TP: HADOOP (MAP-REDUCE)

Alain Tchana, Boris Teabe ENSEEIHT 2016-2017, IRIT/Equipe SEPIA

email: alain.tchana@enseeiht.fr

18 novembre 2016

1 Objectifs

Utilisation de la plateforme *Hadoop* :
— comprendre le concept le modèle de programmation map-reduce ;
— démarrer un job : Word Count ;
— implémenter un job sur des données météorologiques ;
Chaque étudiant dispose d'une machine virtuelle (voir TP1) contenant Hadoop.

2 Hadoop : Généralités

Fonctionne exclusivement sur des paires *<clé, valeur>*. L'entrée (Input) et la sortie (OutPut) d'un job est un ensemble de *<clé, valeur>*. Les classes représentant les clés et les valeurs doivent être sérialisables et implémenter l'interface *Writable*. Voici une illustration de la séquence de traitement d'un Job Hadoop :
(Entrée) <k1, v1> $\Longrightarrow$ *map* $\Longrightarrow$ *<k2, v2>* $\Longrightarrow$ *reduce* $\Longrightarrow$ *<k3, v3> (Sortie)*

2.1 Word Count

Le word count consiste à compter le nombre d'occurences des mots d'un fichier texte.

2.1.1 Comprendre le code

Le programme Word Count est assez simple. Le mapper traite une ligne à la fois. Il split la ligne en *tokens* (séparateur étant l'espace) en utilisant *StringTokenize*, et renvoie un couple clé-valeur *«mot>,1>*. Considérons un fichier contenant cette ligne "Hello world bye world". Le Map aura comme output :
< Hello, 1>
< World, 1>
< Bye, 1>
< World, 1>
Le reducer est implémenté via le méthode reduce. Il somme les occurrences de chaque clé. La sortie pour notre exemple sera :
< Hello, 1>
< World, 2>
< Bye, 1>
Dans la méthode *Main*, certains éléments du Job sont spécifiés, tels que les paths des fichiers d'entrée et de sortie (input/output), le type de clé et de valeur et le format des inputs et des outputs.

2.1.2 Déploiement

Pour ceux qui sont intéressés par la procédure d'installation d'Hadoop, contacter l'enseignant. Nous allons compiler et déployer notre application Word Count sur notre plate-forme d'Hadoop. Nous utiliserons éclipse pour plus de facilité. Il est important que la version de la machine virtuelle JAVA disponible sur votre machine de développement soit la 1.7. La commande "java -version" affiche la version. Si vous n'avez pas la bonne version, signalez le au responsable de TP.

1. Créer un nouveau projet java sous *eclipse* (pour ceux qui utilisent éclipse) ;
2. Télécharger le fichier hadoop-2.2.0.tar.gz (wget https ://archive.apache.org/dist/hadoop/core/hadoop-2.2.0/hadoop-2.2.0.tar.gz) ;
3. Décompresser le fichier téléchargé ;

4. Ajouter les jars : hadoop-common-2.2.0.jar, hadoop-mapreduce-client-common-2.2.0.jar et hadoop-mapreduce-client-core-2.2.0.jar à votre projet éclipse. Ces jars se situent dans les répertoires hadoop-2.2.0/share/hadoop/common et hadoop-2.2.0/share/hadoop/mapreduce du fichier que vous venez de décompresser ;
5. Créer la classe WordCount et copier-coller son contenu ;

```
import java.io.IOException;
import java.util.*;

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;

public class WordCount {

 public static class Map 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 tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "WordCount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.setJar("WordCount.jar");
    job.waitForCompletion(true);
    }
}
```

6. Générer un fichier WordCount.jar pour votre projet (option "export" sous eclipse) ;
7. Connectez-vous à votre machine virtuelle (étudiantxx). Hadoop-2.2.0 y est installé.

8. Copier votre fichier .jar généré dans le répertoire /opt/hadoop-2.2.0/ de votre machine virtuelle en utilisant la commande "scp". La copie se fera ainsi : de votre poste $\Longrightarrow$ cumulus/stratus, ensuite de cumulus/stratus $\Longrightarrow$ votre machine virtuelle.
9. Dans votre machine virtuelle, saisissez la commande : chmod 777 /opt/hadoop-2.2.0/WordCount.jar
10. Saisir les commandes : su - hduser et cd /opt/hadoop-2.2.0/
11. Arrêter le service Hadoop : stop-dfs.sh && stop-yarn.sh (Pour plus d'informations sur les commandes liées à Hadoop, allez sur google ou utilisez l'aide).
12. Formater le nameNode : hdfs namenode -format
13. Démarrer le service Hadoop : start-dfs.sh && start-yarn.sh
14. Saisir la commande "jps" pour s'assurer que le dataNode, le namenode, le RessourceManager et le NodeManager sont bien démarrés. Si tous ces élements n'ont pas démarrés pas la peine d'aller plus loin, et signalez à l'enseignant.
15. Créer un fichier *count* et le remplir avec des mots de votre choix ;
16. Supposons que */user/root/count* sera votre fichier d'entrer dans le système de fichier HDFS et */user/root/ouput* votre fichier de sortie ;
17. Pour Ajouter votre fichier count dans le système de fichiers HDFS, faites :
    — hdfs dfs -mkdir /user
    — hdfs dfs -mkdir /user/root
    — hdfs dfs -put count /user/root/
    — hdfs dfs -ls /user/root/
18. Démarrer votre programme wordCount : hadoop jar WordCount.jar WordCount /user/root/count /user/root/output ;
19. Afficher le résultat de votre job : hdfs dfs -cat /user/root/output/part-r-00000

3 Traitement des données météorologiques

Dans le répertoire "/home/hduser", se trouve des données météorologiques sur plusieurs années de la planète crypton. Il est question d'implémenter deux jobs map-reduce permettant :

— de déterminer la température la plus haute pour chaque année ;
— et de déterminer le nombre de mois ayant eu une température supérieure à une valeur donnée (un paramètre).

Inspirez-vous du word Count et de ce qu'on vient de vous présentez pour réaliser ce travail. Quelques indications :

1. utiliser les types d'Hadoop (qui permettent à Hadoop d'optimiser le transfert des données sur le réseau) à la place des types standards de Java. Exemple utiliser Text au lieu de String, IntWritable au lieu de Integer ;
2. pour les calculs, préférer les types Java aux types Hadoop ;
3. le fichier des données météorologiques se trouve dans le répertoire "/home/hduser/" de votre machine virtuelle, et se nomme "data" ;
4. La structure d'une ligne du fichier "data" est : "jour :mois :année :température :ville".

Bonne chance.