**REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE**
**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE**

**UNIVERSITE BADJI MOKHTAR ANNABA**
**FACULTE DES SCIENCES DE L'INGENIORAT**
**DEPARTEMENT INFORMATIQUE**

# HADOOP MAPREDUCE

Master : Gestion et Analyse des Données Massives (GADM)

$2^{eme}$année

**Dr. Klai Sihem**

# Avant propos...

Le Big Data est une science récente qui a surgie avec l'évolution et la variation des données et d'applications mises et échangées en ligne. Cette science consiste à prendre en charge d'une manière efficace un volume important des données hétérogènes en intégrant des techniques et outils nouveaux, vu que la technologie disponible ne répond plus aux besoins.
Ce polycopié est un support pédagogique qui permet d'initier l'étudiant au domaine des Big Data. Ce cours composé de plusieurs chapitres permet aux étudiants de comprendre la problématique et la motivation du domaine, et de maîtriser l'outil Hadoop avec le modèle MapReduce associé à ce domaine.

Chaque chapitre est élaboré pour répondre à un but pédagogique bien précis, se matérialisant par des explications, définitions accompagnées d'exemples et des illustrations par des figures suivies par des exercices, des solutions envisageables ou des fiches de travaux pratiques bien guidés.

1. Le chapitre I met l'étudiant dans le contexte du Big Data, consiste à lui donner des connaissances générales sur le domaine ;
2. Le chapitre II est consacré à l'étude de Hadoop, le framework qui permet le développement d'applications traitant les données massives. Ce chapitre donne les notions les plus générales avec la procédure d'installation du logiciel ;
3. Le chapitre III détaille la partie qui s'occupe du stockage des données "HDFS", avec la possibilité de la manipulation de ces données selon deux manières différentes à savoir : les commandes et l'API JAVA ;

4. Le chapitre IV étudie en détail la partie traitement des données massives "MapReduce", le modèle qui permet de traiter des blocs de données séparément et parallèlement dans des machines connectées. La modélisation selon le paradigme MapReduce est une étape importante avant le développement des programmes ;
5. Le chapitre V détaille l'implémentation des programmes MapReduce dans Hadoop. Dans le cadre de ce chapitre, nous étudions l'implémentation des programmes en utilisant le langage Java. D'autres langages peuvent être utilisés pour écrire des programmes mapreduce, mais cette partie n'est pas traiteé dans ce cours.

L'élaboration de ce polycopié a été inspirée de plusieurs documents, j'ai pris le soin de les citer dans la partie bibliographie, d'autres documents aussi ont été cité afin d'apporter aux lecteurs plus de commandes et plus de détails sur les parties traitées dans ce cours.

# Terminologie. . .

JVM : Java virtuelle machine

HDFS : Hadoop Distributed File System

YARN : Yet Another Ressource Negociator

AM : Application Master

API java : Application Programming Interfaces java

# Table des matières

Préface 3

Table des matières 6

Liste des figures 8

1 Mise en œuvre de MapReduce dans Hadoop : API Java 1

1.1 Structure d'un programme MapReduce . . . . . 2

1.1.1 La classe Mapper . . . . . 2

1.1.2 La classe Reducer . . . . . 4

1.1.3 La classe Driver . . . . . 6

1.1.4 Entre Maper et Reducer : la classe Combiner . . . . . 9

1.2 Les types de données de MapReduce . . . . . 10

1.2.1 Interface Writable . . . . . 10

1.3 Spécification des Entrées dans un programme Mapreduce . . . . 11

1.3.1 Spécification des fichiers d'entrée . . . . . 11

1.3.2 Spécification des formats des données d'entrée . . . . . 11

1.4 Spécification des résultats intermédiaires (les sorties du Mapper, qui sont aussi les entrées du Reducer) . . . . . 12

1.5 Spécification des résultats (données en sortie) . . . . . 12

1.5.1 Spécification des fichiers de sortie . . . . . 12

1.5.2 Une autre possibilité d'affichage des résultats . . . . . 13

1.6 Exécution des programmes Mapreduce . . . . . 14

1.6.1 Les différents packages Hadoop à importer dans le programme Java 14

1.6.2 Exécution d'un programme Mapreduce . . . . . 14

1.7 Exercices . . . . . 15
1.8 Solution des exercices . . . . . 16
1.8.1 Exercice1 . . . . . 16
1.8.2 Exercice2 . . . . . 18
1.8.3 Exercice3 . . . . . 20

A Annexes **23**

Bibliographie **25**

# Liste des figures

# 1 Mise en œuvre de MapReduce dans Hadoop : API Java

**Objectif Pédagogique**

A la fin de ce chapitre, l'étudiant sera capable d'écrire un programme Mapreduce en java et pourra l'exécuter en testant des jeux de données simples en mode local.

**Pré-requis**

L'étudiant doit maîtriser les bases du langage java et le paradigme Mapreduce et HDFS.

Ce chapitre traite la partie la plus technique de la programmation d'un job MapReduce en Java. Ce cours est tiré de 10, 19, voir aussi 13pour la documentation complète et officielle de hadoop Mapreduce.

## 1.1 Structure d'un programme MapReduce

Un programme MapReduce est définie par trois classes au minimum :

### 1.1.1 La classe Mapper

Toute sous classe de la classe Mapper , contient une seule méthode, appelée map qui reçoit une paire (clé-valeur) en paramètre et génère un nombre quelconque de paires (clé-valeurs), voir le code ci-dessous.

```
/** déclaration classe:**/
public class TraitementMapper
extends Mapper<TypCleEM,TypValEM, TypCleI,TypValI>
{
/** appel méthode map:**/
public void map(TypCleEM cleEM, TypValEM valEM,Context context) throws
    Exception
{
/** traitement: CleI = ..., ValI = ... **/
TypCleI CleI = new TypCleI(...);
TypValI ValI = new TypValI(...);
/** Ecriture du résultat: **/
context.write(CleI, ValI);
}}
```

### Explication du squelette de Mapper

1. On commence par déclarer une sous classe de la classe Mapper. Mapper est une classe générique qui se paramétrise avec quatre types :
   — TypCleEM : le type de la clé d'entrée ;
   — TypValEM : le type de la valeur d'entrée ;
   — TypCleI : le type de clé de sortie de Mapper, c'est une sortie intermédiaire ;
   — TypValI : le type de valeur de sortie de Mapper, c'est une sortie intermédiaire.
2. C'est la méthode map qui est implémentée, elle est appliquée sur toutes les paires (cleEM, valeurEM) formées à partir des données d'entrée ;
3. Le troisième argument, Context, permet de renvoyer un couple (clé,valeur) en sortie de la méthode map ;
4. Le traitement de la fonction map est spécifique à chaque problème ;
5. La transmission du résultat se fait à travers le contexte, c'est le message passé de la classe Mapper à la classe Driver ;
6. Il faut évidemment que le type de la clé et la valeur renvoyé correspondent aux types CleI et ValI de la classe Mapper.

### Exemple : la classe Mapper de Word Count 14 (voir psœudo algorithme dans chapitre ??)

```
public class WordMapper extends 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, Context context)
   throws IOException, InterruptedException{
```

```
String Line= value.toString();
StringTokenizer itr= new StringTokenizer(Line);
while(itr.hasMoreTokens()){
        word.set(itr.nextToken());
        context.write(word, one);}
}}
```

### 1.1.2 La classe Reducer

Une sous-classe de Reducer, elle contient également une seule méthode, appelée reduce qui reçoit une liste de paires en paramètre et génère une seule paire.

```
public class TraitementReducer
extends Reducer<TypCleI,TypValI, TypCleSR, TypValSR>
{
public void reduce(TypCleI CleI, Iterable<TypValI> listeI, Context
    context) throws Exception
{
TypCleS cleSR = new TypCleSR();
TypValSR valSR = new TypValSR();
for (TypValI val: listeI)
{
/** traitement: cleSR.set(...), valSR.set(...) **/
}
context.write(CleSR, ValSR);
}}
```

### Explication du squelette de Reducer

1. On commence par déclarer une sous classe de la classe Reducer. Reducer est une classe générique qui se paramétrise avec quatre types :
   - TypCleI : le type de la clé d'entrée du Reducer qui est le même que celui de la sortie du Mapper ;
   - TypValI : le type de la valeur d'entrée du Reducer qui est le même que celui de la sortie du Mapper ;
   - TypCleSR : le type de clé de sortie de Reducer ;
   - TypValSR : le type de valeur de sortie de Reducer.
2. Elle implémente la méthode reduce du programme Mapreduce. La fonction est appelée une fois par clé distincte. C'est l'argument CleI qui contient la clé distincte, et l'argument Iterable<> listeI qui contient la liste des valeurs correspondant à cette clé ;
3. Donc contrairement à la classe Mapper, la classe Reducer recevra ses arguments sous la forme d'une clé unique et d'une liste de valeurs correspondant à cette clé.
4. Elle génère un couple (clé,valeur) en sortie de l'opération reduce par le biais de la méthode write de l'argument context

### Exemple : la classe Reducer de Word Count 14 (voir psœudo algorithme dans le chapitre ??)

```
public class WordReducer extends Reducer<Text, IntWritable, Text,
    IntWritable> {
public void reduce(Text key, Iterable<IntWritable> values, Context
    context) throws IOException, InterruptedException {
int sum=0;
for(IntWritable value:values){
sum+=value.get();}
context.write(key, new IntWritable(sum));}}
```

### 1.1.3 La classe Driver

C'est le programme principal qui informe Hadoop des différents types, classes utilisées et des fichiers d'entrée/de sortie, etc. Il crée un job faisant appel aux deux précédentes classes. La classe Driver doit au minimum avoir :

- Passer à Hadoop des arguments de la ligne de commande, par le biais d'un objet Configuration.
- Informer Hadoop des classes Driver, Mapper et Reducer par le biais d'un objet Job, et des types de clé et de valeur (voir la section1.3) utilisés au sein du programme Mapreduce par le biais du même objet.
- Spécifier à Hadoop l'emplacement des fichiers d'entrée sur HDFS (voir la section1.3) ;
- spécifier également l'emplacement où stocker les fichiers de sortie (voir la section1.3) ;
- Lancer l'exécution de la tâche map/reduce ; recevoir son résultat ;
- L'extrait de code suivant donne le squelette de la classe Driver (les instructions qui permettent de créer et lancer le job MapReduce ;
- L'extrait de code d'aprés donne le squelette de la méthode run.

```
public class Traitement extends Configured implements Tool{
public int run(String[] args) throws Exception{
}
public static void main(String[] args) throws Exception{
if (args.length != 2) System.exit(-1);
Traitement traitement = new Traitement();
System.exit( ToolRunner.run(traitement, args) );
}}
```

```
public int run(String[] args) throws Exception
{Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "traitement");
job.setJarByClass(Traitement.class);
job.setMapperClass(TraitementMapper.class);
job.setReducerClass(TraitementReducer.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
boolean success = job.waitForCompletion(true);
return success ? 0 : 1;}
```

**Explication : le squelette de la classe Driver**

Création de l'objet Configuration :

```
Configuration conf=new Configuration();
```

Création d'un nouveau Job :

```
Job job = Job.getInstance(conf, "traitement");
```

Indication à Hadoop des classes Driver, Mapper et Reducer :

```
job.setJarByClass(Traitement.class);
job.setMapperClass(TraitementMapper.class);
job.setReducerClass(TraitementReducer.class);
```

Spécification de l'emplacement des fichiers d'entrée/de sortie : args[0] désigne le fichier d'entrée et arg[1] désigne le fichier de sortie.

```
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

**Exemple : la classe Driver de Word Count 14(voir psœudo algorithme dans chapitre ??)**

```
public class WordCount {


        public static void main(String[] args) throws Exception{
                // TODO Auto-generated method stub
    if(args.length !=2){
        System.err.println("input path, output path");
        System.exit(-1);
    }
    Job job=new Job();
    job.setJarByClass(WordCount.class);
    job.setJobName("Word Count");
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setMapperClass(WordMapper.class);
    job.setReducerClass(WordReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    System.exit(job.waitForCompletion(true)?0:1);
    }}
```

### 1.1.4 Entre Maper et Reducer : la classe Combiner

Pour une optimisation éventuelle du traitement, on peut étendre la structure d'un programme Mapreduce en intégrant la classe Combiner.
Dans certains cas, le combiner est identique au Reducer. De ce fait, on peut utiliser la même classe pour les deux. La partie de code qui permet de spécifier les classes Mapper, Reducer et Combiner dans la classe Driver est comme suit :

```
job.setMapperClass(TraitementMapper.class);
job.setCombinerClass(TraitementReducer.class);
job.setReducerClass(TraitementReducer.class);
```

Les combiners reprennent la même interface que les reducers sauf que, les paires de sortie doivent être du même type que les paires d'entrée, ci-dessous la structure de la classe Combiner :

```
public class TraitementCombiner
extends Reducer<TypCleI,TypValI, TypCleI,TypValI>
{
@Override
public void reduce(TypCleI CleI, Iterable<TypValI> listeI,
Context context) throws Exception
{
for (TypValI val: listeI) {
/** traitement: cleS = ..., valS = ... **/
}
context.write(new TypCleI(cleS), new TypValI(valS));
}}
```

## 1.2 Les types de données de MapReduce

Les types de données manipulés par les programmes Mapreduce ne sont pas les types standard de Java, mais des types spéciaux permettant de transmettre efficacement des données entre les différents ordinateurs du cluster (voir le tableau1.1 ). Ces

| Type | Description |
|---|---|
| Text | chaîne UTF8 quelconque |
| BooleanWritable | représente un booléen |
| IntWritable | entier 32 bits |
| LongWritable | entier 64 bits |
| FloatWritable | réel IEEE 32 bits |
| DoubleWritable | réel IEEE 64 bits |

Table 1.1 – *Type de données de Mapreduce*

types sont des implémentations d'une interface appelée Writable.

### 1.2.1 Interface Writable

L'interface Writable permet la sérialisation, c'est à dire l'écriture d'une structure de données sous forme d'octets. L'opération inverse, la désérialisation permet de reconstruire une structure de données à partir d'octets. La sérialisation est nécessaire pour échanger des données entre machines.

Cette interface n'est pas limitée à des types simples, car elle peut gérer des collections (tableaux, listes, dictionnaires...) et classes. Elle comprend :

— un constructeur new IntWritable() : pour initialiser la valeur,

**Exemple 1.1** *IntWritable val= new IntWritable(34)*

— un modificateur set() : pour modifier la valeur.

**Exemple 1.2** *val.set(35)*

— un accesseur get() : pour lire la valeur.

**Exemple 1.3** *int v=val.get()*

#### Classes Text

La classe Text permet de représenter n'importe quelle chaîne. Elle possède quelques méthodes à savoir :

— String to String() extrait la chaîne Java ;

— int getLength() retourne la longueur de la chaîne ;

— int charAt(int position) retourne le code UTF8 (appelé point) du caractère présent à cette position.

## 1.3 Spécification des Entrées dans un programme Mapreduce

### 1.3.1 Spécification des fichiers d'entrée

Les instructions suivantes permettent de spécifier le fichier en entrée dans la classe Driver. Il existe deux possibilités :

```
FileInputFormat.addInputPath(job, new Path(args[0]));
FileInputFormat.addInputPath(job, new Path("NOM"));
```

— la première indique quels sont les fichiers HDFS à traiter, ils sont définis comme arguments au lancement du job (première ligne) ;

— la seconde indique les fichiers définis par le chemin. Si le chemin fourni est un dossier, alors tous ses fichiers seront employés et si les fichiers trouvés sont compressés , alors ils seront automatiquement décompressés (deuxième ligne).

### 1.3.2 Spécification des formats des données d'entrée

#### Le format Text avecTextInputFormat.class

Cette instruction spécifie le type des fichiers à lire et implicitement, les clés et les valeurs rencontrées :

```
job.setInputFormatClass(TextInputFormat.class);
```

important : les types des clés et valeurs d'entrée du Mapper doivent coincider avec la classe indiquée pour le fichier. Ici la classe TextInputFormat est une sous-classe de FileInputFormat<LongWritable,Text> . Donc, il faut écrire :

```
public class TraitementMapper
extends Mapper<LongWritable,Text, TypCleI,TypValI>
{
public void map(LongWritable cleEM, Text valEM, ...
```

## 1.4 Spécification des résultats intermédiaires (les sorties du Mapper, qui sont aussi les entrées du Reducer)

Les types des clés et valeurs sortant du mapper et allant au reducer, notés TypCleI et TypValI sont définis par les instructions suivantes :

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(IntWritable.class);
```

Les types définis ci-dessus doivent coincider avec la définition des types des sorties du mapper et des type des entrées du reducer ainsi :

```
class TraitementMapper extends Mapper<..., Text, IntWritable>
class TraitementReducer extends Reducer<Text, IntWritable,...>
```

## 1.5 Spécification des résultats (données en sortie)

### 1.5.1 Spécification des fichiers de sortie

Les résultats du job sont enregistrés dans des fichiers situés dans le dossier indiqué par :

```
FileOutputFormat.setOutputPath(job, new Path("DOSSIER"));
```

Hadoop enregistre un fichier par Reducer final. Leurs noms sont :
part-r-00000, part-r-00001,...

### 1.5.2 Une autre possibilité d'affichage des résultats

Au lieu de récupérer un simple fichier, on peut afficher proprement le résultat final :

```
job.setOutputFormatClass(SequenceFileOutputFormat.class);
if (job.waitForCompletion(true)) {
SequenceFile.Reader.Option fichier =
SequenceFile.Reader.file(new Path(args[1],"part-r-00000"));
SequenceFile.Reader reader =
new SequenceFile.Reader(conf, fichier);
IntWritable annee = new IntWritable();
FloatWritable temperature = new FloatWritable();
while (reader.next(annee, temperature)) {
System.out.println(annee + " : " + temperature);
}
reader.close();
}
```

#### Spécification du format des résultats

Les types des résultats clé(key), valeur(value) sont définis par les instructions suivantes :

```
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(DoubleWritable.class);
```

Les types des résultats définis dans la classe Driver doivent coincider avec les types de sortie de la classe Reducer :

```
class TraitementReducer
extends Reducer<..., Text, DoubleWritable>
```

## 1.6 Exécution des programmes Mapreduce

### 1.6.1 Les différents packages Hadoop à importer dans le programme Java

Pour le bon fonctionnement d'un programme Mapreduce en Java, il faut d'abord importer les packages de Hadoop suivants :

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

### 1.6.2 Exécution d'un programme Mapreduce

— Compresser le programme Mapreduce au sein d'un paquet Java .jar classique, on peut utiliser Eclipse ;

— Exécution des programmes Mapreduce avec la commande suivante : (on utilise la console Cygwin, pour l'exécution des programmes Mapreduce sous Windows)

```
hadoop jar [JAR FILE] [DRIVER CLASS] [PARAMETERS]
```

— Mettre en place les fichiers en entrée, supprimer le dossier de sortie en utilisant la commande HDFS : hdfs dfs -rm -r -f nomrepertoire et spécifier leurs noms et emplacement dans [PARAMETERS] ;
— Afficher le résultat en utilisant la commande HDFS, dans le dossier sortie : hdfs dfs -cat nomrepertoire/part-r-ooooo.

**Exemple 1.4** *Lancement de l'exécution du programme WordCount :*

```
hadoop jar WordCount.jar WordCount in.txt out
```

— *"hadoop jar" est la commande d'exécution ;*
— *"WordCount.jar" est le fichier jar du programme de l'exemple "WordCount" ;*
— *"WordCount" est le nom de la classe Driver du programme ;*
— *"in.txt" est le fichier en entrée à traiter, si le chemin n'est pas spécifié cela signifie que le fichier existe dans le chemin de hadoop/bin où on lance l'exécution ;*
— *"out" est le nom du répertoire qui contiendra les résultats.*

## 1.7 Exercices

Ecrire les programmes Mapreduce en Java correspondants aux algorithmes des exercices du chapitre **??** .

# Annexes

A

# Bibliographie


[01]. site : Le big data, 2019. URL https://www.lebigdata.fr/definition-big-data.

[02]. Stephane Vialle, CentraleSuplec. Big Data : Informatique pour les donnees et calculs massifs. 24 mai 2017 .

[03]. mohammed zuhair al taie. hadoop ecosystem : an integrated environment for big data. in big data, guest posts 2015. URL http://blog.agroknow.com/?p=3810.

[04]. site : Hadoop. 2016. URL http://www.opentuto.com/category/web-2/big-data/hadoop/.

[05]. amal abid. cours big data : Chapitre2 : Hadoop. 2017. URL https://fr.slideshare.net/AmalAbid1/cours-big-data-chap2.

[06]. celine hudelot, regis behmo. realisez des calculs distribue sur des donnees massives. 2019. URL https://openclassrooms.com/fr/courses/4297166-realisez-des-calculs-distribues-sur-des-donnees-massives.

[07]. Benaouda, Sid Ahmed Amine, implantation du modele mapreduce dans l'environnement distribue, 2015 . URL http://dspace.univ-tlemcen.dz.

[08]. Tom White, Hadoop : The Definitive Guide, June 2009 : First Edition.

[09]. Srinath Perera, Thilina Gunarathne, Hadoop MapReduce Cookbook, First published : February 2013.

[10]. Pierre Nerzic, Outils Hadoop pour le BigData, mars 2018. (Cité page 2.)

[11]. marty hall. map reduce 2.0 (input and output). 2013. URL https://www.slideshare.net/martyhall/hadoop-tutorial-mapreduce-part-4-input-and-output.

[12]. build projects, learn skills, get hired. hadoop mapreduce- java-based processing framework for big data. URL https://www.dezyre.com/hadoop-course/mapreduce.

[13]. URL https://hadoop.apache.org/docs/r2.4.1/api/org/apache/hadoop/mapreduce/. (Cité page 2.)

[14]. URL https://gist.github.com/kzk/712029/9d0833aac03b23ec226e034d98f5871d9580724e. (Cité pages 3, 5 et 8.)

[15]. Jeffrey Dean and Sanjay Ghemawat, MapReduce : Simplified Data Processing on Large Clusters, 2004 .

[16]. Univ. Lille1- Licence info 3eme annee, Cours nÂ°1 : Introduction Ã  la programmation fonctionnelle, 2013-2014 .

[17] dataflair team. hadoop architecture in detail hdfs,yarn et mapreduce. 2019. URL https://data-flair.training/blogs/hadoop-architecture/.

[18] vlad korolev. hadoop on windows with eclipse. 2008. URL http://v-lad.org/Tutorials/Hadoop/.

[19]. Donald Miner and Adam Shook. MapReduce Design Patterns. OReilly, 2012. (Cité page 2.)

[20]. Tom White. Hadoop : The Definitive Guide, 4th Edition. OReilly, 2015.

[21]. Hadoop Training. 2018. URL https://www.slideshare.net/AnandMHadoop/session-19-mapreduce.