

# HADOOP

# HADOOP

- Hadoop est un SGBD pour les Big Data
- Hadoop est un framework libre et open source géré par Apache et écrit en Java destiné à faciliter la création d'applications distribuées (au niveau du stockage des données et de leur traitement) et évolutives (scalables) permettant aux applications de travailler avec des milliers de nœuds et des pétaoctets de données.

# HADOOP

- Hadoop :
  - Common : des bibliothèques et utilitaires JAVA requis par d'autres modules Hadoop
  - HDFS : Hadoop Distributed File System
  - YARN : un framework pour la planification des tâches (jobs) et la gestion des ressources (*ResourceManager*)
  - MapReduce : un système basé sur YARN pour le traitement parallèle de larges volumes de données.

# HADOOP



MapReduce  
(Distributed Computation)

HDFS  
(Distributed Storage)

YARN Framework

Common Utilities

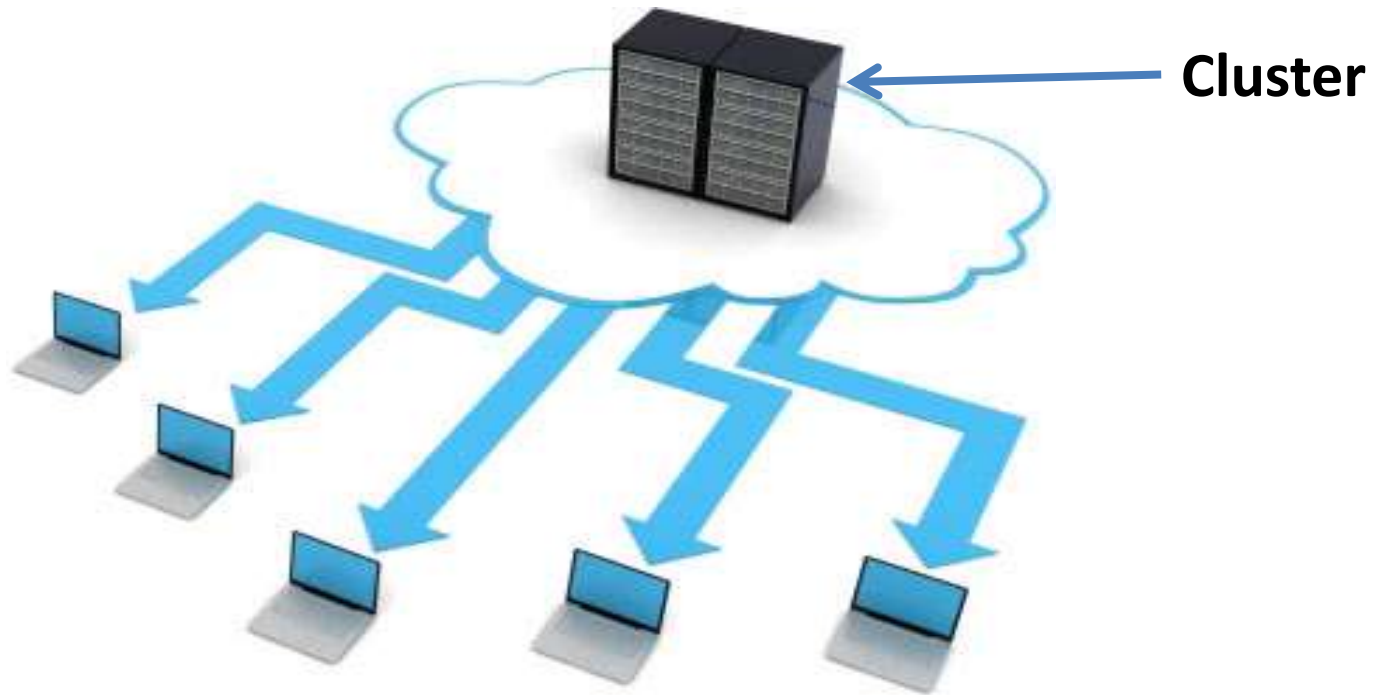
# HADOOP

- Cluster : Ensemble ou regroupement d'ordinateur indépendant, appelés nœuds, afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur pour :
  - augmenter la disponibilité ;
  - faciliter la montée en charge ;
  - permettre une répartition de la charge ;
  - faciliter la gestion des ressources (processeur, mémoire vive, disques dur, bande passante réseau).
- Le cluster apparaît comme un seul ordinateur ayant plus de performances.

# HADOOP



# HADOOP



- Nœud :
  - De stockage
  - De calcul : pour réaliser des traitements parallèles

# HDFS

- **HDFS**: Hadoop Distributed File System.
- **HDFS n'est pas une base de données mais un Système de fichiers** distribué associé à Hadoop. C'est là qu'on stocke les données d'entrée, de sortie, etc.
- Caractéristiques:
  - Distribué
  - Redondé
  - Conscient des caractéristiques physiques de l'emplacement des serveurs (racks).



# HDFS

- Qu'est-ce qu'un système de fichier **distribué**? Quel différence avec un système de fichier classique ou non distribué?

Un système de fichier distribué prend en charge toutes les fonctions de répliquations des données et de tolérance aux pannes sur un cluster.

HDFS a été conçu pour stocker des fichiers de très grandes tailles.

HDFS définit la notion de bloc qui correspond à la plus petite quantité de données qu'il est possible de lire ou d'écrire sur le disque (environ 64 Mo) beaucoup plus grande que les systèmes de fichiers ordinaires (512 octets) afin d'optimiser le temps de transfert de fichiers volumineux.

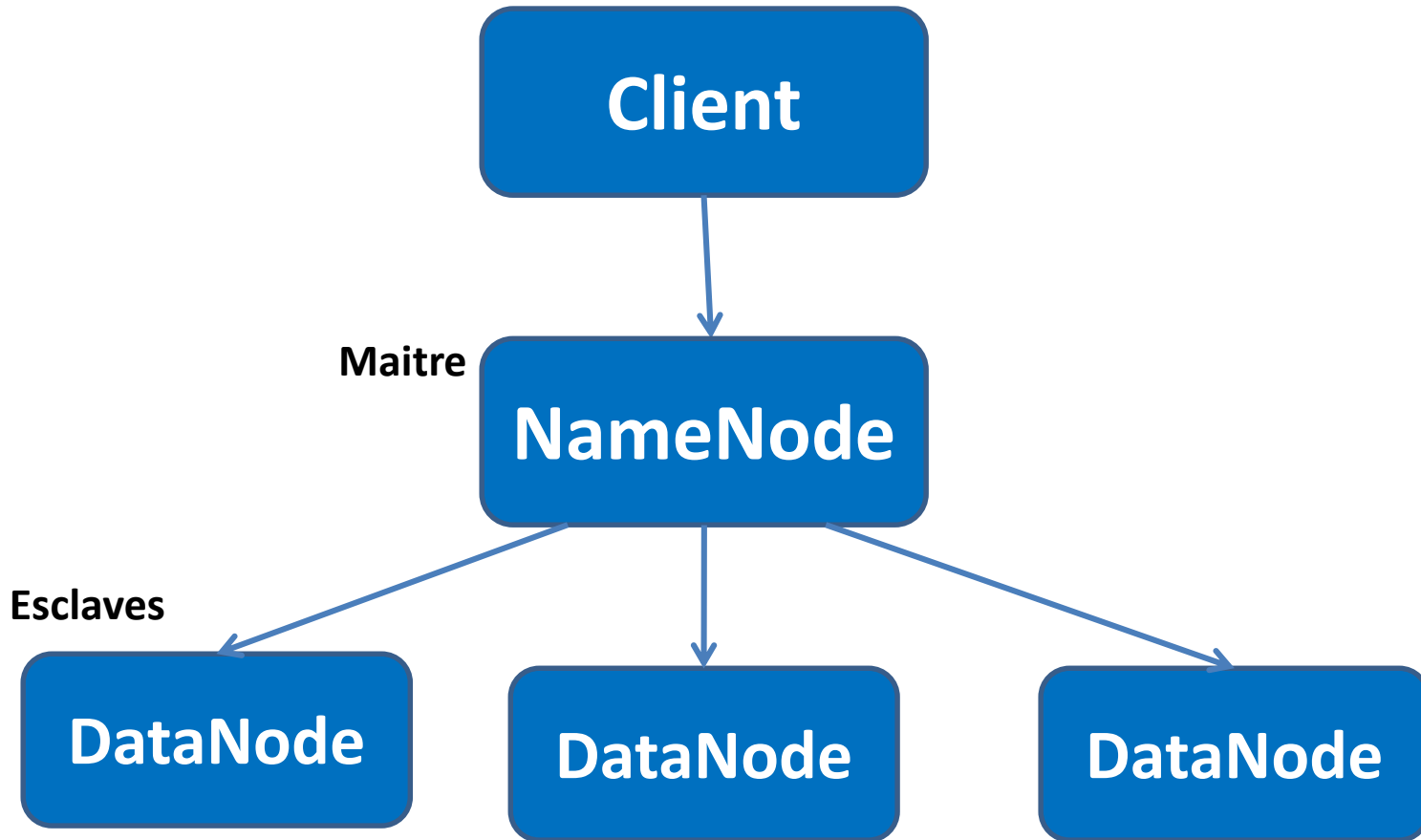
# HDFS

- Un fichier est divisé en plusieurs blocs qui seront répartis sur tout le cluster en fonction des disponibilités des machines.
- Comment l'espace de stockage est géré?

# HDFS

- Repose sur deux serveurs (système maître-esclave):
  - Le **NameNode** : unique sur le cluster. Stocke les informations relatives aux noms de fichiers et à leurs caractéristiques de manière centralisée.
  - Le **DataNode** : plusieurs par cluster. Stocke le contenu des fichiers eux-mêmes, fragmentés en blocs (64MB par défaut). Chaque bloc étant répliqué en 3 copies par défaut, d'où la tolérance aux pannes de HDFS (fault tolerance).
- Inspiré de GFS, lui-même issu de recherches de Google. « The Google File System », 2003.

# HDFS



# HDFS

- Pour écrire un fichier:
  - Le client contacte le NameNode du cluster, indiquant la taille du fichier et son nom.
  - Le NameNode confirme la demande, fragmente le fichier en blocs, et envoie tel ou tel bloc à tel ou tel DataNode.
  - Les DataNodes assurent ensuite la réplication des blocs.

# HDFS

- Pour lire un fichier:
  - Le client contacte le NameNode du cluster, indiquant le fichier qu'il souhaite obtenir.
  - Le NameNode cherche la taille, en blocs, du fichier, et pour chaque bloc une liste de DataNodes susceptibles de lui fournir.
  - Le NameNode 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 NameNode contacte un DataNode alternatif de la liste pour l'obtenir.

# HDFS

- L'ensemble du système de fichier virtuel apparaît comme un disque « unique »: on ne se soucie pas de l'emplacement réel des données.
- Tolérant aux pannes: blocs répliqués
- Inconvénient : NameNode unique → si problème sur le serveur en question, HDFS est indisponible. Pour répondre à cette problématique, un Namenode secondaire appelé Secondary Namenode a été mis en place dans l'architecture Hadoop

# MapReduce

- Automatiser le calcul parallèle: Quoi et Pourquoi?
- Contraintes liées au traitement des grands volumes de données → paralléliser le traitement sur un cluster de plusieurs centaines de milliers de machines.
- La parallélisation des traitements n'est pas une tâche facile; elle nécessite un savoir faire spécialisé et une longue phase de conception → Automatiser la parallélisation des traitements
- Un grand progrès dans ce domaine est venu de travaux de recherche de Google pour les besoins de son moteur de recherche



# MapReduce

- MapReduce est un modèle de programmation parallèle pour écrire des applications distribuées, créé par Google pour un traitement efficace de larges volumes de données (terabytes), sur de grands clusters (milliers de noeuds), de manière fiable et qui supporte les pannes.
- MapReduce signifie que chaque travail ou tâche ou job est divisé en deux fonctions : Map et Reduce.

# MapReduce

- MapReduce consiste en deux fonctions **map()** et **reduce()** :
  - Dans l'étape *Map* le nœud analyse un problème, le découpe en sous-problèmes, et les délègue à d'autres nœuds (qui peuvent en faire de même récursivement).
  - Ensuite, dans l'étape *Reduce*, les nœuds les plus bas font remonter leurs résultats au nœud parent qui les avait sollicités. Celui-ci calcule un résultat partiel à l'aide de la fonction *Reduce* (réduction) . Puis il remonte l'information à son tour. À la fin du processus, le nœud d'origine peut recomposer une réponse au problème qui lui avait été soumis.

# MapReduce

- Exemple : classe WordCount

Cette classe permet de compter le nombre d'occurrence de mots dans un fichier texte.

- Exemple d'application : exportations de fruits

Voici le contenu du fichier en entrée :

```
apple marrakech 100  
orange mango rabat 300  
orange fes meknes 100  
grapes tangier meknes 50
```

# MapReduce

- **La phase map du programme WordCount :**

1.function map(LongWritable inKey1, String inValue1)

2. foreach word w in inValue1:

3. write(w, 1)

Offset de la ligne depuis le début du  
fichier



Fichier en entrée



- Le fichier en sortie de mapper sera donc le suivant :

# MapReduce

- ***Entre la phase map et la phase reduce :***

Avant d'être envoyé au reducer, le fichier est automatiquement trié par clef par Hadoop : c'est ce que l'on appelle la phase de "shuffle & sort". Le fichier en entrée du reducer est le suivant :

# MapReduce

- **La phase Reduce :**

Liste de valeurs du type `intWritable`



1. `function reduce(Text inKey2, Iterator<intWritable> inValue2)`
2. `set wordCount = 0`
3. `foreach v in inValue2:`
4. `wordCount = wordCount + v`
5. `write(inKey2, wordCount)`

- Le fichier en sortie de reducer sera donc le suivant :

- Classe du Word Count Mapper:

```
import java.io.IOException;
```

```
import java.util.StringTokenizer;
```

```
import org.apache.hadoop.io.*;
```

```
import org.apache.hadoop.mapred.*;
```

```
public class WordCountMapper extends MapReduceBase implements Mapper<LongWritable,  
    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(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,  
        Reporter reporter) throws IOException
```

```
{
```

```
    //taking one line at a time and tokenizing the same
```

```
    String line = value.toString();
```

```
    StringTokenizer tokenizer = new StringTokenizer(line);
```

Types de l'entrée du Mapper: <Input Key Type, Input Value Type, Output Key Type, Output Value Type>

-Entrée du Mapper est une ligne  
-Sortie du Mapper: Ensemble de Clés/Valeurs

Le Reporter reporte l'état de la tâche à l'environnement HADOOP

- Classe du Word Count Mapper: suite

```
//iterating through all the words available in that line and forming the key-value pairs
while (tokenizer.hasMoreTokens())
{
    word.set(tokenizer.nextToken());
    //sending to output collector which inturn passes the same to reduce
    output.collect(word, one);
}
}
```

- Les fonctionnalités de la méthode map sont:
  1. Créer une variable 'one' de type IntWritable avec la valeur 1
  2. Convertir la ligne entrée en String
  3. Utiliser le tokenizer pour diviser la ligne en mots
  4. Itérer sur chaque mot et former une paire de clé/valeur où:
    - a. La clé est le mot en cours
    - b. La valeur est 1 (la valeur de la variable 'one')
  5. Envoyer la paire vers la sortie

```
apple,1
marrakech,1
100,1
orange,1
mango,1
Rabat,1
300,1
orange,1
fes,1
meknes,1
100,1
grapes,1
tangier,1
meknes,1
50,1
```



- Classe du Word Count Reducer :

```
import java.io.IOException;
```

```
import java.util.Iterator;
```

```
import org.apache.hadoop.io.*;
```

```
import org.apache.hadoop.mapred.*;
```

```
public class WordCountReducer extends MapReduceBase implements Reducer<Text,  
    IntWritable, Text, 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, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>  
        output, Reporter reporter)throws IOException
```

```
{
```

```
    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*/
```

```
    while (values.hasNext())
```

```
{
```

```
        sum += values.next().get();
```

```
}
```

```
    output.collect(key, new IntWritable(sum));
```

```
} }
```

# MapReduce

- Les fonctionnalités de la méthode Reduce sont:

1. Initialiser la variable 'sum' a 0
2. Itérer sur toutes les valeurs c et les sommer toutes
3. Ecrire sur la sortie la clé et la s

apple, 1  
marrakech, 1  
100, 2  
orange, 2  
mango, 1  
Rabat, 1  
300, 1  
fes,1  
meknes, 2  
Grapes, 1  
Tangier, 1  
50, 1

une clé

# MapReduce

- La classe Driver :

Cette classe est responsable du lancement de notre tâche (job).

Dans cette classe, on doit mettre:

- Le nom de notre tâche
- Les types de données d'entrée et sortie
- Les noms des classe Map et Reduce

## Classe Driver de WordCount :

```
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.conf.*;  
import org.apache.hadoop.io.*;  
import org.apache.hadoop.mapred.*;  
import org.apache.hadoop.util.*;
```

```
public class WordCount extends Configured implements Tool{  
    public int run(String[] args) throws Exception  
    {
```

```
        //creating a JobConf object and assigning a job name for identification purposes
```

```
        JobConf conf = new JobConf(getConf(), WordCount.class);
```

```
        conf.setJobName("WordCount");
```

← Créer une nouvelle tâche/job et lui donner un nom

```
        //Setting configuration object with the Data Type of output Key and Value
```

```
        conf.setOutputKeyClass(Text.class);
```

```
        conf.setOutputValueClass(IntWritable.class);
```

← Configurer le type des données de sortie : clé et valeur

```
        //Providing the mapper and reducer class names
```

```
        conf.setMapperClass(WordCountMapper.class);
```

```
        conf.setReducerClass(WordCountReducer.class);
```

← Spécifier les noms des classes Mapper et Reducer

## Classe Driver de WordCount : suite

//the hdfs input and output directory to be fetched from the command line

~~FileInputFormat.addInputPath~~(conf, **newPath**(args[0]));

FileOutputFormat.setOutputPath(conf, **newPath**(args[1]));

← Les fichiers HDFS entrée et sortie sont des arguments donnés sur la commande line

JobClient.runJob(conf);

return 0;

}

**public static void** main(String[] args) **throws** Exception

{

int res = ToolRunner.run(**new** Configuration(), **new** WordCount(),args);

System.exit(res);

}

}

# MapReduce

## Exécution:

Exécuter les étapes suivantes:

1. Compiler les 3 classes et les rassembler dans un jar et le copier dans un endroit choisi, ex.: (C:/irisi/wordcount/wordcount.jar)
2. Copier le fichier input vers (c:/irisi/wordcount/input/)
3. Copier le fichier input vers HDFS
4. Exécuter le jar:

```
hadoop jar c:/irisi/wordcount/wordcount.jar  
com.bejoy.samples.wordcount.WordCount c:/irisi/wordcount/input/  
c:/wordcount/output/
```

# Des Mappers et Reducers parallèles

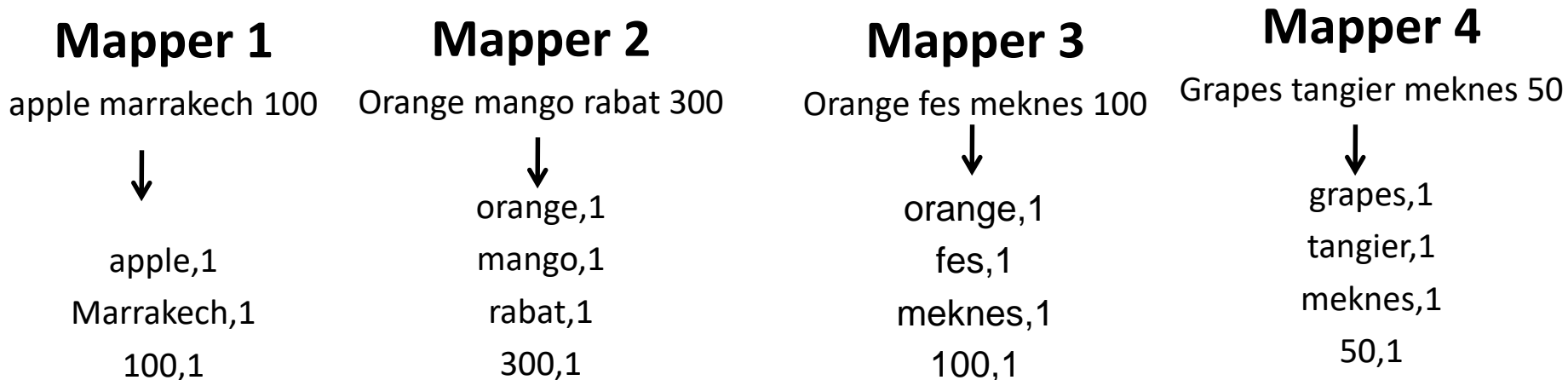
- Il faut néanmoins reconnaître qu'avec un seul mapper et un seul reducer les performances de notre programme ne seront pas meilleures que celle d'un programme classique s'exécutant sur une seule machine.
- Pour tirer parti des spécificités de Hadoop, nous allons faire évoluer notre cluster en le configurant pour qu'il dispose de :
  - Quatre mappers.
  - Deux reducers.

# Des Mappers et Reducers parallèles

- La phase map:

Chacun des quatre mappers va travailler sur une partie du fichier en entrée, par exemple :

- Le mapper n° 1 va traiter la ligne 1.
- Le mapper n° 2 va traiter la ligne 2.
- Le mapper n° 3 va traiter la ligne 3.
- Le mapper n° 4 va traiter la ligne 4





# Des Mappers et Reducers parallèles

- ***Entre la phase map et la phase reduce :***  
Avant le transfert des résultats intermédiaires des mappers vers les reducers :
- Les enregistrements sont triés par clef.
- Tous les enregistrements correspondant à une même clef sont envoyés vers un seul et même reducer.

Ainsi, Hadoop garantit :

- Que tous les enregistrements correspondant à la clef orange seront regroupés et envoyés au même reducer: ainsi, si un reducer reçoit le couple orange,1 du mapper2, alors il recevra aussi le couple orange,1 du mapper3.

# Des Mappers et Reducers parallèles

- Les fichiers en entrées des Reducers seront par exemple :

## Reducer 1

apple, [1]  
marrakech,[1]  
100,[1,1]  
orange [1,1]  
fes,[1]  
50,[1]

## Reducer 2

meknes, [1,1]  
300,[1]  
mango,[1]  
rabat [1]  
grapes,[1]  
Tangiers,(1]

- ***La phase reduce :***

Les fichiers en sortie des reducers seront alors les suivants :

**Reducer 1**

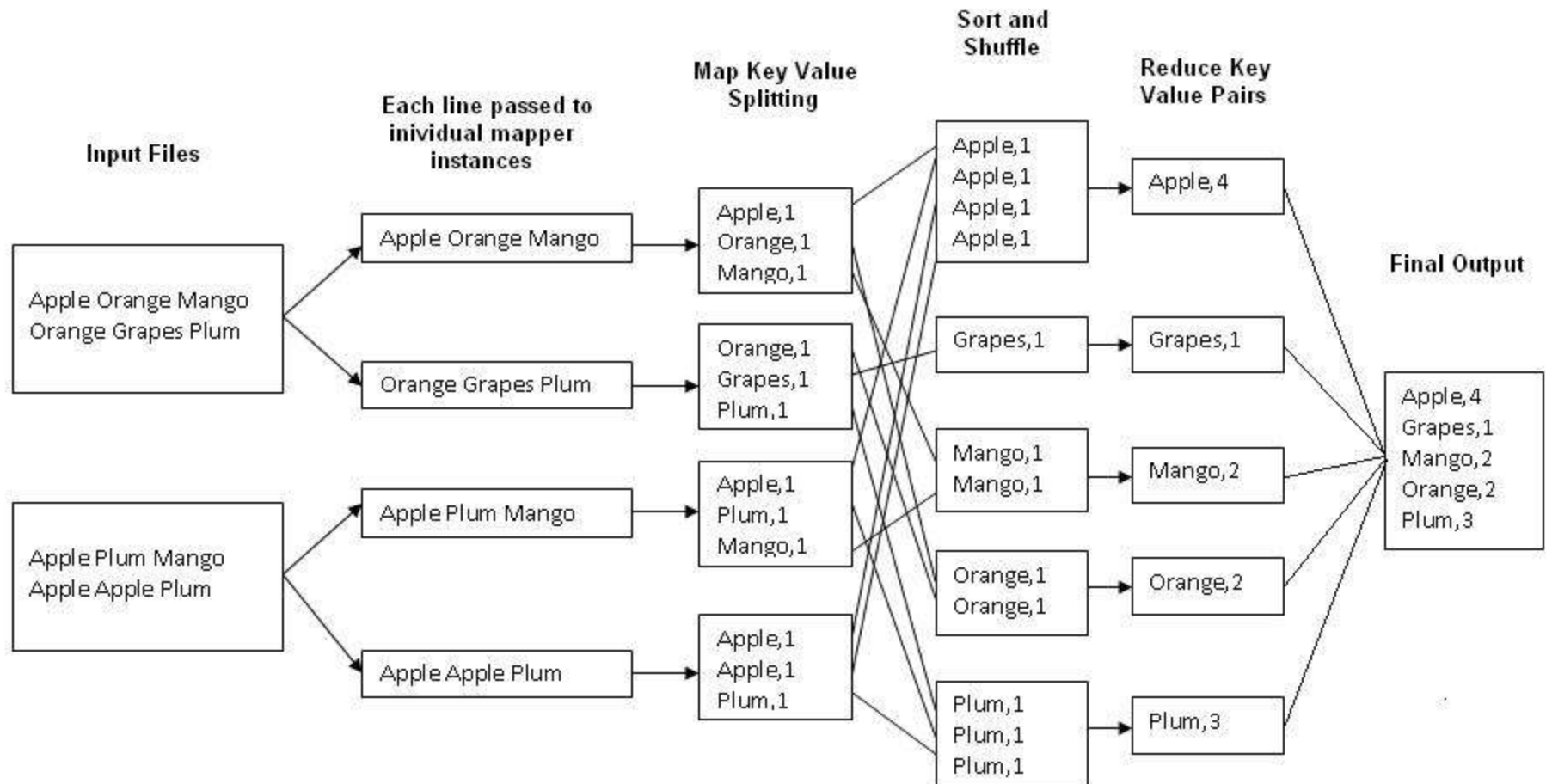
apple, 1  
marrakech,1  
100,2  
orange,2  
fes,1  
50,1

**Reducer 2**

meknes, 2  
300,1  
mango,1  
rabat,1  
grapes,1  
tangiers,1

Il ne reste plus qu'à fusionner les fichiers issus des deux reducers pour obtenir le résultat cherché.

# Des Mappers et Reducers parallèles



# MapReduce

- Par rapport au premier cluster (un mapper et un reducer), ce nouveau cluster (quatre mappers et deux reducers) permet :
  - De diviser par un facteur de l'ordre de quatre le temps d'exécution de la phase map.
  - De diviser par un facteur de l'ordre de deux le temps d'exécution de la phase reduce.
- A l'échelle d'une entreprise, cela permet un grand gain

# MapReduce

- Remarque:
- Calculer la moyenne d'un ensemble de données est un problème délicat à traiter avec Hadoop, car le calcul de la moyenne n'est pas une opération associative : si l'on divise l'ensemble des données en plusieurs parties – chacune correspondant par exemple à un mapper – la moyenne de l'ensemble ne sera généralement pas égale à la moyenne des moyennes des parties.

# MapReduce

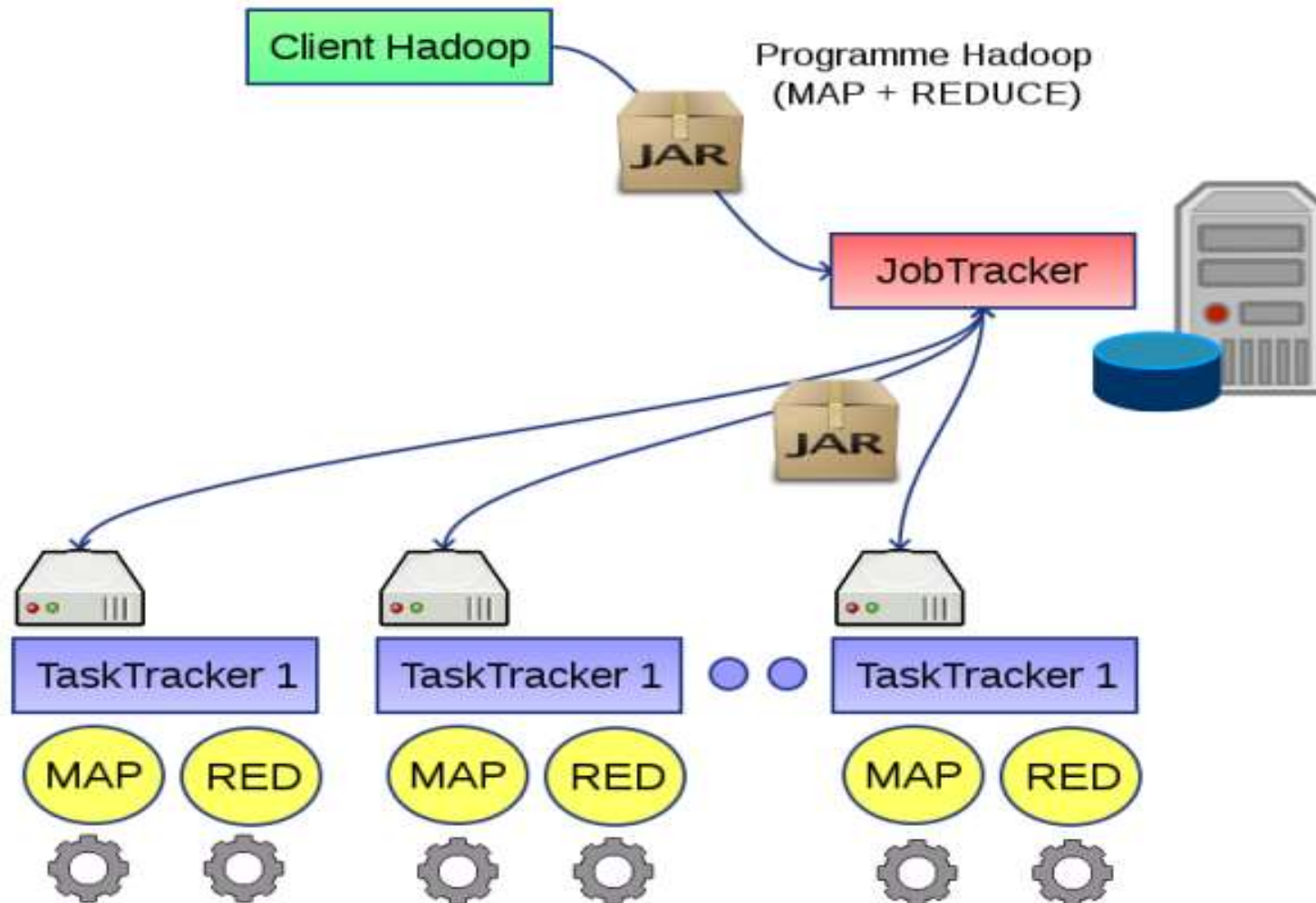
- Repose sur deux serveurs:
  - Le **JobTracker**, unique sur le cluster. Reçoit les tâches map/reduce à exécuter (sous la forme d'une archive Java .jar), organise leur exécution sur le cluster.
  - Le **TaskTracker**, plusieurs par cluster. Exécute le travail map/reduce lui-même (sous la forme de tâches map et reduce ponctuelles avec les données d'entrée associées).
- Chacun des TaskTrackers constitue une unité de calcul du cluster.

# MapReduce

- 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.
- Pour exécuter un programme map/reduce, on devra:
  - Écrire les données d'entrée sur HDFS.
  - Soumettre le programme au JobTracker du cluster.
  - Récupérer les données de sortie depuis HDFS.



# MapReduce



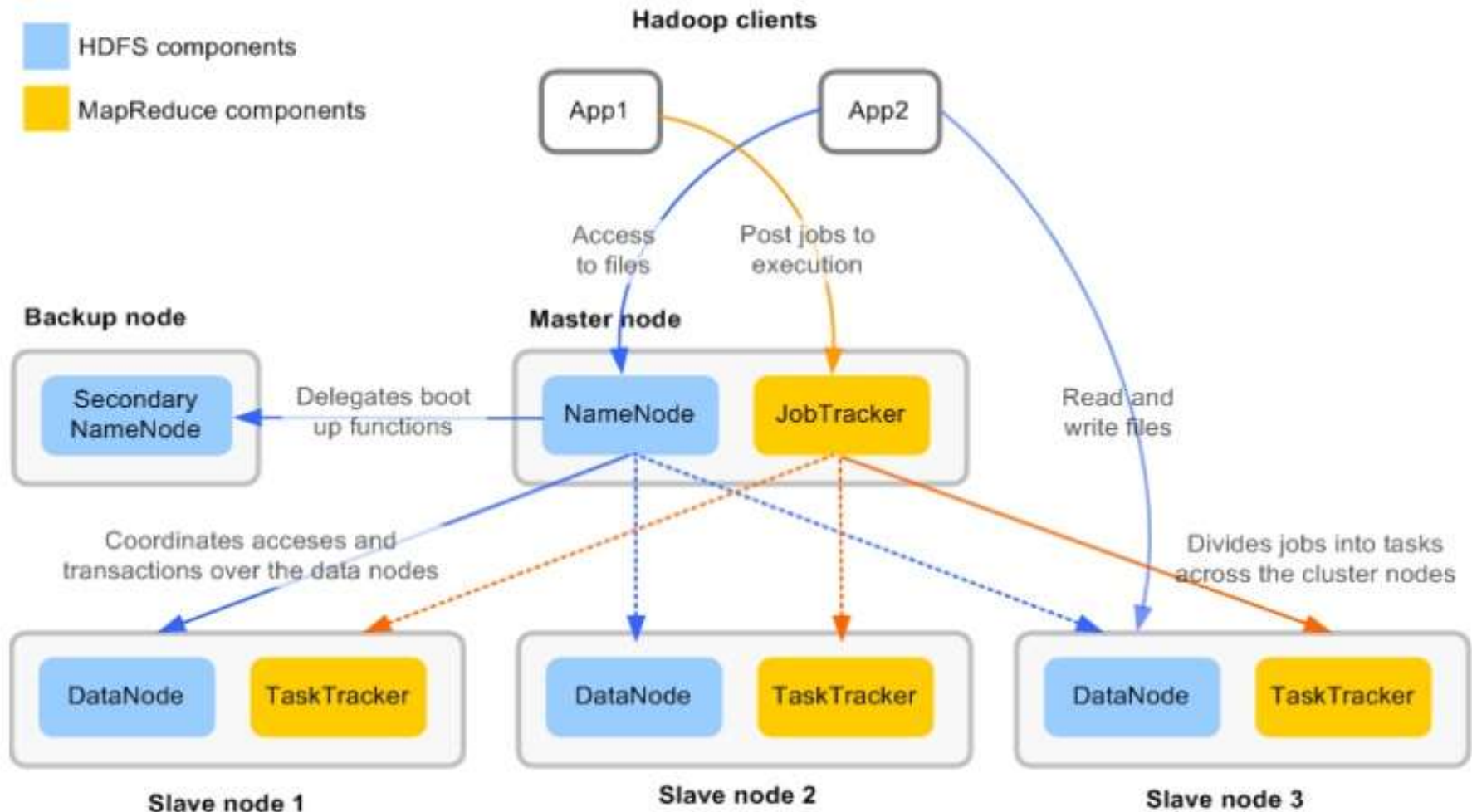
# MapReduce

- Tous les TaskTrackers signalent leur statut continuellement par le biais de paquets heartbeat.
- En cas de défaillance d'un TaskTracker (heartbeat manquant ou tâche échouée), le JobTracker avise en conséquence: redistribution de la tâche à un autre nœud, etc.
- Au cours de l'exécution d'une tâche, on peut obtenir des statistiques détaillées sur son évolution (étape actuelle, avancement, temps estimé avant completion, etc.), toujours par le biais du client console hadoop.

# MapReduce

- Inconvénient : Un seul JobTracker sur le serveur: point de défaillance unique.
- MapReduce est pour du traitement parallèle distribué.
- MapReduce n'est pas pour:
  - Les jointures
  - Les transactions

# Architecture HADOOP



# Avantages de HADOOP

- Hadoop permet aux utilisateurs de rapidement écrire et tester des systèmes distribués de manière efficace.
- Les serveurs peuvent être ajoutés ou retirés dynamiquement sans que Hadoop soit arrêté ou interrompu.
- Hadoop ne compte pas sur les serveurs eux-mêmes pour la gestion des pannes. La librairie Hadoop a été conçue pour détecter et gérer les dysfonctionnements.
- Hadoop est compatible avec toutes les plates-formes puisqu'il est écrit en JAVA.

# Qui utilise HADOOP?

The logo for Yahoo!, featuring the word "YAHOO!" in a purple, serif font.The logo for eBay, featuring the word "eBay" in a multi-colored, lowercase sans-serif font.The logo for Facebook, featuring the word "facebook" in white lowercase letters on a blue rectangular background.The logo for the Massachusetts Institute of Technology (MIT), featuring the letters "MIT" in a stylized red font with vertical bars, and the full name "Massachusetts Institute of Technology" in a smaller red font to the right.The logo for Amazon.com, featuring the word "amazon.com" in a black sans-serif font with a curved orange arrow underneath the word "amazon".The logo for Google, featuring the word "Google" in its characteristic multi-colored sans-serif font.The logo for LinkedIn, featuring the word "Linked" in black and "in" in white inside a blue square.The logo for Microsoft, featuring the four-pane Windows logo in red, green, blue, and yellow, followed by the word "Microsoft" in a grey sans-serif font.The logo for the University of California, Berkeley, featuring the word "Berkeley" in a large, black serif font, with "UNIVERSITY OF CALIFORNIA" in a smaller, black sans-serif font below it.

Et des milliers d'entreprises et universités à travers le monde.

# HADOOP

- 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.

# Installation de HADOOP

- Pré-requis :
  - JDK
- 2 types d'installation:
  - Standalone mode : Hadoop tourne comme un seul processus JAVA.
  - Single Node Cluster (mode pseudo-distribué) : une seule machine est utilisée. Chaque service (HDFS, YARN, MapReduce) tourne sur un processus JAVA distinct.
  - Multi-Node Cluster (mode distribué) : un minimum de 2 machines utilisées.



# Installation de HADOOP

## Single Node Cluster-Windows

- **Etape 1 :**

Pré-installer et configurer les logiciels suivants :

- Microsoft .NET Framework 4 (Standalone Installer)  
: <https://www.microsoft.com/en-in/download/details.aspx?id=17718>
- Windows SDK 7 Installer : <https://www.microsoft.com/en-in/download/details.aspx?id=8279>, or you can also use offline installer ISO from <https://www.microsoft.com/en-in/download/details.aspx?id=8442>. Vous allez trouver 3 ISOs différents (Choisir selon le type de votre OS) :
  - GRMSDK\_EN\_DVD.iso (x86)
  - GRMSDKX\_EN\_DVD.iso (AMD64)
  - GRMSDKIAI\_EN\_DVD.iso (Itanium)

# Installation de HADOOP

## Single Node Cluster-Windows

- **7-zip** : <http://www.7-zip.org/download.html>
- Download & extract **Maven** : <https://maven.apache.org/download.cgi>
- **ProtocolBuffer 2.5.0**  
: <https://github.com/google/protobuf/releases/download/v2.5.0/protobuf-2.5.0-win32.zip>
- **CMake** : <https://cmake.org/download/>
- **Cygwin** : <https://cygwin.com/>

# Installation de HADOOP

## Single Node Cluster-Windows

- **Etape 2 :**

Télécharger Hadoop source : **hadoop-x.x.x-src**

- **Etape 3 :**

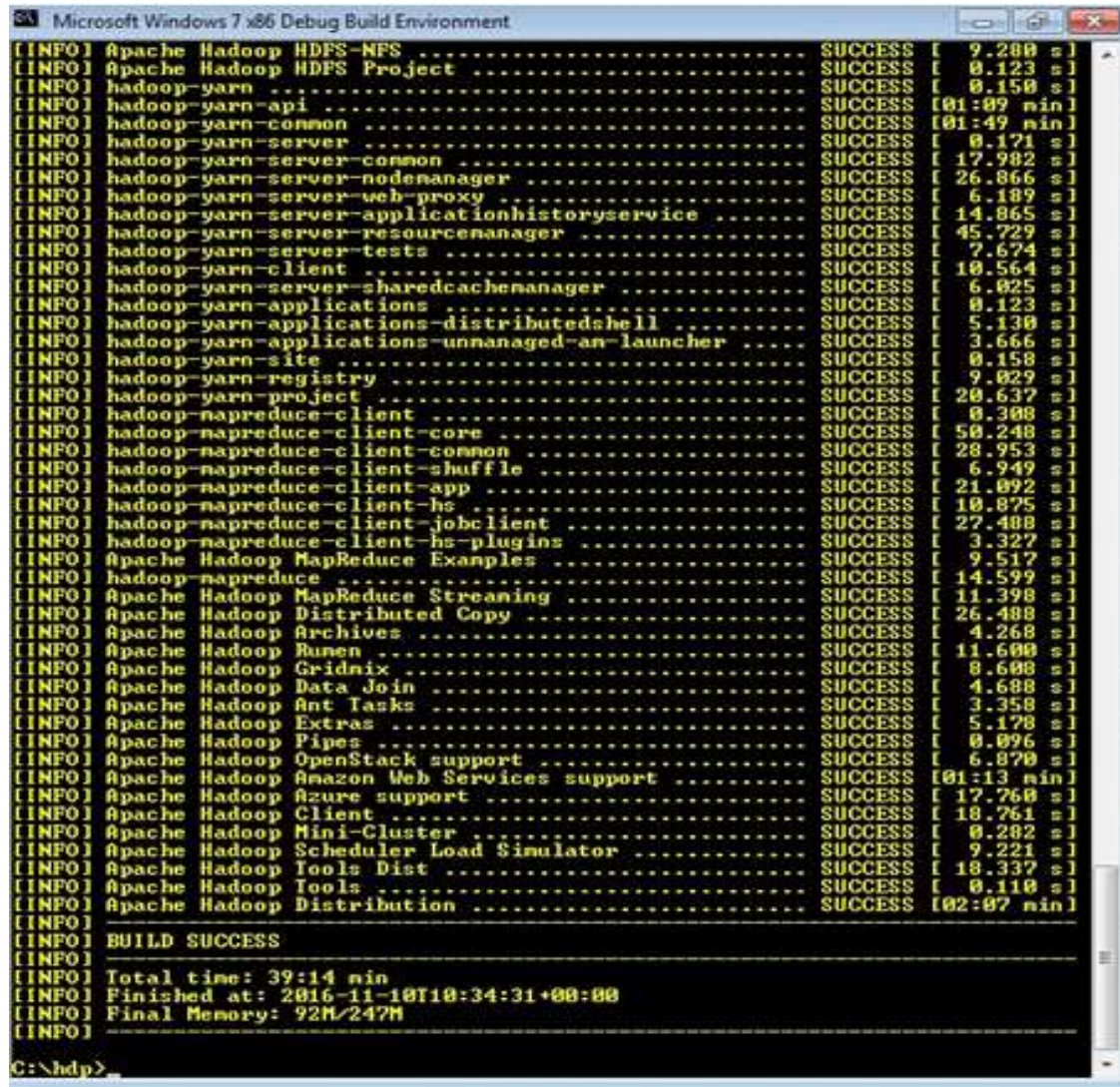
Lancer Windows SDK Command Prompt, aller au dossier contenant Hadoop source et taper la commande :

**mvn package -Pdist,native-win -DskipTests -Dtar**

L'opération prend 40 à 45 minutes.

# Installation de HADOOP

## Single Node Cluster-Windows

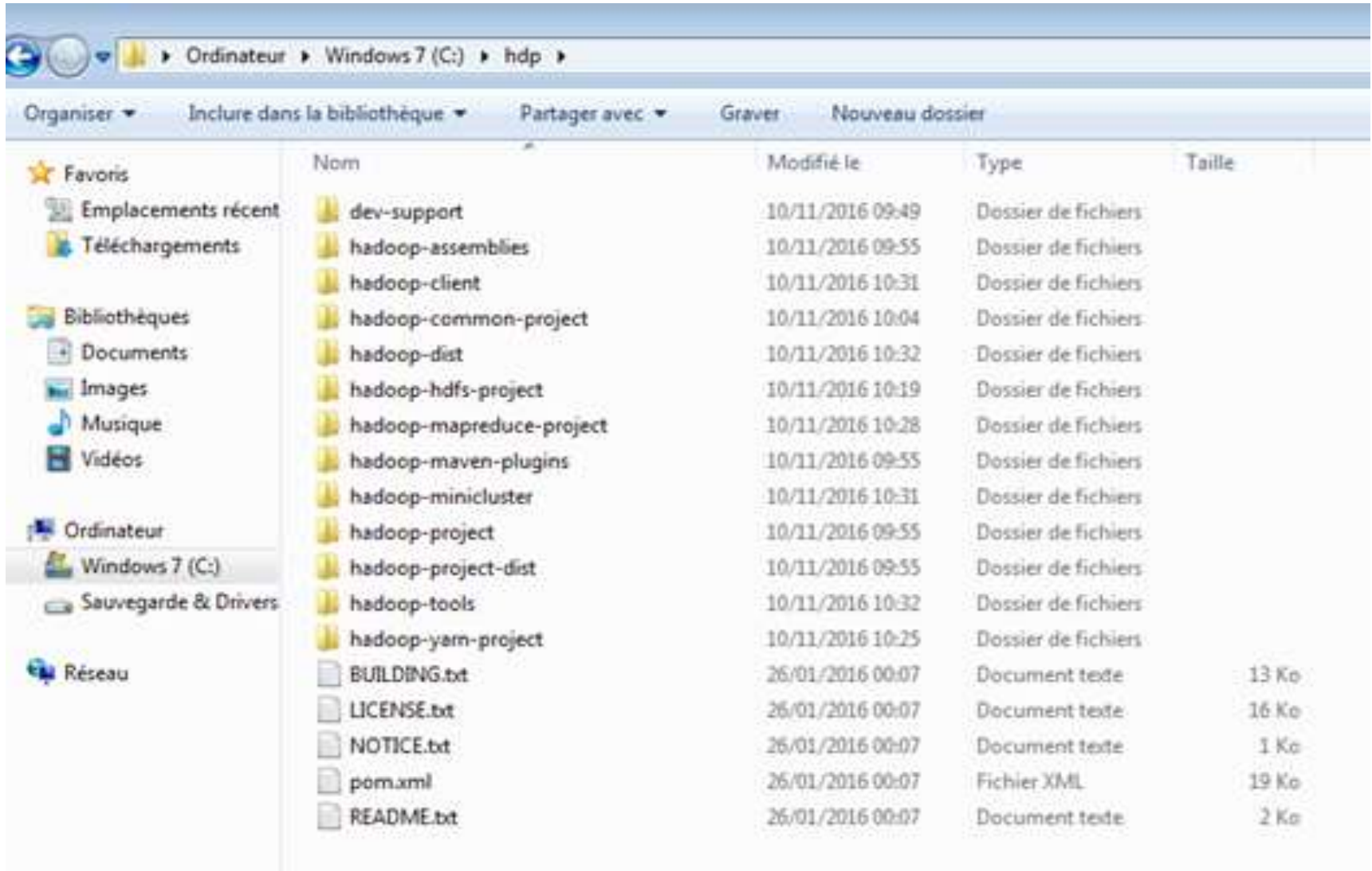


```
Microsoft Windows 7 x86 Debug Build Environment

[INFO] Apache Hadoop HDFS-NFS ..... SUCCESS [ 9.200 s]
[INFO] Apache Hadoop HDFS Project ..... SUCCESS [ 0.123 s]
[INFO] hadoop-yarn ..... SUCCESS [ 0.150 s]
[INFO] hadoop-yarn-api ..... SUCCESS [01:09 min]
[INFO] hadoop-yarn-common ..... SUCCESS [01:49 min]
[INFO] hadoop-yarn-server ..... SUCCESS [ 0.171 s]
[INFO] hadoop-yarn-server-common ..... SUCCESS [ 17.982 s]
[INFO] hadoop-yarn-server-nodemanager ..... SUCCESS [ 26.866 s]
[INFO] hadoop-yarn-server-web-proxy ..... SUCCESS [ 6.189 s]
[INFO] hadoop-yarn-server-applicationhistoryservice ..... SUCCESS [ 14.865 s]
[INFO] hadoop-yarn-server-resourcemanager ..... SUCCESS [ 45.729 s]
[INFO] hadoop-yarn-server-tests ..... SUCCESS [ 7.674 s]
[INFO] hadoop-yarn-client ..... SUCCESS [ 10.564 s]
[INFO] hadoop-yarn-server-sharedcachemanager ..... SUCCESS [ 6.025 s]
[INFO] hadoop-yarn-applications ..... SUCCESS [ 0.123 s]
[INFO] hadoop-yarn-applications-distributedshell ..... SUCCESS [ 5.130 s]
[INFO] hadoop-yarn-applications-unmanaged-an-launcher ..... SUCCESS [ 3.666 s]
[INFO] hadoop-yarn-site ..... SUCCESS [ 0.158 s]
[INFO] hadoop-yarn-registry ..... SUCCESS [ 9.029 s]
[INFO] hadoop-yarn-project ..... SUCCESS [ 20.637 s]
[INFO] hadoop-mapreduce-client ..... SUCCESS [ 0.308 s]
[INFO] hadoop-mapreduce-client-core ..... SUCCESS [ 50.248 s]
[INFO] hadoop-mapreduce-client-common ..... SUCCESS [ 28.953 s]
[INFO] hadoop-mapreduce-client-shuffle ..... SUCCESS [ 6.949 s]
[INFO] hadoop-mapreduce-client-app ..... SUCCESS [ 21.092 s]
[INFO] hadoop-mapreduce-client-hs ..... SUCCESS [ 10.875 s]
[INFO] hadoop-mapreduce-client-jobclient ..... SUCCESS [ 27.488 s]
[INFO] hadoop-mapreduce-client-hs-plugins ..... SUCCESS [ 3.327 s]
[INFO] Apache Hadoop MapReduce Examples ..... SUCCESS [ 9.517 s]
[INFO] hadoop-mapreduce ..... SUCCESS [ 14.599 s]
[INFO] Apache Hadoop MapReduce Streaming ..... SUCCESS [ 11.398 s]
[INFO] Apache Hadoop Distributed Copy ..... SUCCESS [ 26.488 s]
[INFO] Apache Hadoop Archives ..... SUCCESS [ 4.268 s]
[INFO] Apache Hadoop Runen ..... SUCCESS [ 11.600 s]
[INFO] Apache Hadoop Gridmix ..... SUCCESS [ 0.600 s]
[INFO] Apache Hadoop Data Join ..... SUCCESS [ 4.688 s]
[INFO] Apache Hadoop Ant Tasks ..... SUCCESS [ 3.358 s]
[INFO] Apache Hadoop Extras ..... SUCCESS [ 5.178 s]
[INFO] Apache Hadoop Pipes ..... SUCCESS [ 0.096 s]
[INFO] Apache Hadoop OpenStack support ..... SUCCESS [ 6.870 s]
[INFO] Apache Hadoop Amazon Web Services support ..... SUCCESS [01:13 min]
[INFO] Apache Hadoop Azure support ..... SUCCESS [ 17.760 s]
[INFO] Apache Hadoop Client ..... SUCCESS [ 18.761 s]
[INFO] Apache Hadoop Mini-Cluster ..... SUCCESS [ 0.202 s]
[INFO] Apache Hadoop Scheduler Load Simulator ..... SUCCESS [ 9.221 s]
[INFO] Apache Hadoop Tools Dist ..... SUCCESS [ 18.337 s]
[INFO] Apache Hadoop Tools ..... SUCCESS [ 0.110 s]
[INFO] Apache Hadoop Distribution ..... SUCCESS [02:07 min]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 39:14 min
[INFO] Finished at: 2016-11-10T10:34:31+00:00
[INFO] Final Memory: 92M/247M
[INFO]
C:\hdp>
```

# Installation de HADOOP

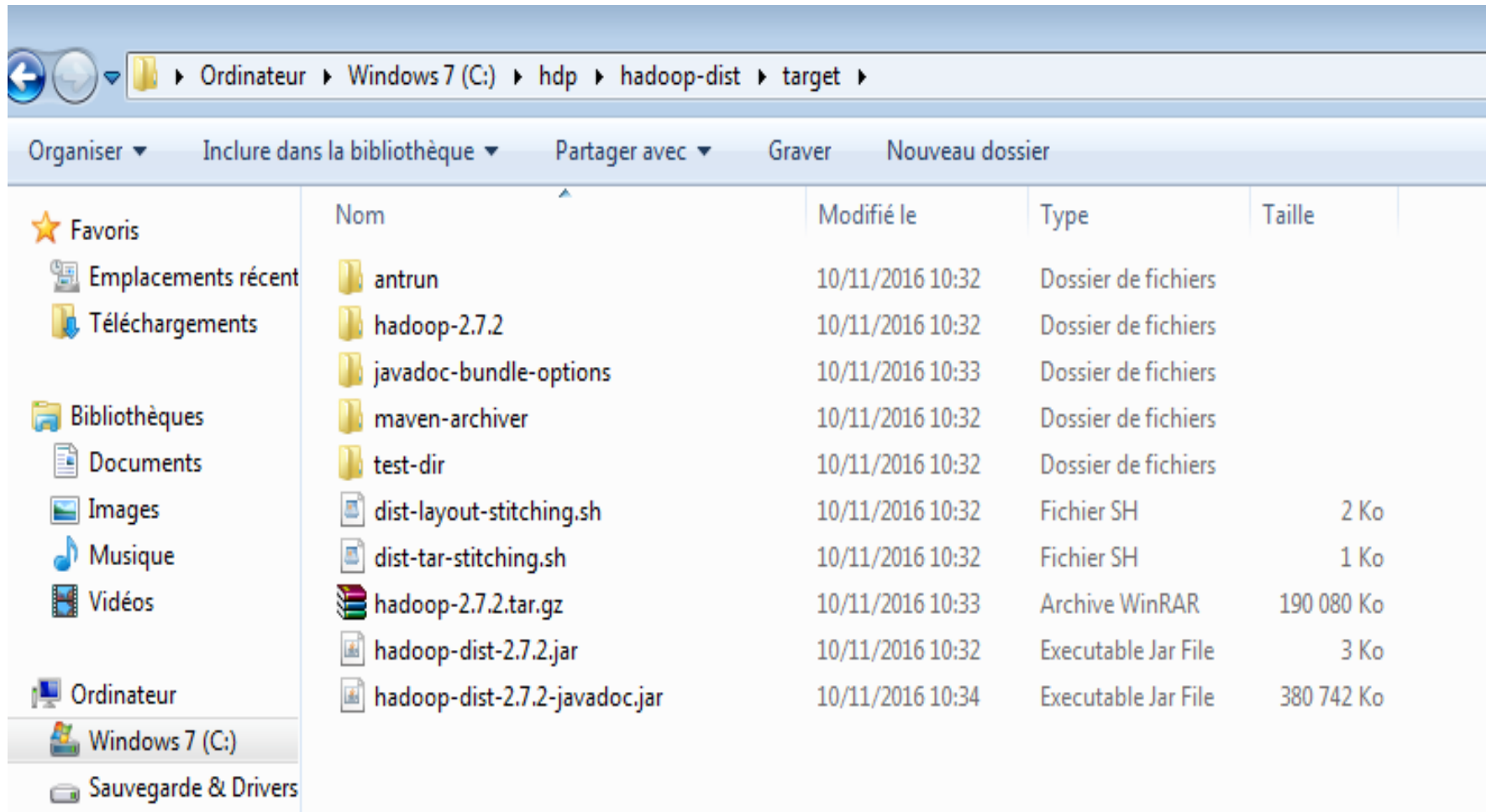
## Single Node Cluster-Windows



Nom	Modifié le	Type	Taille
dev-support	10/11/2016 09:49	Dossier de fichiers	
hadoop-assemblies	10/11/2016 09:55	Dossier de fichiers	
hadoop-client	10/11/2016 10:31	Dossier de fichiers	
hadoop-common-project	10/11/2016 10:04	Dossier de fichiers	
hadoop-dist	10/11/2016 10:32	Dossier de fichiers	
hadoop-hdfs-project	10/11/2016 10:19	Dossier de fichiers	
hadoop-mapreduce-project	10/11/2016 10:28	Dossier de fichiers	
hadoop-maven-plugins	10/11/2016 09:55	Dossier de fichiers	
hadoop-minicluster	10/11/2016 10:31	Dossier de fichiers	
hadoop-project	10/11/2016 09:55	Dossier de fichiers	
hadoop-project-dist	10/11/2016 09:55	Dossier de fichiers	
hadoop-tools	10/11/2016 10:32	Dossier de fichiers	
hadoop-yarn-project	10/11/2016 10:25	Dossier de fichiers	
BUILDING.txt	26/01/2016 00:07	Document texte	13 Ko
LICENSE.txt	26/01/2016 00:07	Document texte	16 Ko
NOTICE.txt	26/01/2016 00:07	Document texte	1 Ko
pom.xml	26/01/2016 00:07	Fichier XML	19 Ko
README.txt	26/01/2016 00:07	Document texte	2 Ko

# Installation de HADOOP

## Single Node Cluster-Windows





# Installation de HADOOP

## Single Node Cluster-Windows

Name	Size	Packed Size	Modified	Mode	User
hdfs	12 223	12 288	2016-01-26 05:50	0rwxr-xr-x	jenkins
mapred.cmd	6 310	6 656	2016-01-26 05:50	0rwxr-xr-x	jenkins
mapred	5 953	6 144	2016-01-26 05:50	0rwxr-xr-x	jenkins
yarn.cmd	11 386	11 776	2016-01-26 05:50	0rwxr-xr-x	jenkins
rcc	1 776	2 048	2016-01-26 05:50	0rwxr-xr-x	jenkins
hdfs.cmd	7 478	7 680	2016-01-26 05:50	0rwxr-xr-x	jenkins
yarn	13 352	13 824	2016-01-26 05:50	0rwxr-xr-x	jenkins
hadoop.cmd	8 786	9 216	2016-01-26 05:50	0rwxr-xr-x	jenkins
hadoop	6 488	6 656	2016-01-26 05:50	0rwxr-xr-x	jenkins
test-container-executor	205 195	205 312	2016-01-26 05:50	0rwxr-xr-x	jenkins
container-executor	160 351	160 768	2016-01-26 05:50	0rwxr-xr-x	jenkins

Name	Size	Packed Size	Modified	Mode	User
hadoop	6 488	6 656	2016-04-08 15:04	0rwxrwxr-x	admin
hadoop.cmd	8 786	9 216	2016-04-08 15:04	0rwxrwxr-x	admin
hadoop.dll	86 016	86 016	2016-04-08 15:04	0rwxr-x---	admin
hadoop.exp	17 069	17 408	2016-04-08 15:04	0rwxr-x---	admin
hadoop.lib	28 808	29 184	2016-04-08 15:04	0rwxr-x---	admin
hadoop.pdb	478 208	478 208	2016-04-08 15:04	0rwxr-x---	admin
hdfs	12 223	12 288	2016-04-08 15:04	0rwxrwxr-x	admin
hdfs.cmd	7 478	7 680	2016-04-08 15:04	0rwxrwxr-x	admin
hdfs.dll	60 416	60 416	2016-04-08 15:04	0rwxr-x---	admin
hdfs.exp	8 895	9 216	2016-04-08 15:04	0rwxr-x---	admin
hdfs.lib	15 030	15 360	2016-04-08 15:04	0rwxr-x---	admin
hdfs.pdb	355 328	355 328	2016-04-08 15:04	0rwxr-x---	admin
hdfs_static.lib	344 038	344 064	2016-04-08 15:04	0rwxr-x---	admin
libwinutils.lib	1 235 894	1 235 968	2016-04-08 15:04	0rwxr-x---	admin
mapred	5 953	6 144	2016-04-08 15:04	0rwxr-x---	admin
mapred.cmd	6 310	6 656	2016-04-08 15:04	0rwxr-x---	admin
rcc	1 776	2 048	2016-04-08 15:04	0rwxrwxr-x	admin
winutils.exe	109 568	109 568	2016-04-08 15:04	0rwxr-x---	admin
winutils.pdb	896 000	896 000	2016-04-08 15:04	0rwxr-x---	admin
yarn	13 352	13 824	2016-04-08 15:04	0rwxr-x---	admin
yarn.cmd	11 386	11 776	2016-04-08 15:04	0rwxr-x---	admin

# Configuration de HADOOP

## Single Node Cluster-Windows

- **core-site.xml : etc/hadoop**

```
<configuration>  
  <property>  
    <name>fs.default.name</name>  
    <value>hdfs://0.0.0.0:19000</value>  
  </property>  
</configuration>
```

- **hdfs-site.xml : etc/hadoop**

```
<configuration>  
  <property>  
    <name>dfs.replication</name>  
    <value>1</value>  
  </property>  
</configuration>
```



# Configuration de HADOOP

## Single Node Cluster-Windows

- **Créer mapred-site.xml : etc/hadoop . Remplacer %USERNAME% par votre nom d'utilisateur**

```
<configuration>

  <property>
    <name>mapreduce.job.user.name</name>
    <value>%USERNAME%</value>
  </property>

  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>

  <property>
    <name>yarn.apps.stagingDir</name>
    <value>/user/%USERNAME%/staging</value>
  </property>

  <property>
    <name>mapreduce.jobtracker.address</name>
    <value>local</value>
  </property>

</configuration>
```

# Configuration de HADOOP

## Single Node Cluster-Windows

- **yarn-site.xml : voir la page "Build and Install Hadoop 2.x or newer on Windows"**

# Configuration de HADOOP

## Single Node Cluster-Windows

- **Formater le système de fichier HDFS :**

`bin\hdfs namenode -format`

- **Lancer HDFS :**

`sbin\start-dfs.cmd`

- **Lancer YARN :**

`sbin\start-yarn.cmd`

- **Tester les services avec la commande : `jps`**

**et sur : `http:// localhost:8088`**

**et `http:// localhost:50070`**

# Configuration de HADOOP

## Single Node Cluster-Windows

- Liste des commandes Hadoop :

Hadoop fs

- Copier un fichier dans HDFS :

hadoop fs -mkdir /demo

hadoop fs -put C:\fich.txt /demo/demoinput

- Fusionner 2 fichiers :

hadoop fs -appendToFile C:\fich2.txt /demo/demoinput

# Configuration de HADOOP

## Single Node Cluster-Windows

- Liste des jobs pré-installés :

**hadoop-x.x.x\share\hadoop\mapreduce > dir**

```
Directory of C:\work\hadoop-2.5.2\share\hadoop\mapreduce

09/10/2015  05:40 AM    <DIR>          .
09/10/2015  05:40 AM    <DIR>          ..
11/15/2014  05:23 AM             491,332  hadoop-mapreduce-client-app-2.5.2.jar
11/15/2014  05:23 AM             662,945  hadoop-mapreduce-client-common-2.5.2.jar
11/15/2014  05:23 AM            1,497,922  hadoop-mapreduce-client-core-2.5.2.jar
11/15/2014  05:23 AM             233,035  hadoop-mapreduce-client-hs-2.5.2.jar
11/15/2014  05:23 AM             4,064  hadoop-mapreduce-client-hs-plugins-2.5.2.
jar
11/15/2014  05:23 AM            1,487,215  hadoop-mapreduce-client-jobclient-2.5.2-t
ests.jar
11/15/2014  05:23 AM             35,721  hadoop-mapreduce-client-jobclient-2.5.2.j
ar
11/15/2014  05:23 AM             43,655  hadoop-mapreduce-client-shuffle-2.5.2.jar
11/15/2014  05:23 AM            270,323  hadoop-mapreduce-examples-2.5.2.jar
09/10/2015  05:40 AM    <DIR>          lib
09/10/2015  05:40 AM    <DIR>          lib-examples
09/10/2015  05:40 AM    <DIR>          sources
          9 File(s)          4,726,212 bytes
          5 Dir(s)  27,841,724,416 bytes free
```

# Configuration de HADOOP

## Single Node Cluster-Windows

- Liste des jobs pré-installés :

`hadoop-x.x.x\share\hadoop\mapreduce > hadoop jar  
hadoop-mapreduce-examples-x.x.x`

```
aggregatewordhist: An Aggregate based map/reduce program that computes the histogram of the words in the input files.  
bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.  
dbcount: An example job that count the pageview counts from a database.  
distbbp: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.  
grep: A map/reduce program that counts the matches of a regex in the input.  
join: A job that effects a join over sorted, equally partitioned datasets  
multifilewc: A job that counts words from several files.  
pentomino: A map/reduce tile laying program to find solutions to pentomino problems.  
pi: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.  
randomtextwriter: A map/reduce program that writes 10GB of random textual data per node.  
randomwriter: A map/reduce program that writes 10GB of random data per node.  
secondarysort: An example defining a secondary sort to the reduce.  
sort: A map/reduce program that sorts the data written by the random writer.  
sudoku: A sudoku solver.  
teragen: Generate data for the terasort  
terasort: Run the terasort  
teravalidate: Checking results of terasort  
wordcount: A map/reduce program that counts the words in the input files.  
wordmean: A map/reduce program that counts the average length of the words in the input files.  
wordmedian: A map/reduce program that counts the median length of the words in the input files.
```

# Configuration de HADOOP

## Single Node Cluster-Windows

- Exécuter un job pré-installés :

```
hadoop-x.x.x\share\hadoop\mapreduce > hadoop jar  
hadoop-mapreduce-examples-x.x.x wordcount  
/demo/demoinput /demo/wordcount-op
```

- Visualiser le résultat du job :

```
hadoop fs -cat /demo/wordcount-op/*
```

# Configuration de HADOOP

## Single Node Cluster-Windows

- **Fermer HDFS et YARN :**

`sbin\stop-dfs.cmd`

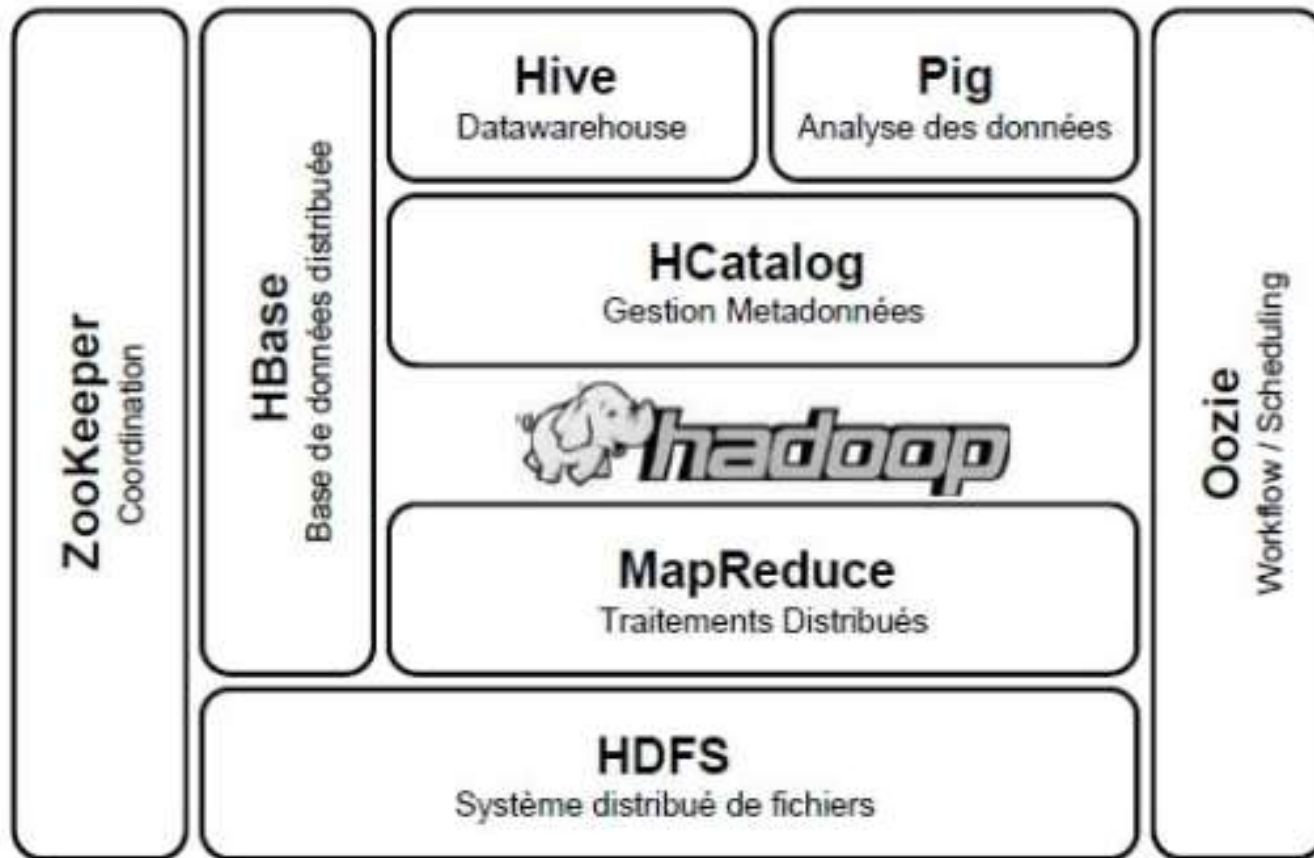
`sbin\stop-yarn.cmd`



# Distributions HADOOP

- Ecosystème HADOOP: Plate-forme qui regroupe un ensemble d'outils logiciels autour de HADOOP.
- HortonWorks
- Cloudera
- MapR
- Syncfusion

# Distributions HADOOP



# Ecosystème HADOOP

- **HDFS (Hadoop Distributed File System)** : C'est le **Système de fichiers** distribué de Hadoop. Il stocke au format natif tout type de données.
- **HBASE : Base de données** NoSQL orientée colonnes. Le stockage à proprement parler reste du HDFS mais HBASE apporte une surcouche de stockage qui permet de bénéficier des avantages des bases orientées colonnes, à savoir **les colonnes dynamiques**.
- **HIVE** : Hive est un langage de requêtage basé sur le SQL pour la définition et l'écriture des tâches ('jobs' dans le jargon) MapReduce. Il fonctionne aussi bien en mode interactif qu'en mode Batch. Il fournit à l'utilisateur familier du SQL un langage de requêtage similaire appelée "HiveQL".
- **PIG** : Moteur permettant de manipuler tous types de données avec un langage beaucoup plus intuitif que le java mapreduce. Une fois exécuté le code Pig génère du mapreduce.

# Ecosystème HADOOP-suite

- **Sqoop** : Sqoop ou SQL-to-Hadoop est un outil de l'écosystème qui permet de transférer les données d'une base de données relationnelle au HDFS d'Hadoop et vice-versa.
- **Zookeeper** : permet la coordination des processus distribués au moyen d'un espace de nommage partagé. C'est un projet indépendant de Hadoop. Beaucoup d'applications distribuées utilisent des services de synchronisation de type Zookeeper qui offre un registre de nommage pour les applications distribuées afin de coordonner les jobs distribués (service de nommage, système de configuration distribué, mécanisme de synchronisation distribué, système de file de message, système de notification).
- **Spark** : Spark fournit une bibliothèque de classes d'implémentation Java d'algorithmes analytiques pour l'exécution dans un cluster Hadoop. Ces classes peuvent être exploitées à l'aide d'un langage de programmation comme Scala, Java, Python ou tout autre langage compatible avec Spark.

# HBASE



# HBASE

- Limitations de HADOOP: Hadoop peut uniquement exécuter des requêtes en mode batch ou offline, et les données sont accessibles uniquement en mode séquentiel (à l'opposé du mode random access) → vous devez parcourir toute la base de données même pour une tâche simple.
- Hbase (tout comme MongoDB, Cassandra, etc) permettent un *accès aléatoire* aux données, appelé aussi *accès direct*.

# HBASE

- **Hbase est une base de données orientée colonnes** déployée au dessus de Hadoop-HDFS, open source et horizontalement évolutive.
- Hbase permet un accès direct et en temps réel aux données.



# HBASE

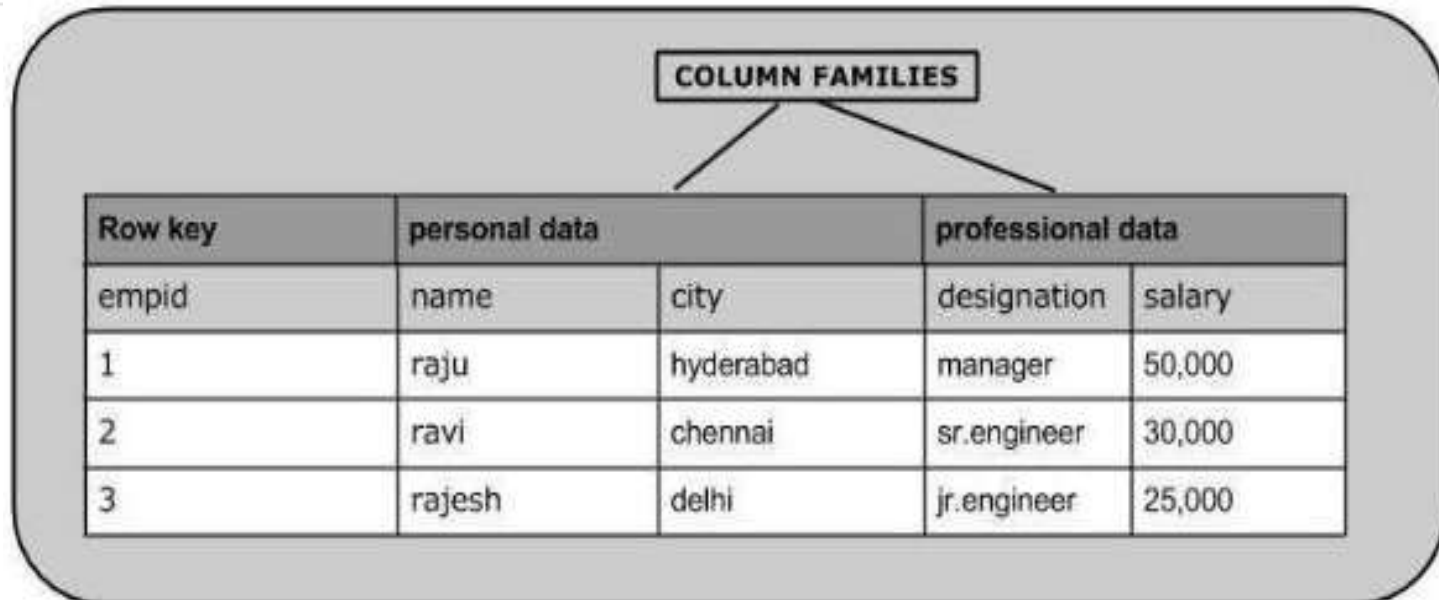
- Les tables dans Hbase sont triés par ligne.
- Le schéma de chaque table définit uniquement les familles des colonnes.
- Chaque table possède plusieurs familles de colonnes et chaque famille de colonne peut avoir plusieurs colonnes. En définitive:
  - Une table est une collection de lignes.
  - Une ligne est une collection de familles de colonnes.
  - Une famille de colonne est une collection de colonnes.
  - Une colonne est une collection de paires clé/valeur.



# HBASE

- Exemple de table sous Hbase :

Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												



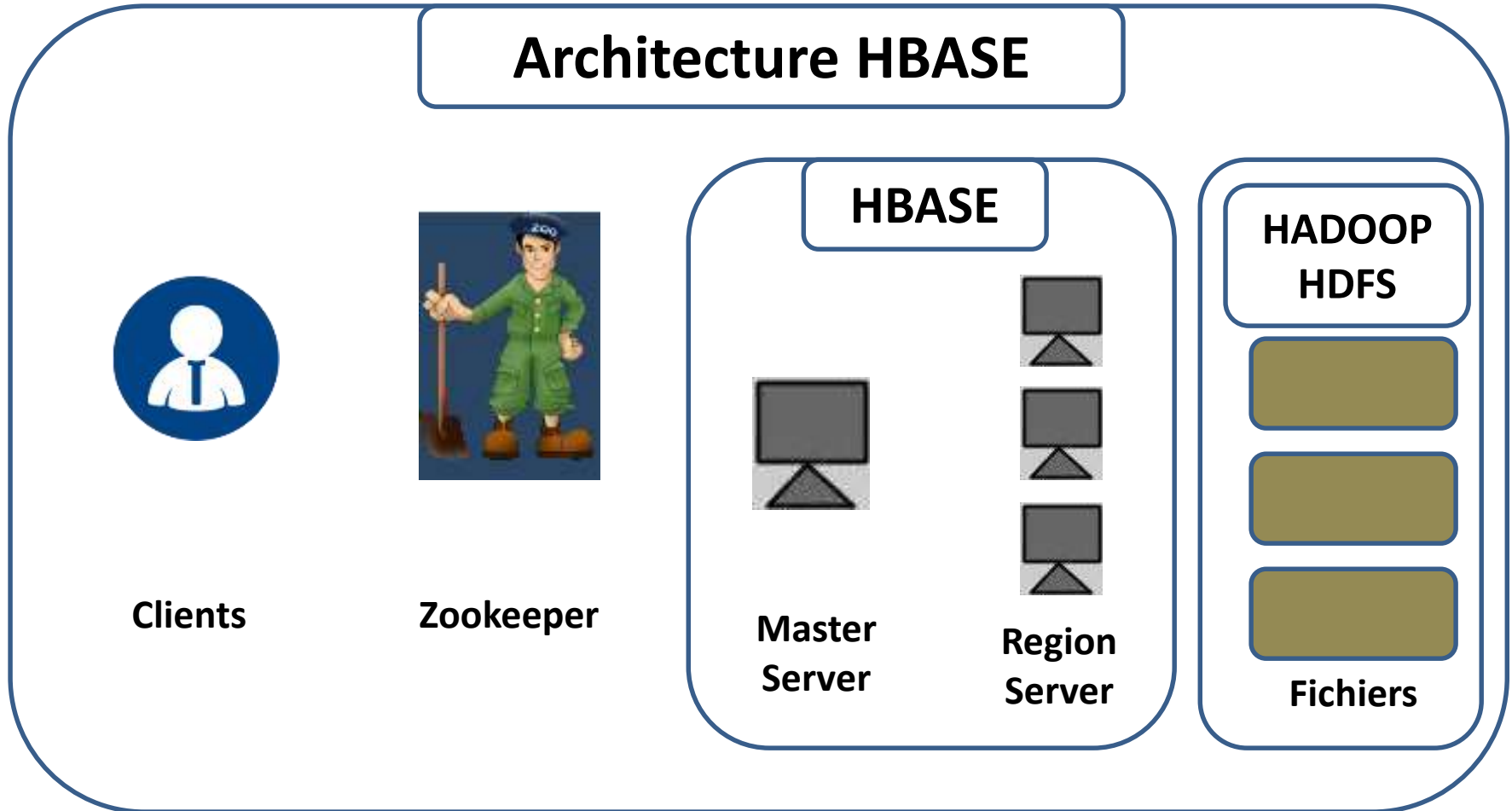
# HBASE

- HBASE vs RDBM :

RDBM	HBASE
Destiné aux données structurées	Destiné aux données structurées et semi-structurées
Géré par son schéma qui décrit la structure des tables.	Ne possède pas de schéma. Pas de structure fixe. Définit uniquement les familles de colonnes.
Destiné aux faibles volumes de données. Non évolutif.	Destiné aux grands volumes. Horizontalement évolutif.
Transactionnel. Doté de jointures.	Pas de transaction, pas de jointures.

- Apache HBase est utilisé pour avoir un accès read/write aléatoire (direct) et temps réel aux données big Data.
- Des sociétés comme Facebook, Twitter, Yahoo, et Adobe utilisent Hbase.

# Architecture HBASE



# HBASE-ZOOKEEPER

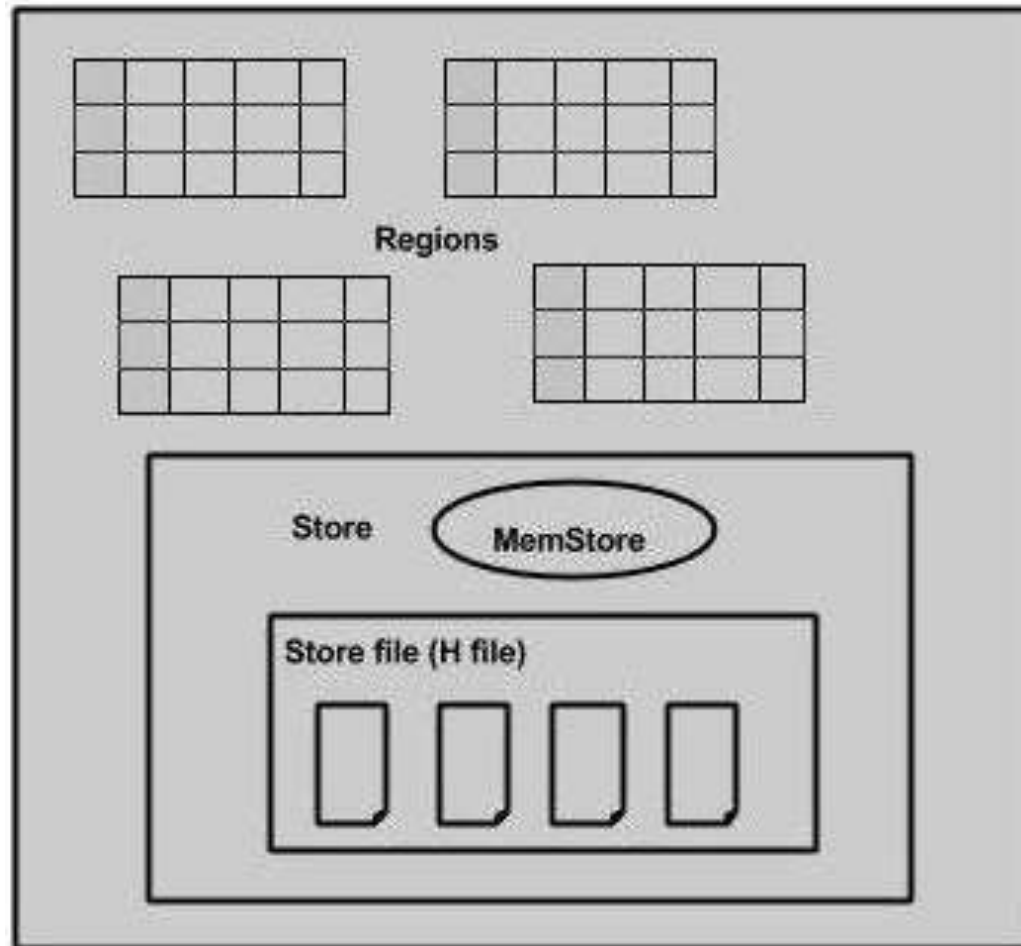


- Zookeeper est un projet open source qui fournit des services tels que le maintien des informations de configuration, le naming, la synchronisation distribuée, etc.
- Zookeeper possède des représentations des différents RegionServers pour aider le MasterServer à découvrir les serveurs disponibles, ainsi qu'à déterminer les serveurs en défaillance.
- Les clients communiquent avec les RegionServers à travers Zookeeper.

# Les Régions Hbase

- Une région est une table qui est fractionnée ou divisée et répartie sur le cluster.
- Le rôle des régions Hbase est le stockage des données.
- Les régions sont verticalement divisées par les familles de colonnes en des stores (magasins, entrepôts,...)
- Les stores sont sauvegardés comme des fichiers HDFS.

# Les Régions Hbase



# Comment fonctionne HBASE?

- Deux types de nœuds:
  - **Master** et **RegionServer**
- **Master** (un par cluster):
  - Gère les opérations du cluster:
    - Affectation des régions au RegionServer, répartition de la charge, fractionnement
    - Sollicite l'aide de Zookeeper
- **RegionServer** (plusieurs par cluster):
  - Héberge les tables, exécute les requêtes(read/write) pour toutes les régions qu'il gère
  - Décide de la taille des régions
  - Peut être ajouté ou supprimé du cluster selon le besoin

# Installation de Hbase

- Hbase s'installe au dessus de Hadoop.
- Après installation, configuration et démarrage de Hbase, accédez à son interface web sur :  
<http://localhost:60010>

Cette interface liste toutes les Region Servers en cours d'exécution, les Master de sauvegarde, ainsi que les tables Hbase.



# Utilisation de Hbase

- Commandes générales Hbase:
  - **status** - fournit le statut de HBase, par exemple le nombre de serveurs.
  - **version** - fournit la version du Hbase utilisé.
  - **table\_help** - fournit de l'aide pour les commandes sur les tables.
  - **whoami** - fournit des informations sur l'utilisateur.

# Utilisation de Hbase

- Commandes opérant sur les tables Hbase (DataDefinition Language):
  - **create** - Créer une table.
  - **list** - Lister toutes les tables dans HBase.
  - **disable** - Désactiver une table.
  - **is\_disabled** - Vérifier si la table est désactivée.
  - **enable** - Activer une table.
  - **is\_enabled** - Vérifier si une table est activée.
  - **describe** - Fournit une description d'une table.
  - **alter** - Modifier une table.
  - **exists** - Vérifier si une table existe.
  - **drop** - Supprimer une table de HBase.
  - **drop\_all** - Supprimer les tables correspondant au 'regex' donné dans la commande.

# Création d'une table

- Avec la commande shell:

create '<table name>','<column family>'

Exemple: `hbase(main):002:0> create 'emp', 'personal data', 'professional data'`

Row key	personal data	professional data

Résultat:

```
0 row(s) in 1.1300 seconds
=> Hbase::Table - emp
```

Vérification : Vous pouvez vérifier si la table a été créée avec la commande **list** :

```
hbase(main):002:0> list
TABLE
emp
2 row(s) in 0.0340 seconds
```

# Utilisation de Hbase

- Commandes opérant sur les tables Hbase:
  - **Java Admin API** - En plus des commandes précédentes, Java fournit une API pour réaliser les fonctionnalités DDL functionalities à travers la programmation.

Au sein du package **org.apache.hadoop.hbase.client**, **HBaseAdmin** et **HTableDescriptor** sont les classes les plus importantes pour fournir les fonctionnalités DDL.

**1- HBaseAdmin** est une classe représentant l'Admin.

En utilisant cette classe, vous pouvez réaliser les tâches d'un administrateur. Vous pouvez récupérer une instance de l'admin en utilisant la méthode **Connection.getAdmin()**.

**2- HTableDescriptor** contient le détail sur une table Hbase tel que:

- La description de toutes les familles de colonnes
- Si la table est en mode lecture seule
- La taille maximum des stores, etc

# Utilisation de Hbase

- **HBaseAdmin :**

- **void createTable(HTableDescriptor desc) :** créer une nouvelle table
- **void createTable(HTableDescriptor desc, byte[][] splitKeys):**  
Créer une nouvelle table avec un ensemble initial de régions vides définies par l'argument splitkeys.
- **void deleteColumn(byte[] tableName, String columnName):**  
Supprimer une colonne d'une table.
- **void deleteColumn(String tableName, String columnName):**  
Supprimer une colonne d'une table.
- **void deleteTable(String tableName):** Supprimer une table.

# Utilisation de Hbase

- **HTableDescriptor:**
  - **HTableDescriptor(TableName name):** Construit une table descriptive en spécifiant le nom de la table objet de la description.
  - **HTableDescriptor addFamily(HColumnDescriptor family) :** Ajoute une famille de colonne au descripteur

# Création d'une table

```
• import java.io.IOException;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.conf.Configuration;

public class CreateTable {

    public static void main(String[] args) throws IOException {

        // Instantiating configuration class
        Configuration con = HBaseConfiguration.create();

        // Instantiating HbaseAdmin class
        HBaseAdmin admin = new HBaseAdmin(con);

        // Instantiating table descriptor class
        HTableDescriptor tableDescriptor = new
        HTableDescriptor(TableName.valueOf("emp"));

        // Adding column families to table descriptor
        tableDescriptor.addFamily(new HColumnDescriptor("personal"));
        tableDescriptor.addFamily(new HColumnDescriptor("professional"));

        // Execute the table through admin
        admin.createTable(tableDescriptor);
        System.out.println(" Table created ");
    }
}
```

Instancier la classe de configuration et la passer au constructeur de HbaseAdmin

Créer les descripteurs de la table et des familles de colonnes

Exécuter la création de la table

# Lister les tables

- Avec la commande shell:

```
hbase(main):002:0> list  
TABLE  
emp  
2 row(s) in 0.0340 seconds
```



# Lister les tables

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.MasterNotRunningException;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class ListTables {

    public static void main(String args[]) throws MasterNotRunningException, IOExcepti

        // Instantiating a configuration class
        Configuration conf = HBaseConfiguration.create();

        // Instantiating HBaseAdmin class
        HBaseAdmin admin = new HBaseAdmin(conf);

        // Getting all the list of tables using HBaseAdmin object
        HTableDescriptor[] tableDescriptor = admin.listTables();

        // printing all the table names.
        for (int i=0; i<tableDescriptor.length;i++ ){
            System.out.println(tableDescriptor[i].getNameAsString());
        }

    }
}
```

Méthode listTables de  
HbaseAdmin

Afficher les noms des tables

# Utilisation de Hbase

- Commandes opérant sur les données (Data Manipulation Language):
  - **put** - Mettre une valeur-cellule dans une colonne spécifique sur une ligne spécifique au sein d'une table spécifique.
  - **get** - Chercher le contenu d'une ligne ou une cellule.
  - **delete** - Effacer une valeur de cellule dans une table.
  - **deleteall** - Effacer toutes les cellules d'une ligne donnée.
  - **scan** - Scanner et retourner les données d'une table.
  - **count** - Compter et retourner le nombre de lignes d'une table.
  - **truncate** - Désactiver, supprimer et recréer une table spécifique.
  - **Java client API** - En plus des commandes précédentes, Java fournit une API pour réaliser les fonctionnalités DML, les opérations **CRUD** (Create Retrieve Update Delete), à travers la programmation, avec le package **org.apache.hadoop.hbase.client** package. **HTable Put** and **Get** sont les classes les plus importantes de ce package.

# Créer une donnée

- Avec une commande shell:

```
put '<table name>','row1','<colfamily:colname>','<value>'
```

**Exemple :** On voudrait créer la table suivante:

COLUMN FAMILIES				
Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

# Créer une donnée

- Insertion de la première ligne

```
hbase(main):005:0> put 'emp','1','personal data:name','raju'  
0 row(s) in 0.6600 seconds  
hbase(main):006:0> put 'emp','1','personal data:city','hyderabad'  
0 row(s) in 0.0410 seconds  
hbase(main):007:0> put 'emp','1','professional  
data:designation','manager'  
0 row(s) in 0.0240 seconds  
hbase(main):007:0> put 'emp','1','professional data:salary','50000'  
0 row(s) in 0.0240 seconds
```

```
hbase(main):022:0> scan 'emp'
```

ROW	COLUMN+CELL
-----	-------------

1	column=personal data:city, timestamp=1417524216501, value=hyderabad
---	---

1	column=personal data:name, timestamp=1417524185058, value=ramu
---	--

1	column=professional data:designation, timestamp=1417524232601,
---	--

	value=manager
--	---------------

1	column=professional data:salary, timestamp=1417524244109, value=50000
---	---

2	column=personal data:city, timestamp=1417524574905, value=chennai
---	---

2	column=personal data:name, timestamp=1417524556125, value=ravi
---	--

2	column=professional data:designation, timestamp=1417524592204,
---	--

	value=sr:engg
--	---------------

2	column=professional data:salary, timestamp=1417524604221, value=30000
---	---

3	column=personal data:city, timestamp=1417524681780, value=delhi
---	---

# Client API

- Classes CRUD (package **org.apache.hadoop.hbase.client**):
  - Htable
  - Put
  - Get
  - Delete
  - Result

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;

public class InsertData{

    public static void main(String[] args) throws IOException {

        // Instantiating Configuration class
        Configuration config = HBaseConfiguration.create();

        // Instantiating HTable class
        HTable hTable = new HTable(config, "emp");

        // Instantiating Put class
        // accepts a row name.
        Put p = new Put(Bytes.toBytes("row1"));

        // adding values using add() method
        // accepts column family name, qualifier/row name ,value
        p.add(Bytes.toBytes("personal"),
            Bytes.toBytes("name"),Bytes.toBytes("raju"));

        p.add(Bytes.toBytes("personal"),
            Bytes.toBytes("city"),Bytes.toBytes("hyderabad"));

        p.add(Bytes.toBytes("professional"),Bytes.toBytes("designation"),
            Bytes.toBytes("manager"));
```

Instancier la classe de configuration et la passer au constructeur de Htable qui prend aussi comme argument le nom de la table où les données seront ajoutées.

Spécifier le nom de la ligne (en string) où les données seront ajoutées

Ajouter les données en spécifiant la famille de la colonne, la colonne et la valeur

# Créer une donnée

```
p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),  
Bytes.toBytes("50000"));
```

Ajouter les données en  
spécifiant la famille de la  
colonne, la colonne et la  
valeur

```
// Saving the put Instance to the HTable.
```

```
hTable.put(p);
```

```
System.out.println("data inserted");
```

Sauvegarder l'ajout avec la  
méthode put

```
// closing HTable
```

```
hTable.close();
```

Enfin fermer l'instance hTable

```
}  
}
```



# Connecter l'application Java à Hbase

1- Créer un fichier de configuration xml:

```
1  <configuration>
2      <property>
3          <name>hbase.zookeeper.quorum</name>
4          <value>localhost</value>
5      </property>
6      <property>
7          <name>hbase.zookeeper.property.clientPort</name>
8          <value>2181</value>
9      </property>
10 </configuration>
```

# Connecter l'application Java à Hbase

2- Créer un objet de configuration en ajoutant les fichiers hbase\_site.xml et hbase-core.xml comme des ressources:

```
Configuration config = HBaseConfiguration.create();

String path = this.getClass()
    .getClassLoader()
    .getResource("hbase-site.xml")
    .getPath();
config.addResource(new Path(path));
```

Ou bien:

```
Configuration config = HBaseConfiguration.create();
config.addResource(new Path("/etc/hbase/conf/hbase-site.xml"));
config.addResource(new Path("/etc/hadoop/conf/core-site.xml"));
```

# Connecter l'application Java à Hbase

3- Créer une connection et récupérer l'objet admin:

```
Connection connection = ConnectionFactory.createConnection(config)
Admin admin = connection.getAdmin();
```

# Travail pratique

- Créer des données avec syncfusion studio shell
- Créer une application java qui se connecte à Hbase et fait des requêtes read/write.