répondre à la tolérance aux pannes :

On peut imaginer :

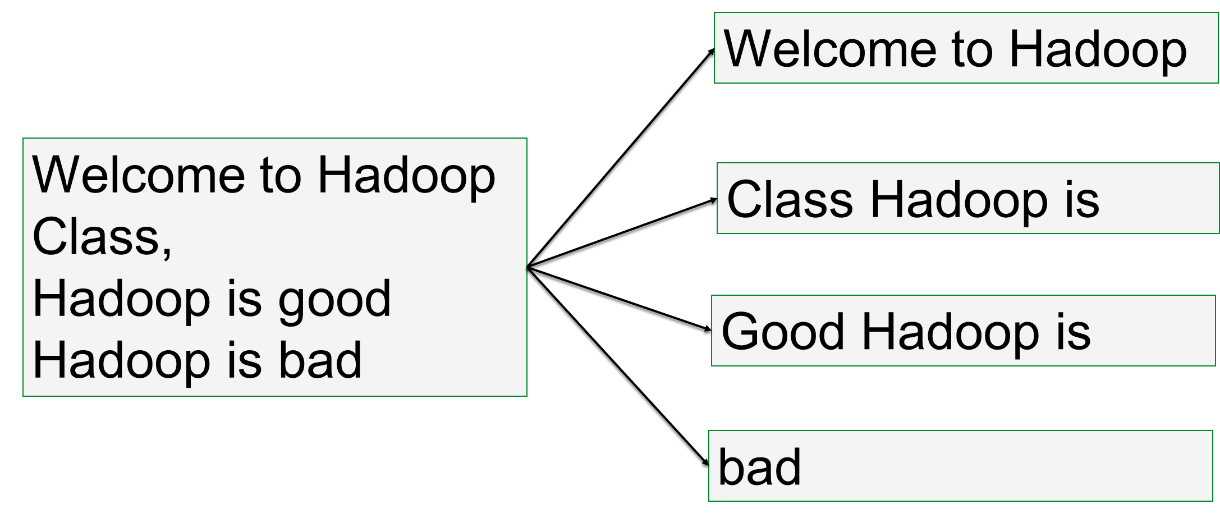
**Un protocole simple :**

* Si une panne survient on redémarre le traitement qui tournait sur la machine
* Si un job qui requiert une semaine de traitement, si on a une panne toutes les semaines, le traitement ne se terminera jamais !!!

Donc nous avons besoin de trouver un protocole meilleur que le simple.

**Un meilleur protocole :**

* Découper les données en plusieurs parties
* Lancer des petits traitements sur les petites parties
* Si une panne survient sur une machine, relancer le traitement qui a subi la panne

****

Pour relever le devis « tolérance aux pannes » :

* Un gros traitement doit être découpé en plusieurs sous traitements simples
* Besoin d’un orchestrateur pour gérer les traitements

# Rencontre avec Hadoop

Une plateforme à grand succès pour lancer massivement des jobs sur des systèmes distribués

**Plusieurs composants** dont :

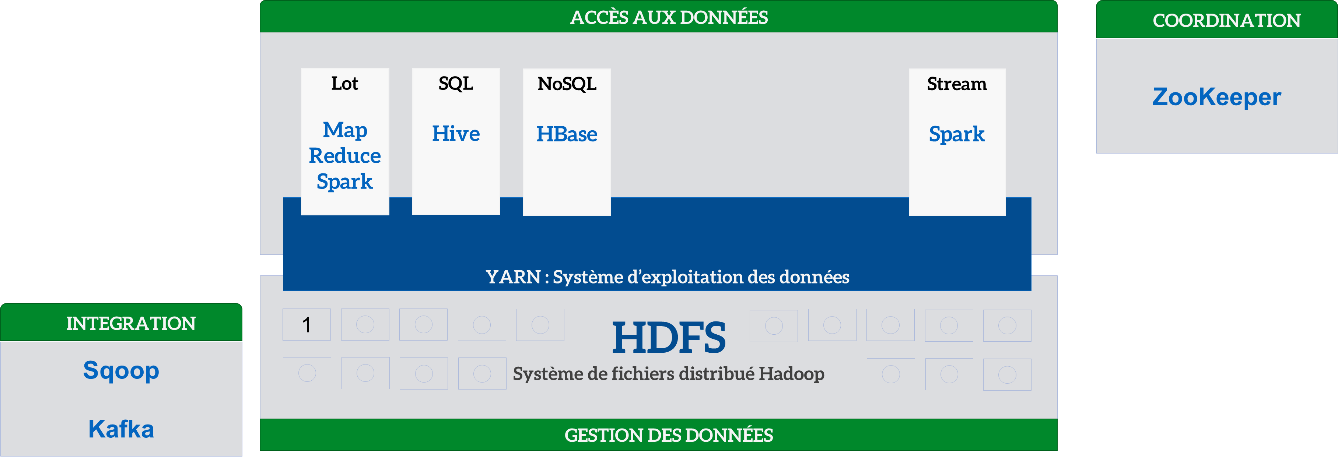
* Système de fichier distribué d’Hadoop (HDFS)
* Gestionnaire de ressources YARN
* Couche MapReduce

Hadoop est une implémentation à MapReduce & Google File System (GFS) inventé par Google utilisés dans les data centers pour principalement :

* Construction & maintenance de l’index inversé
* Exécution de l’algorithme de PageRank (de recherche)

## Ecosystème Hadoop

Ecosystème hadoop (mettant en évidence les modules qui seront traités dans le cadre de ce cours)



# HDFS

Hadoop par ses composant relève les défis qui sont imposé par le monde du Big Data, notamment HDFS qui est au cœur du framework pour la partie de stockage de données, HDFS devait donc répondre aux hypothèses et objectifs suivants :

* Etre tolérant aux pannes
* Offrir un accès aux données en continu
* Etre capable de stocker des données de tailles importantes (Tera, Peta et au-delà),
* Offrir un modèle de cohérence de données simple (une seule écriture, plusieurs lectures) -> immutable
* Etre capable de déplacer un traitement au lieu de déplacer la donnée
* Offrir une portabilité sur des systèmes hétérogènes

## Définitions

Voici quelques définitions qui décrivent brièvement HDFS

* HDFS vient de Hadoop Distributed File System
* C’est un système de fichiers qui maintient des fichiers à travers les nœuds du cluster. Il sauvegarde les fichiers, chaque fichier a un nom et situé dans un répertoire spécifique
* Il supporte la plupart des opérations connues pour les systèmes de fichier ordinaires
* Chaque cluster HDFS est composé :
  + Un ou deux **NameNode** et
  + Plusieurs **DataNode**

## NameNode :

* Gère les métadonnées et les répertoires du cluster de manière centralisée
* Il est le nœud maitre du cluster
* Détermine et maintient comment les données sont découpées et distribuées à travers le cluster
* Il est unique mais dispose d’un backup en cas de panne

## DataNode :

* Un ou plusieurs par cluster
* Il stocke et restitue une partie des données (chunk of data) et responsable de la réplication de la donnée sur les autres nœuds
* Gère les opérations sur les blocs (création, suppression)
* Nombre de **réplicas** est de 3 par défaut (mais peut être personnalisé par fichier)
* La taille par défaut d’un **bloc** est de 128MB sur la plupart des clusters

## Architecture

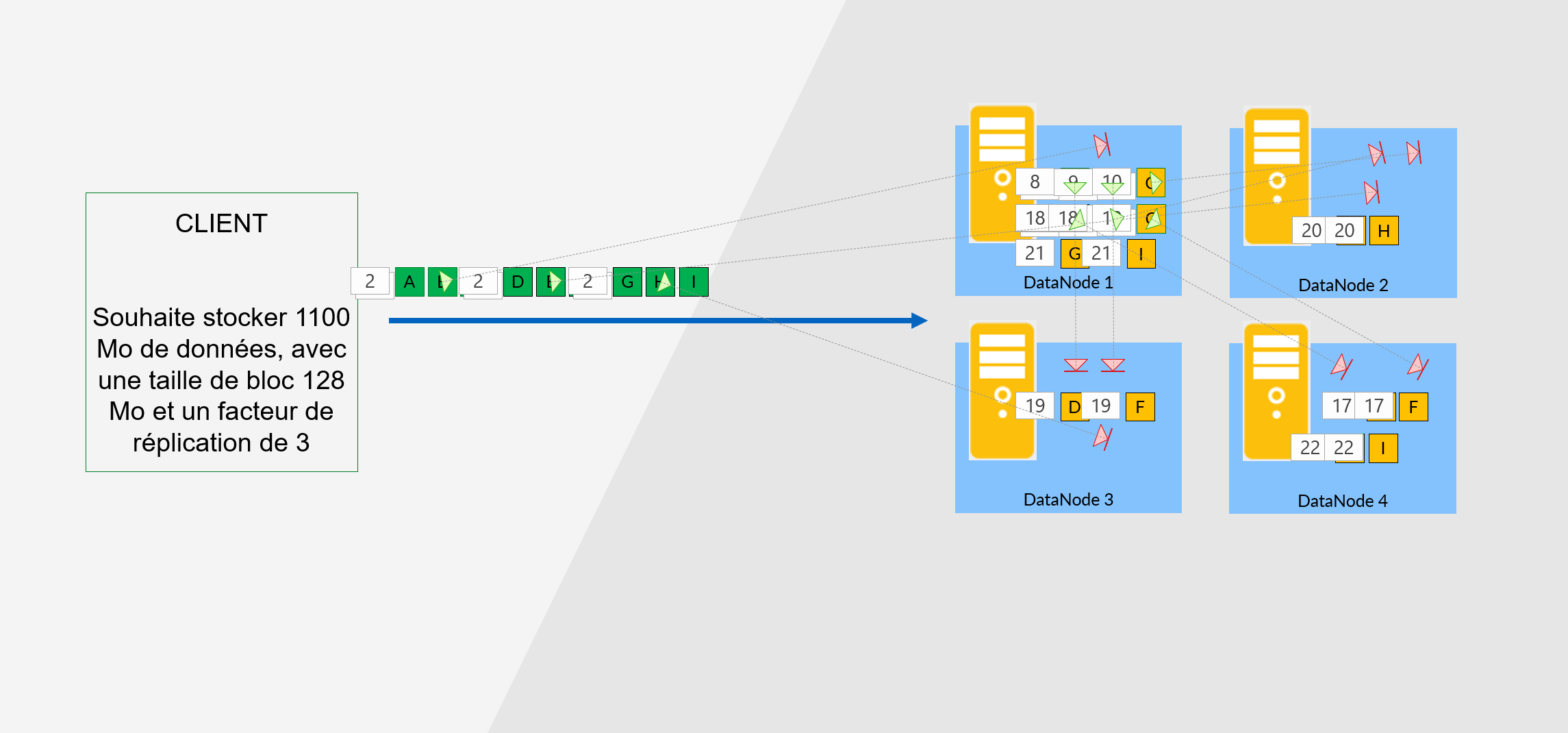


## Réplication

Pour être capable de gérer les pannes, les données doivent être répliquées, HDFS réplique les blocs du fichier sur les machines du cluster en respectant le nombre de réplicas du cluster sur lequel HDFS tourne.

Pour résumer dans rentrer dans les détails pour stocker un fichier voici ce que HDFS fait :

* Découpe le fichier en plusieurs blocs de même tailles (128Mo par défaut)
* Ecrit les blocs sur les DataNodes
* Réplique autant de fois que nécessaire chaque bloc sur les DataNodes du cluster



## Gestion des pannes

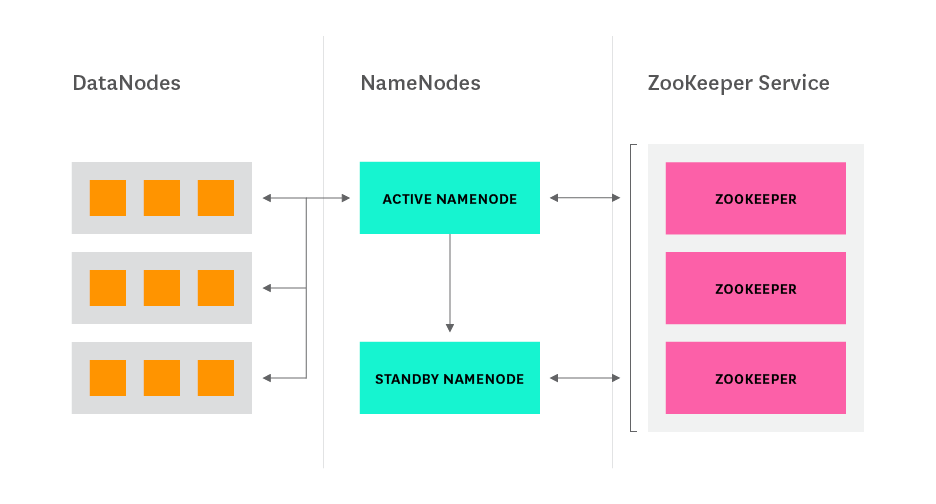
* Chaque DataNode envoi périodiquement un signal de vie « heartbeats » au NameNode
* Si le NameNode ne réceptionne pas de signal de vie au bout de X minutes, alors le DataNode sera considéré comme éteint/hors service
* Le NameNode initie la réplication des blocs qui se trouvaient sur le DataNode tombé sur un autre DataNode

## Zookeeper et interactions avec les NameNode & DataNode

ZooKeeper est un serveur open source qui coordonne de manière fiable les traitements distribués, il fournit des services opérationnels pour le cluster Hadoop :

* Un service de configuration distribué,
* Un service de synchronisation
* Un registre de nommage pour les systèmes distribués.

De plus, les applications distribuées l’utilisent pour stocker et gérer les mises à jour des informations de configuration importantes.

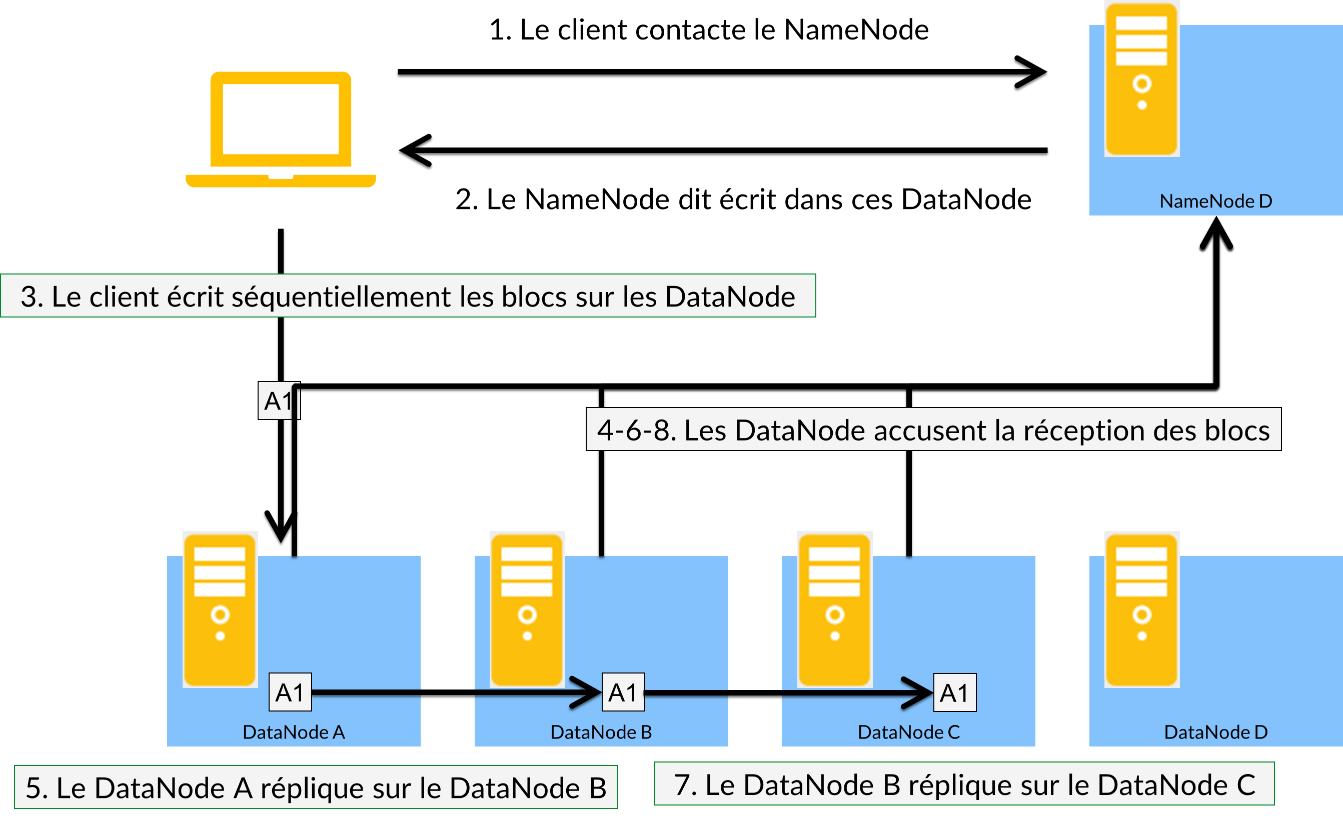


## Ecriture

Voici dans le détail comment l’écriture d’un fichier se passe dans HDFS :

1. Le client contact le NameNode pour demander à écrire un fichier
2. Le DataNode répond en fournissant toutes les informations nécessaires pour stocker
   1. Taille des blocs
   2. Sur quels nœuds seront écrits/répliqués les bloc s
   3. Etc
3. Le client commence son écriture séquentielle sur les DataNodes pour chaque bloc en fournissant les informations de réplication au DataNode
4. Le DataNode écrit le bloc chez lui et confirme au NameNode que le bloc est écrit
5. Dans la foulée demande au prochain DataNode de répliquer le bloc

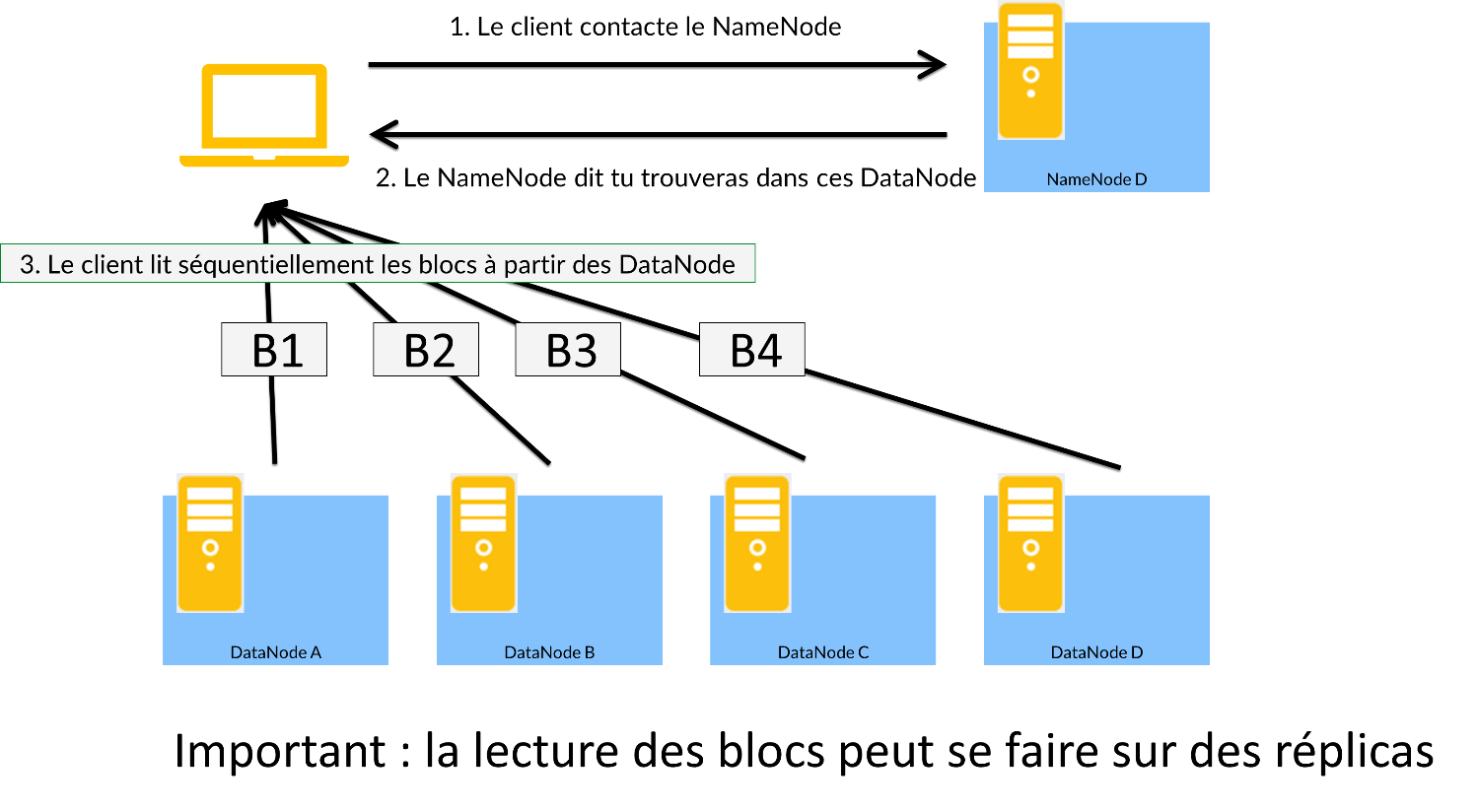
Ainsi de suite



## Lecture

Voici dans le détail comment la lecture d’un fichier se passe dans HDFS :

1. Le client contacte le NameNode
2. Le NameNode dit tu trouveras dans ces DataNode
3. Le client lit séquentiellement les blocs à partir des DataNode



# MapReduce

## Intuition

Map Reduce est un framework de traitements parallélisés, il décompose des gros traitements en un ensemble de traitements plus petits qui produisent des sous résultats -> **Map**, ensuite il agrège l’ensemble des résultats -> **Reduce**

Les tâches sont exécutées par des **Workers**

**Exemple intuitif :**

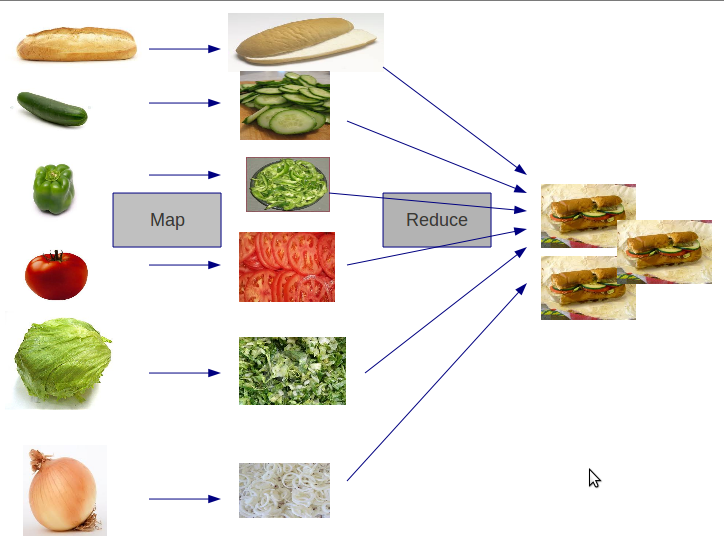
Dans une boulangerie, ou on doit produire une quantité de sandwichs conséquente pour la clientèle de bureau, intuitivement on découpe la fabrication du sandwich en plusieurs sous-taches et on affecte chacune des sous taches à un ou plusieurs employés.

On peut donc facilement avoir quelqu’un :

* Qui coupe le pain
* Qui tranche les concombres
* Qui trance les poivrons
* Etc
* Ils représentent les **mappers**

Et un dernier employé ou plusieurs s’occupent d’assembler les ingrédients pour fabriquer les sandwichs finaux

* Ils représentent les **reducers**



## En action

**Workers** : Traitement des tâches

**Master** : Orchestrateur de tâches

**Client** : Lanceur des traitements MapReduce

