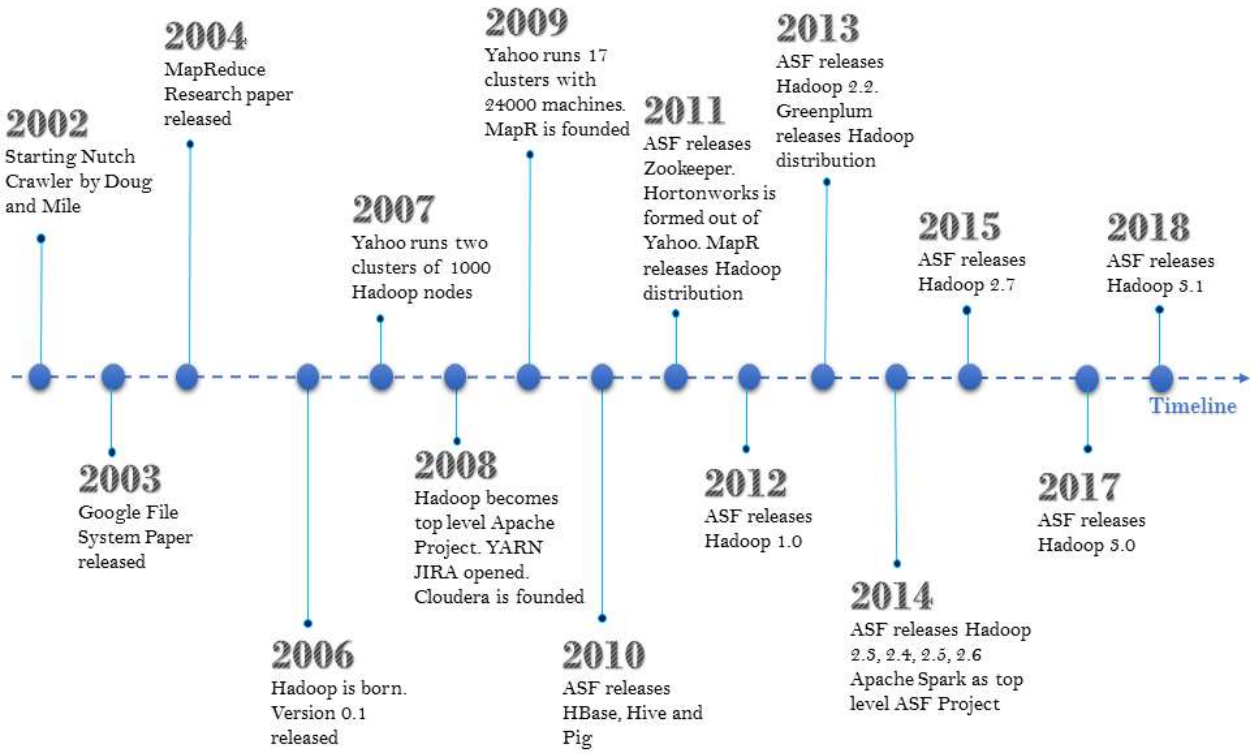

HADOOP

l’écosystème Hadoop

- Hadoop est un système matériel et logiciel distribué capable de répondre aux besoins du Big Data, tant au plan technique qu’économique. Hadoop est capable de stocker et de traiter de manière séquentielle, et à des coûts "raisonnables", des volumes de données de plusieurs péta-octets.
- Permet la scalabilité sur le Commodity hardware
- Fault tolerance
- Optimisé pour une variété de types de données: Texte, images, réseaux sociaux…
- Facilite un environnement partagé

Historique


2002
Starting Nutch Crawler by Doug and Mile
2003
Google File System Paper released
2004
MapReduce Research paper released
2006
Hadoop is born. Version 0.1 released
2007
Yahoo runs two clusters of 1000 Hadoop nodes
2008
Hadoop becomes top level Apache Project. YARN JIRA opened. Cloudera is founded
2009
Yahoo runs 17 clusters with 24000 machines. MapR is founded
2010
ASF releases HBase, Hive and Pig
2011
ASF releases Zookeeper. Hortonworks is formed out of Yahoo. MapR releases Hadoop distribution
2012
ASF releases Hadoop 1.0
2013
ASF releases Hadoop 2.2. Greenplum releases Hadoop distribution
2014
ASF releases Hadoop 2.3, 2.4, 2.5, 2.6 Apache Spark as top level ASF Project
2015
ASF releases Hadoop 2.7
2017
ASF releases Hadoop 3.0
2018
ASF releases Hadoop 3.1
Timeline

Utilisation

Hadoop est particulièrement efficace pour traiter des problèmes présentant une ou plusieurs des caractéristiques suivantes :

- Volume des données à stocker ou à traiter très important.
- Besoin d'effectuer des traitements sur l'ensemble des données (traitements par lots plutôt que transactionnels, donc).
- Données hétérogènes en termes d'origine, de structure et de format (XML, JSON, CSV, texte, binaire…).
- Exécution des tâches d'un job Hadoop en parallèle, sans ordre préétabli.

Hadoop concepts

- Un ensemble de machines fonctionnant avec HDFS, MapReduce et YARN s'appelle un cluster Hadoop
- Un nœud : une machine
- Un cluster peut avoir des milliers de nœuds.
- Un cluster Hadoop doit pouvoir stocker et traiter des volumes de données très importants, dans des délais et à un coût acceptables.

  Si un nœud d'un cluster Hadoop tombe en panne :

  - Cela ne doit jamais entraîner de perte de données.
  - Sa charge de travail doit être répartie automatiquement entre les nœuds restants.
  - S'il est en train d'exécuter une tâche pour un job, la panne ne doit pas affecter le bon déroulement du job.
  - Après qu'un nœud défaillant a été réparé, il doit pouvoir réintégrer le cluster sans qu'il soit besoin de redémarrer ce dernier.
  - L'ajout de nœuds dans un cluster doit se traduire par une amélioration proportionnelle de ses performances.

Tolérance aux pannes

Lors de l'exécution d'un job, Hadoop répartit les tâches entre les nœuds de telle sorte que le nœud qui exécute la tâche héberge aussi les données nécessaires à l'exécution de cette tâche. C'est ce que l'on appelle la "Data Locality" (proximité des données) dans le langage Hadoop. Cette approche est le contraire de celle qui prévaut traditionnellement en informatique.

Le choix d'envoyer les programmes (quelques Mo) vers les données (plusieurs To), permet de limiter le volume des données circulant dans le cluster et d'économiser à la fois de la bande passante et du temps.
Si une tâche d'un job en cours ne se termine pas normalement, Hadoop est capable :

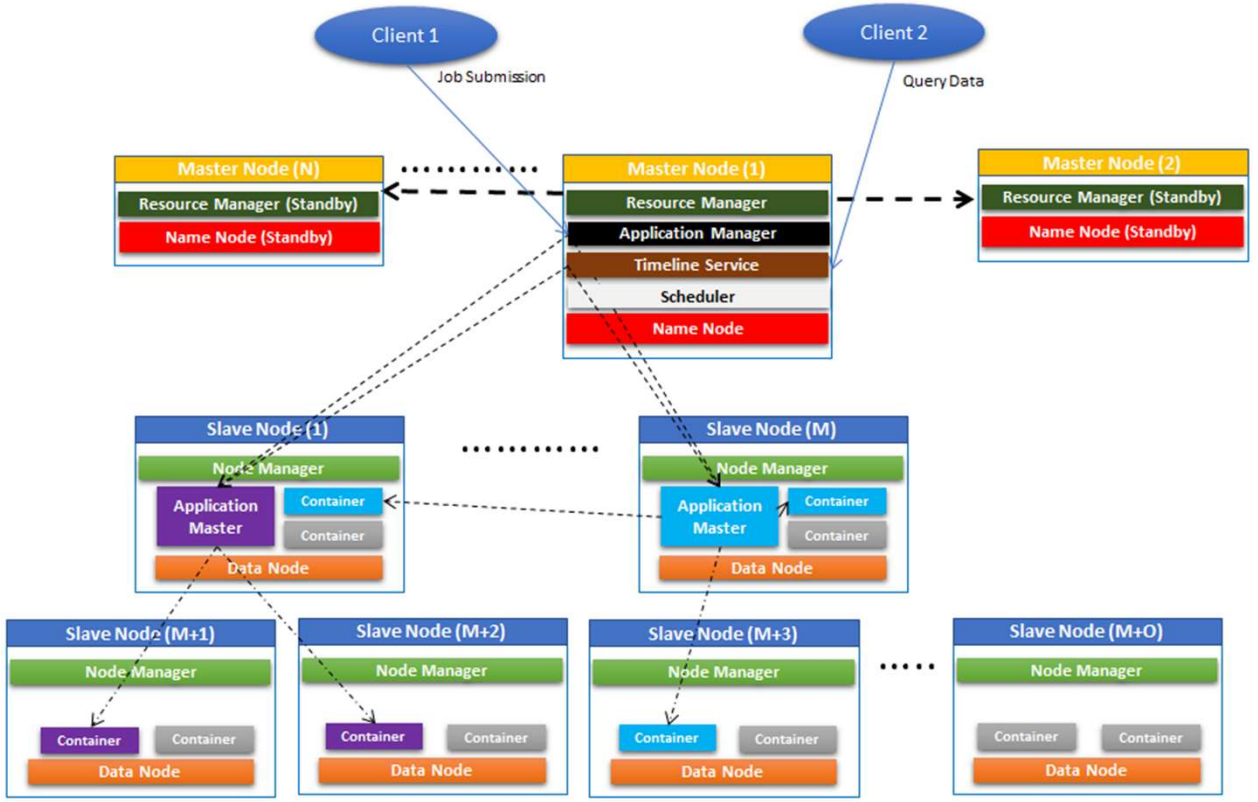
- de détecter l'incident.
- de déterminer avec précision la tâche concernée (code et données).
- de relancer la tâche sur un autre nœud et, si le nœud choisi ne dispose pas des données nécessaires à la bonne exécution de la tâche, d'aller chercher une des deux autres copies des données présentes dans le cluster grâce à la réplication automatique.

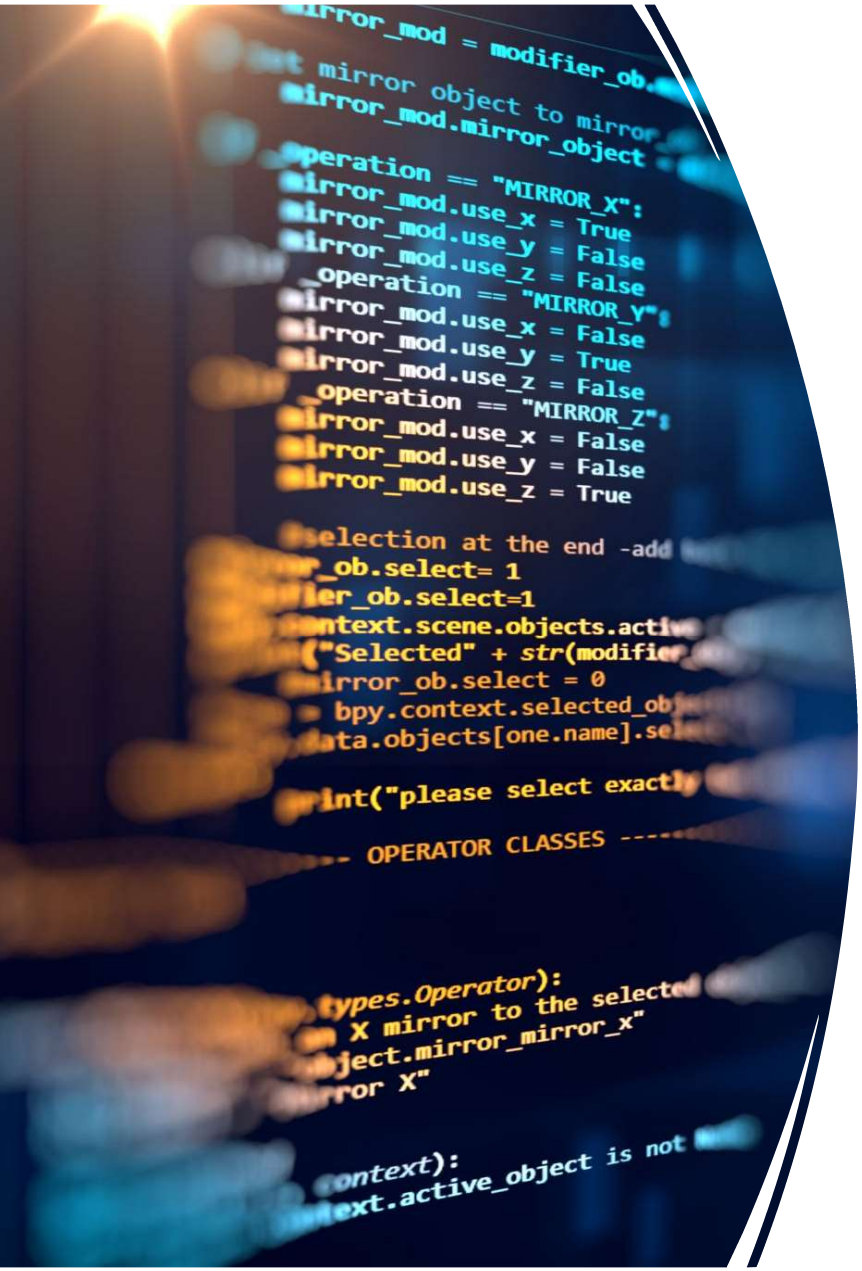
Hadoop dispose de fonctions natives permettant d'ajouter des nœuds à un cluster Hadoop en fonctionnement, sans arrêter ou relancer celui-ci.
L'opération inverse, c'est-à-dire supprimer des nœuds d'un cluster Hadoop en fonctionnement, est possible dans les mêmes conditions.

HDFS

- Inspiré par GFS (Google File System)
- Scalabilité: 200 PB, un cluster de 4500 serveurs, taille d'un bloc par défaut: 64 MB, 128 MB recommandé.
- Tolérance aux pannes par réplication, la réplication assure aussi la localisation des données (replicas par défaut=3)
- La réplication des données répond à deux objectifs :
  - En cas de panne d'un nœud, matérielle ou logicielle, deux copies des données, stockées sur d'autres nœuds, restent disponibles.
  - Lors de l'exécution d'un job Hadoop, chaque tâche peut être exécutée sur n'importe quel nœud, surtout s'il stocke une copie des données nécessaires à la tâche. En conséquence, plus il y a de copies des données et plus il y a de nœuds susceptibles d'exécuter la tâche (c'est-à-dire d'être disponibles) dans des conditions optimales à un moment donné.
- Lecture personnalisée pour différentes types de fichiers:
  - Texte: Par ligne, par mot
  - SIG: par vecteur
  - BIO: FASTA FASTQ
- Deux composants:
  - NameNode pour les méta data généralement 1 par cluster : coordinateur du cluster elle stocke l'arborescence de l'ensemble des fichiers d'un cluster avec des pointeurs vers leurs emplacements.
  - DataNode pour le stockage des blocs généralement 1 par machine

Un cluster Hadoop





MapReduce

MapReduce est un modèle de calcul et une infrastructure logicielle pour l'écriture d'applications qui s'exécutent sur Hadoop. Ces programmes MapReduce sont capables de traiter d'énormes données en parallèle sur de grands clusters.

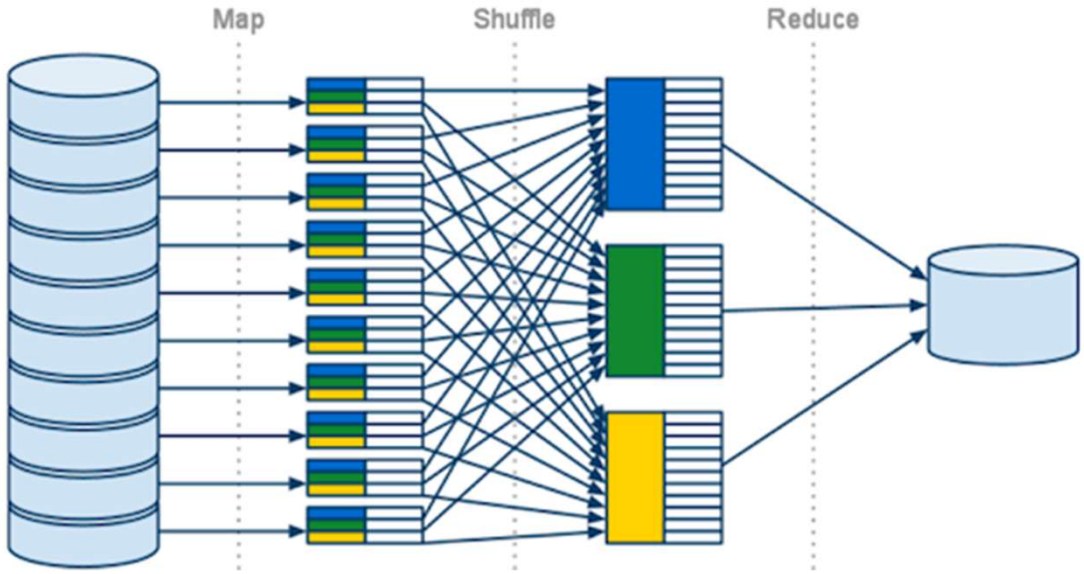
MapReduce

Un programme MapReduce est composé de 3 éléments:

- Le mapper (fonction map)
- Le reducer (fonction reduce)
- Le driver : définit les paramètres d'exécution.

La fonction map applique des opérations sur chaque élément en entrée (L'entrée est fournie comme une collection de paires de clé-valeur) et retourne le résultat comme une collection de clé-valeur

La fonction reduce reçoit la sortie de la fonction map et retourne une liste de clé-valeurs



(Entrée) <k1, v1> -> map -> <k2, v2> -> combiner -> <k2, v2> -> reduce -> <k3, v3> (Sortie)

Exemple WordCount

Mapper

```java
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

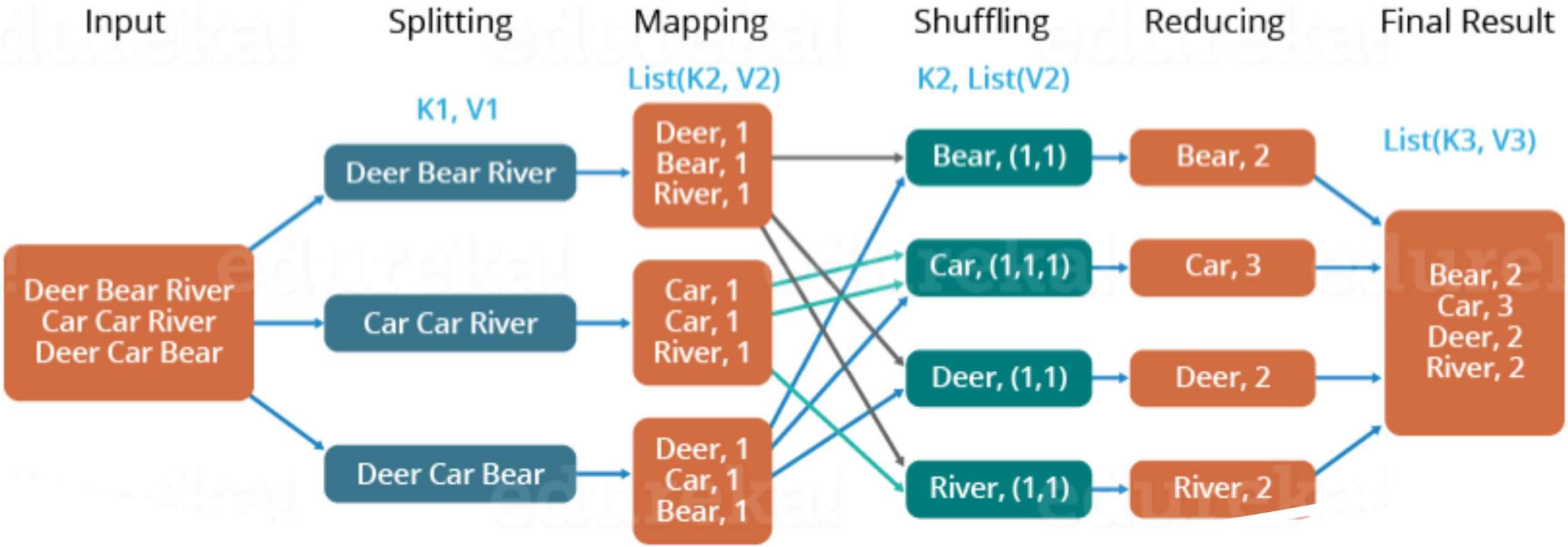
Reducer

```java
public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

Driver

```java
public static void main(String[] args) throws Exception {
    // Lire la configuration par défaut du cluster à partir des fichiers xml
    Configuration conf = new Configuration();
    // Initialiser le job par la configuraion par défaut du cluser
    Job job = Job.getInstance(conf, "word count");
    // Affectation de la classe driver
    job.setJarByClass(WordCount.class);
    // InputFormatClass responsable du parsing du de l'entrée en clé/valeur
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    // OutputFormatClass responsable du parsing du de l'entrée en clé/valeur
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```



Exemple WordCount

- La function map reçoit une ligne à la fois <numéro ligne , ligne> et retourne <mot, 1> pour chaque mot dans la ligne
- Les sorties de map sont triées par mot
- l'instruction (dans le driver) setCombinerClass(IntSumReducer.class) regroupe les paires par mot et retourne <mot,nombre occurrences> pour chaque map
- Le reducer regroupe encore une foils les entrées par mot

Les entrées / sorties

- Dans MapReduce les données sont toujours sous la forme clé-valeur
- Les données en entrée du mapper sont définies dans le driver
  - Leurs localisation (fichier ou répertoire)
  - Type: définit par la classe InputFormat
  - InputSplit: volume de données à lire pendant chaque opération, par défaut c'est la taille d'un bloc HDFS.
- Types de données en entrée (InputFormat)
  - TextInputFormat (par défaut): clé= numéro de ligne , valeur=ligne
  - KeyValueTextInputFormat: chaque ligne est supposée être au format <key><separator><value>\n.
  - SequenceFileInputFormat : Permet de lire un fichier binaire de paires <key, value>.
  - SequenceFileAsTextInputFormat : Format identique au précédent mais, en plus, convertit les clés et les valeurs en strings (<key.toString(), value.toString()>).
- Les clés

L'utilisation de l'interface Writable permet d'optimiser le processus de sérialisation lors des accès disque. Tout type de données dans Hadoop doit implémenter Writable. Par défaut, Hadoop propose les types de données suivants :

- IntWritable (int en Java).
- LongWritable (long en Java).
- loatWritable (float en Java).
- DoubleWritable (double en Java).
- Text (string en Java).

YARN (Resource Manager)

- Permet l’exécution des traitements, Resource manager est le composant principal de YARN et il joue le rôle d’un proxy entre les clients et les nœuds du claster.
- Il existe un Resource manager par cluster, il permet l’accès à tous les nœuds du cluster et à leurs ressources (GPU, CPU, mémoire...).

Hadoop Images

- Cloudera
- HortonWorks
- MAPR

L’écosystème Hadoop

- Principaux composants
  - HDFS
  - MapReduce
  - YARN