ENGINEERING SCHOOL
Creating the future together

TP : MapReduce, YARN et API Java

Résumé

Dans ce TP nous allons écrire des programmes Java afin d'extraire des informations synthétiques à partir d'un cluster Hadoop.

L'objectif général est d'écrire des programmes qui implémentent le patron d'architecture MapReduce afin d'extraire des données stockées sur le système de fichiers HDFS d'un cluster. Ce TP présente l'API Java qui permet de réaliser des traitements qui s'exécutent parallèlement sur les nœuds du cluster Hadoop.

1. Introduction

MapReduce est un Framework de traitement de données en clusters créé par Google. Son utilité est de permettre l'écriture de programmes de traitement et de génération de larges ensembles de données sur un cluster de machines. MapReduce est un composant central de Hadoop permettant le traitement résilient et distribué d'ensembles de données massives sur des clusters d'ordinateurs, au sein desquels chaque nœud possède son propre espace de stockage. Concrètement MapReduce propose deux fonctionnalités principales. Il réparti le labeur sur les différents nœuds du cluster (map), puis les organise et réduit les résultats fournis par chaque nœud en une seule réponse qui correspond à la requête (reduce).

Afin de gérer les traitements écrits selon le patron d'architecture MapReduce, Hadoop offre son composant YARN. Ce dernier, permets aux programmeurs de lancer des traitements (jobs) sur des données du système de fichiers HDFS, et de suivre leurs avancements.

2. Exemple de traitement MapReduce

Dans cet exemple nous allons effectuer des traitements afin d'extraire des informations synthétiques sur un fichier de données stocké sur le HDFS. Le fichier de données à utiliser contient des informations sur des ventes. La structure du fichier est la suivante :

Date	Prod	Prix	Paiement	Nom	Ville	Etat	Pays	Compte	DernConn

L'objectif est de trouver le nombre des produits vendus dans chaque pays.

1) Le fichier des données de travail est déjà mis sur le HDFS. Son chemin est le suivant :

```
storage/Sales/SalesJan2009.csv
```

2) Lancer votre IDE et créer un nouveau projet avec le nom « Vente ». Il faut enregistrer ce projet dans le répertoire « TP_Hadoop » de la machine **nodeX**.

3) Dans le projet « Vente » écrire les trois classes dont les codes sont indiqués dans les pages ci-après.

4) Afin de compiler et lancer le projet un script dédié pour la compilation et l'exécution du code Java est déjà mis en place. Vous pouvez le télécharger sur l'url suivant (**Attention ! : il faut le télécharger au sein de votre projet**):

```
$ wget https://azammouri.com/min-5a/Makefile
```

5) Lancer la compilation et l'exécution via la commande :

```
$ make
```

ENGINEERING SCHOOL
Creating the future together

Le Mapper

VenteMapper.java

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class VenteMapper extends Mapper<LongWritable, Text, Text, IntWritable>
{

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

	public void map(LongWritable Key, Text value, Mapper.Context context)
throws IOException, InterruptedException{

		String InputLine = value.toString();

		String[] Splited_InputLine = InputLine.split(",");

		Country.set(Splited_InputLine[7]);
		context.write(Country, one);
	}
}
```

Pour chaque appel de la méthode map() une paire clé-valeur est passée en arguments. Le traitement de la méthode map() commence par diviser le texte qui est reçu comme entrée de la méthode.

Afin de diviser le texte, nous avons utilisé la méthode « split ». Le caractère selon lequel la séparation est faite est : « , » . Par la suite une paire de (clé, valeur) est construite. Cette paire consiste à associer à chaque ville une valeur de « 1 ».

ENGINEERING SCHOOL
Creating the future together

Le Reducer

VenteReducer.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class VenteReducer 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 = Sum + val.get();
		}

		result.set(Sum);

		context.write(key, result);
	}
}
```

Les deux premiers types de données de la fonction reduce() sont « Text » et « IntWritable » qui représentent respectivement les types de la clé et la valeur.

La suite du mapper est de la frome : <Nom_Pays, 1>. Cette sortie du mapper devient une entrée du reducer. D'où l'utilisation des types « Text » et « IntWritable ».

Les deux derniers arguments de la fonction reduce() sont du type « Text » et « IntWritable » respectivement. Ceci correspond à la clé et à la valeur de sortie de la méthode reduce().

ENGINEERING SCHOOL
Creating the future together

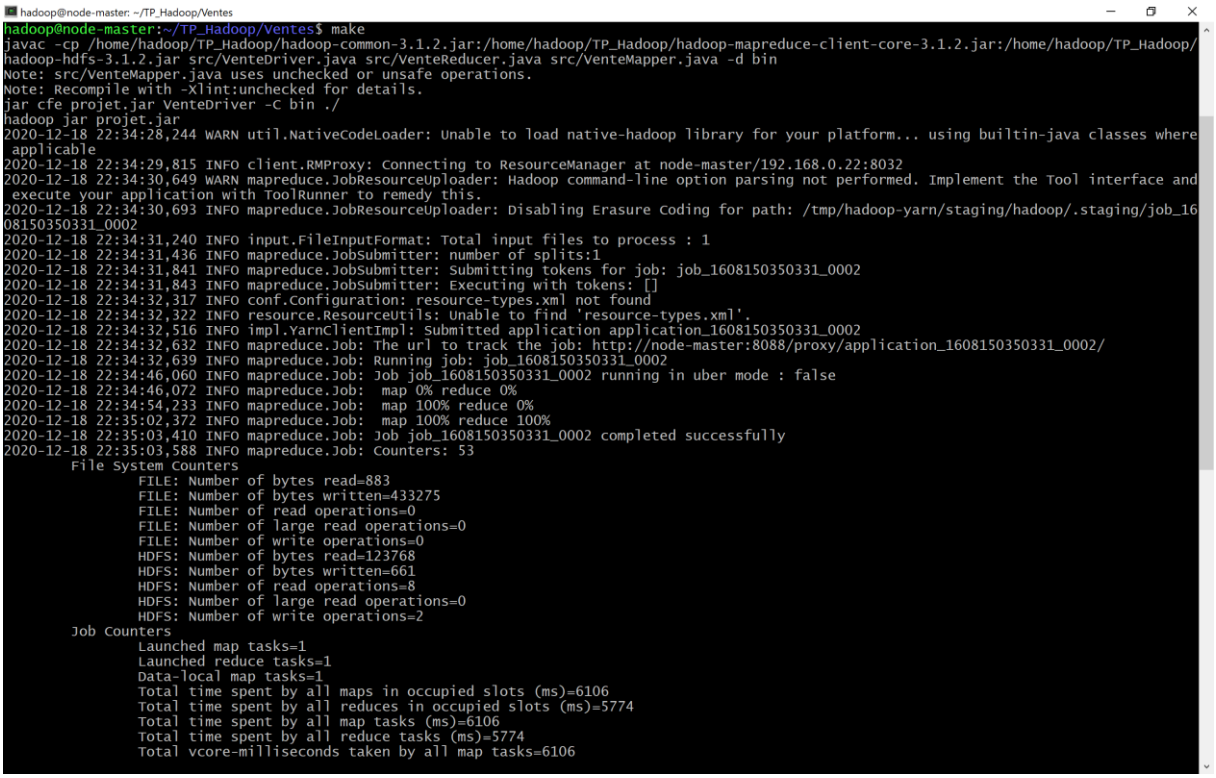
Le Driver

VenteDriver.java

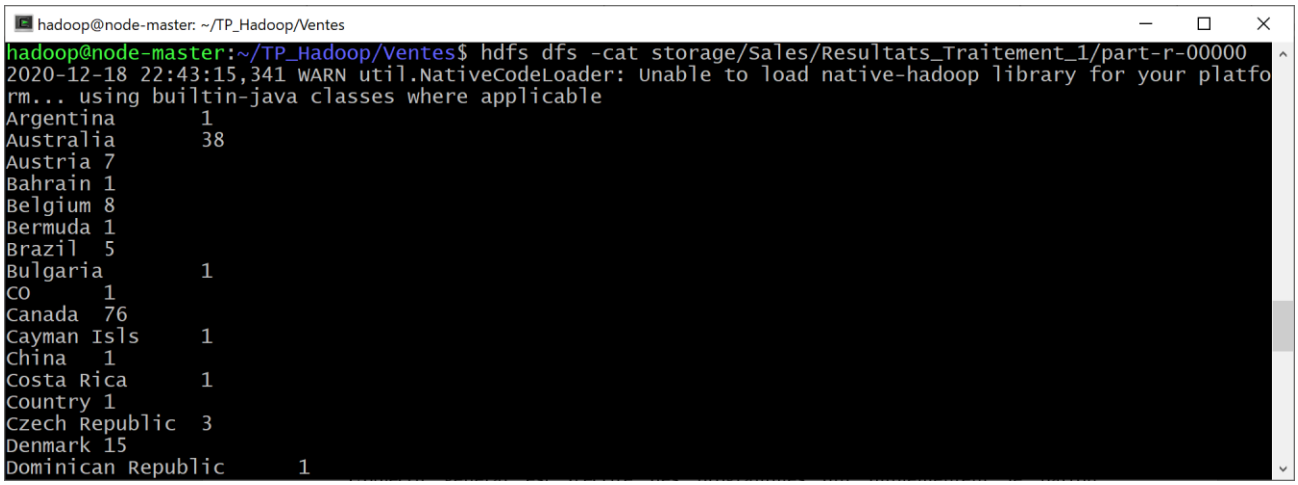
```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class VenteDriver {

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

       Configuration Conf = new Configuration();
       Job Traitement = Job.getInstance(Conf, "Total Ventes par Pays");

       Traitement.setJarByClass(VenteDriver.class);
       Traitement.setMapperClass(VenteMapper.class);
       Traitement.setReducerClass(VenteReducer.class);
       Traitement.setOutputKeyClass(Text.class);
       Traitement.setOutputValueClass(IntWritable.class);

       FileInputFormat.addInputPath(Traitement, new
Path("storage/Sales/SalesJan2009.csv"));

        FileOutputFormat.setOutputPath(Traitement, new
Path("storage/Sales/Resultats_Traitement_1"));

        System.exit(Traitement.waitForCompletion(true) ? 0 : 1);
   }
}
```

ENGINEERING SCHOOL
Creating the future together

Si tout se passe bien, l'exécution de la commande « make » donnera le résultat présenté sur la figure ci-après :



Vous pouvez visualiser le résultat en exécutant la commande ci-après :



3. Travail demandé

Vous allez travailler sur le fichier « `purchases.txt` », déjà existant sur votre cluster (`storage/purchases.txt`). La structure du fichier est donnée ci-après.

Date	Heure	Ville	Achat	Prix	Type de paiement

Objectif 1 :

En utilisant le patron d'architecture MapReduce, affichez le montant total des ventes par ville.

Un compte-rendu de votre travail doit être rendu avant le **13 Novembre 2023**.

Votre compte-rendu doit contenir vos codes avec des commentaires qui permettent de comprendre vos démarches.