Rapport TP2

Traitement par Lot et Streaming avec Spark

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Préparée par : encadré par :

* Am-mara khaoula
* Selma Houraji Mr.Badir

Objectifs de TP :

* Comprendre les fondamentaux de Spark .
* Effectuer des traitements par lot avec Spark.
* Effectuer des traitements en streaming avec Spark.

Installation de Spark sur le cluster Hadoop :

**Définition**:

Spark est une plateforme de traitement distribué et de calcul en mémoire rapide conçue pour gérer de gros volumes de données et des tâches complexes. Elle est principalement utilisée pour le traitement et l'analyse de données à grande échelle.

 Il peut s'exécuter sur plusieurs plateformes: Hadoop, Mesos, en standalone ou sur le cloud,et dans ce TP, nous allons exécuter Spark sur Hadoop YARN.

**Installation:**

Puisqu’on a déjà installé Spark sur le cluster Hadoop utilisé dans le TP1 en suivant les étapes de la partie "Installation" du TP1, nous pouvons simplement démarrer nos machines en utilisant les commandes suivantes :

1. On Démarre les conteneurs du cluster Hadoop en exécutant la commande :

docker start hadoop-master hadoop-slave1 hadoop-slave2

1. Accédez au conteneur principal (hadoop-master) en exécutant la commande :

docker exec -it hadoop-master bash

1. À l'intérieur du conteneur hadoop-master, lancez les démons YARN et HDFS en exécutant la commande :

./start-hadoop.sh

Maintenant on peut verifier que tous les démons sont démarrés avec succès en tapant la commande jps à l'intérieur du conteneur hadoop-master.

Pour les nœuds esclaves (hadoop-slave1 et hadoop-slave2), on peut y accéder à partir de notre machine hôte en exécutant la commande docker exec -it hadoop-slave1 bash (ou hadoop-slave2 pour le deuxième nœud esclave). À l'intérieur du conteneur, on peut également vérifier les processus en tapant jps

Une fois que tous les démons sont lancés avec succès sur tous les nœuds du cluster, nous pouvons commencer à utiliser Spark pour le traitement distribué de vos données.

Test de Spark avec Spark-Shell :

Pour tester l'exécution de Spark avec Spark-Shell, on va suivre les étapes suivantes :

1. Créer un fichier file1.txt sur votre nœud maître contenant le texte suivant :

Hello Spark Wordcount!

Hello Hadoop Also :)

1. Chargez le fichier dans HDFS en utilisant la commande :

hadoop fs -put file1.txt

1. Pour vérifier que Spark est bien installé, on lance la commande suivante :

spark-shell

1. Pour tester Spark avec un code Scala simple, on exécute les lignes suivantes une par une dans le shell Spark :

val lines = sc.textFile("file1.txt")

val words = lines.flatMap(\_.split("\\s+"))

val wc = words.map(w => (w, 1)).reduceByKey(\_ + \_)

wc.saveAsTextFile("file1.count")

1. Pour afficher le résultat, on va quitter le shell Spark. Ensuite, on va télécharger le répertoire file1.count créé dans HDFS en utilisant la commande :

hadoop fs -get file1.count

Spark Batch en Java:

### **Environnement et Code**

Dans cette partie, nous allons créer un projet Spark Batch en Java pour effectuer un simple WordCount, le charger sur le cluster et lancer le job.

1. Pour commencer, nous allons créer un projet Maven avec IntelliJ IDEA en utilisant les configurations suivantes dans le fichier pom.xml :

<groupId>spark.batch</groupId>

<artifactId>wordcount</artifactId>

<version>1</version>

1. On ajoute également les dépendances nécessaires dans le fichier pom.xml et spécifiez la version du compilateur Java :

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.11</artifactId>

<version>2.1.0</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.22</version>

</dependency>

</dependencies>

1. On creee un package dans le répertoire "java" avec le nom "tn.insat.tp21", puis une classe appelée "WordCountTask" à l'intérieur de ce package. Voici le code à insérer dans la classe WordCountTask :

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import scala.Tuple2;

import java.util.Arrays;

import static jersey.repackaged.com.google.common.base.Preconditions.checkArgument;

public class WordCountTask {

private static final Logger LOGGER = LoggerFactory.getLogger(WordCountTask.class);

public static void main(String[] args) {

checkArgument(args.length > 1, "Please provide the path of input file and output dir as parameters.");

new WordCountTask().run(args[0], args[1]);

}

public void run(String inputFilePath, String outputDir) {

String master = "local[\*]";

SparkConf conf = new SparkConf()

.setAppName(WordCountTask.class.getName())

.setMaster(master);

JavaSparkContext sc = new JavaSparkContext(conf);

JavaRDD<String> textFile = sc.textFile(inputFilePath);

JavaPairRDD<String, Integer> counts = textFile

.flatMap(s -> Arrays.asList(s.split(" ")).iterator())

.mapToPair(word -> new Tuple2<>(word, 1))

.reduceByKey((a, b) -> a + b);

counts.saveAsTextFile(outputDir);

}

}

### **Test du code en Local :**

Pour tester le code en local, on va suivre les étapes suivantes :

1. On Insére un fichier texte de notre choix dans le répertoire src/main/resources. Par exemple, on peut utiliser le fichier loremipsum.txt.
2. Nous créons une nouvelle configuration de type "Application" dans IntelliJ IDEA en allant dans Run -> Edit Configurations. On nomme la configuration WordCountTask et définisse les arguments suivants (chemin du fichier d'entrée et répertoire de sortie) dans le champ "Program arguments":

src/main/resources/loremipsum.txt src/main/resources/out

1. On Clique sur OK et on lance la configuration.

### **Lancement du code sur le cluster :**

Pour lancer le code sur le cluster, on va suiver les étapes suivantes :

1. Modifiez la classe WordCountTask .
2. On Lance une configuration de type Maven avec les commandes package et install. Cela créera un fichier nommé wordcount-1.jar dans le répertoire target.
3. Ensuite on copie le fichier JAR généré dans le conteneur Docker en exécutant la commande suivante depuis le répertoire du projet dans votre terminal:

docker cp target/wordcount-1.jar hadoop-master:/root/wordcount-1.jar

1. Revenons à notre conteneur hadoop-master et lancons un job Spark en utilisant le fichier JAR généré à l'aide de la commande spark-submit, qui est un script utilisé pour exécuter des applications Spark sur un cluster.

spark-submit --class tn.insat.tp21.WordCountTask

--master local

--driver-memory 4g --executor-memory 2g --executor-cores 1

wordcount-1.jar

input/purchases.txt

output

Spark Streaming :

Spark Streaming est un module de Spark qui permet de traiter des données en streaming. Il peut lire les données à partir de diverses sources telles que Kafka, Flume, Kinesis ou des sockets TCP, et les traiter à l'aide d'algorithmes complexes. Les données traitées peuvent ensuite être stockées dans des systèmes de fichiers, des bases de données ou des tableaux de bord. Il est même possible d'effectuer des opérations de machine learning et de traitement de graphes sur les flux de données.

Le fonctionnement interne de Spark Streaming est le suivant : il reçoit les données en streaming et les divise en micro-batchs, qui sont ensuite traités par le moteur Spark pour générer le flux final de résultats.

### **Environnement et Code :**

Nous allons tester le streaming en local, comme d'habitude. Pour cela:

1. On va créer un nouveau projet Maven, avec le fichier pom suivant:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>spark.streaming</groupId>

<artifactId>stream</artifactId>

<version>1</version>

<dependencies>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.11</artifactId>

<version>2.2.1</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_2.11</artifactId>

<version>2.2.1</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

1. On va Créer une classe tn.insat.tp22.Stream avec le code suivant:

import org.apache.spark.SparkConf;

import org.apache.spark.streaming.Durations;

import org.apache.spark.streaming.api.java.JavaDStream;

import org.apache.spark.streaming.api.java.JavaPairDStream;

import org.apache.spark.streaming.api.java.JavaReceiverInputDStream;

import org.apache.spark.streaming.api.java.JavaStreamingContext;

import scala.Tuple2;

import java.util.Arrays;

public class Stream {

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

SparkConf conf = new SparkConf()

.setAppName("NetworkWordCount")

.setMaster("local[\*]");

JavaStreamingContext jssc =

new JavaStreamingContext(conf, Durations.seconds(1));

JavaReceiverInputDStream<String> lines =

jssc.socketTextStream("localhost", 9999);

JavaDStream<String> words =

lines.flatMap(x -> Arrays.asList(x.split(" ")).iterator());

JavaPairDStream<String, Integer> pairs =

words.mapToPair(s -> new Tuple2<>(s, 1));

JavaPairDStream<String, Integer> wordCounts =

pairs.reduceByKey((i1, i2) -> i1 + i2);

wordCounts.print();

jssc.start();

jssc.awaitTermination();

}

}

### **Test du code en Local :**

1. On exécute la classe Stream et ouvre un terminal supplémentaire.
2. On Utilise la commande nc -lk 9999 dans le terminal supplémentaire pour créer un flux de données en continu.
3. Toutes les secondes, l'application Stream comptera le nombre de mots dans le flux de données et affichera les résultats dans la console.

### **Lancement du code sur le cluster :**

Pour exécuter le code sur un cluster, nous devons apporter les modifications suivantes :

1. Modifier la classe Stream pour spécifier l'adresse IP de votre machine locale dans la méthode socketTextStream.
2. Générer un fichier JAR en exécutant la commande mvn package install.
3. Copier le fichier JAR sur le conteneur Hadoop du cluster.
4. Utiliser la commande spark-submit pour soumettre le job Spark Streaming au cluster .