Chap II: Spark (pyspark)

Spark I

Tables des matières

Spark jour 1

Introduction

Introduction

Spark est un Framework open source de calcul distribué conçu pour effectuer des traitements Big Data. Il contient des modules de traitement du Machine Learning, du Streaming, des Graphs et du SQL.

Développé en 2009 en Californie à l'université de Berkeley par Matei Zaharia

Spark passe en open source en 2010 puis en 2013 Spark rejoint la fondation

Pourquoi Spark

- Faire du calcul distribué sur plusieurs machine
- Framework pour coordonner l’exécution des tâches sur toutes les machines du cluster.
- Un framework facile à prendre en main et pour un nombre important de cas d’usage
- Palier aux limites de Hadoop

Qui utilisent Spark ?

Introduction

Spark est un framework complet et unifié

- Charger de gros volumes de données provenant de sources variées
- Interroger, explorer et visualiser les données
- Exécuter des requêtes SQL
- Analyser les données en temps réel
- Concevoir des modèles de Machine Learning.
- Manipuler les Graphs
- Manager et coordonner l'exécution des traitements sur l'ensemble des nœuds du cluster

Remarque:
Spark n'est pas
une base de données
un cluster
un langage de programmation

Introduction

Apache Spark est codé en Scala

Les APIs de Spark sont utilisables en :
Scala, Python, Java, R et SQL
API Python de Spark s'appelle PySpark

La documentation relative à Pyspark se trouve ici:
https://spark.apache.org/docs/latest/api/python/reference/index.html

Exemple de code en python et en scala:

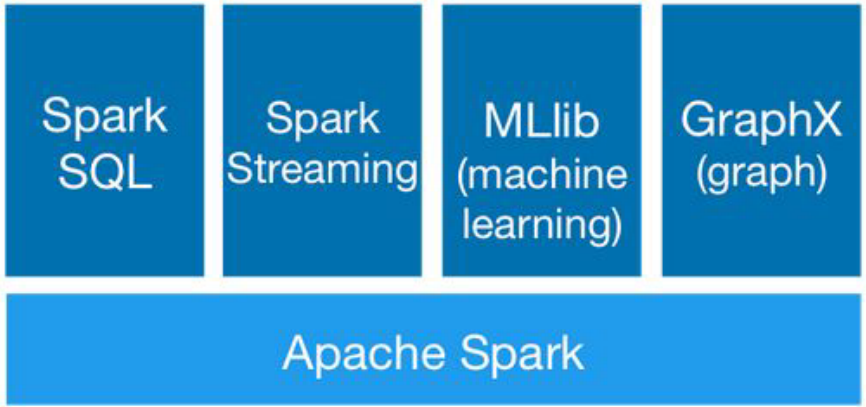
```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```scala
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Les modules Spark

Le Framework Spark comporte 4 modules

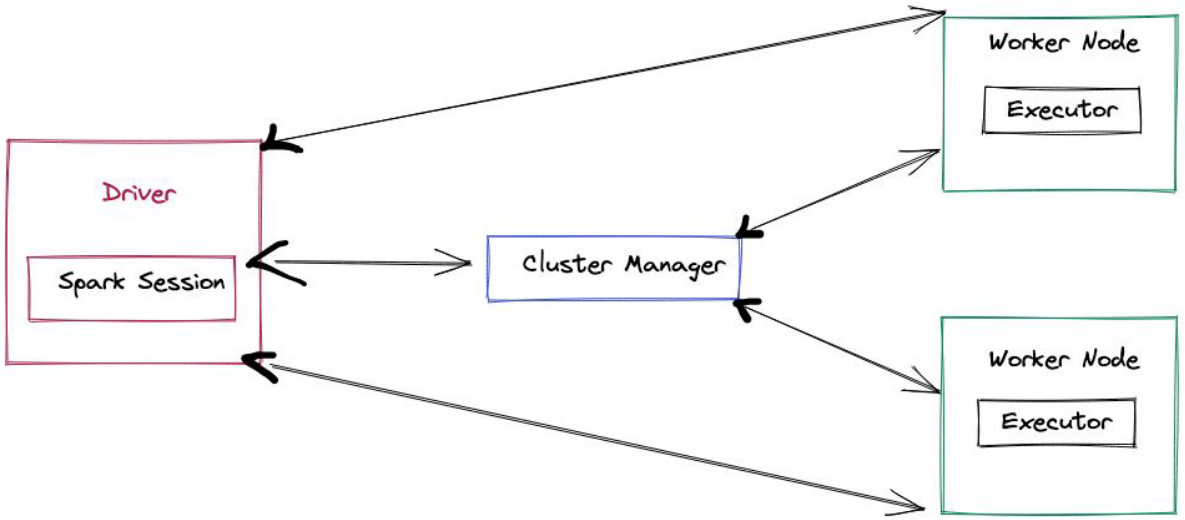
- Spark SQL : Pour manipuler des données volumineuses et structurées
- Spark Streaming : Pour traiter les données en flux continu
- Spark ML : Pour faire du Machine Learning
- Spark GraphX : Manipuler des Graph



L’architecture de Spark

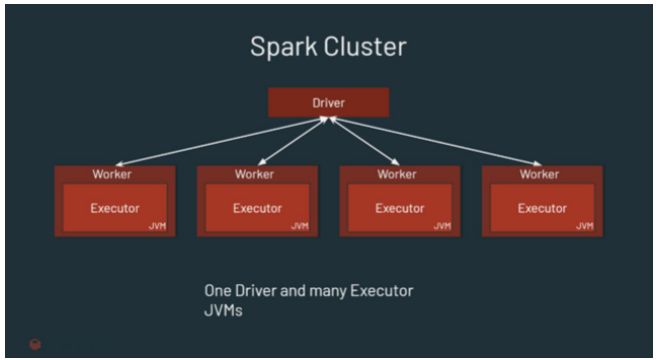
Architecture

Trois composants principales



Architecture

Spark fonctionne sur le modèle « Maître/Esclave » : un Driver + plusieurs Worker



Le Driver

- Optimise et traduit le code en jobs spark
- Orchestre et monitore l’exécution des jobs
- Stocke les métadonnées sur les Rdds/Dataframes et les partitions associées

Le Cluster Manager

- Alloue les ressources nécessaires à l’exécution des différents jobs
- Gère la disponibilité ou non des executors

Les Executors

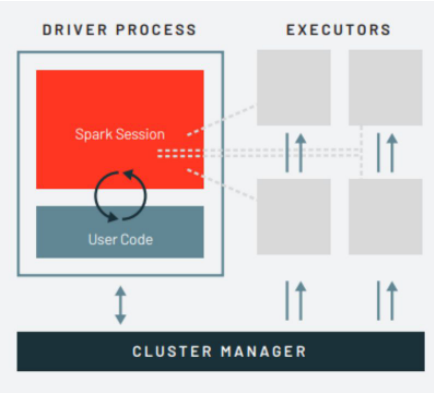
- Exécuter les différents jobs soumis par le driver
- Retourne les résultats au driver
- Interagit avec les systèmes de stockage (Lectures/Écritures)
- Ecrire les résultats intermédiaires en mémoire ou sur disque

Workflow

Lors de l'exécution du code

1. Le **Driver** demande des ressources au **Cluster Manager**
2. Le **Cluster Manager** alloue des **Executors**
3. Le **Driver** optimise le code
4. Le **Driver** envoie les tâches aux **Executors**
5. Les **Executors** traitent les tâches et retournent le résultat au **Driver**

SparkSession est le point d'entrée unique qui permet d'accéder à toutes les fonctionnalités de Spark
Avant, il y avait plusieurs points d'entrée : Spark context, Hive Context, SQL context
Depuis la version 2.0, SparkSession les unifie tous.



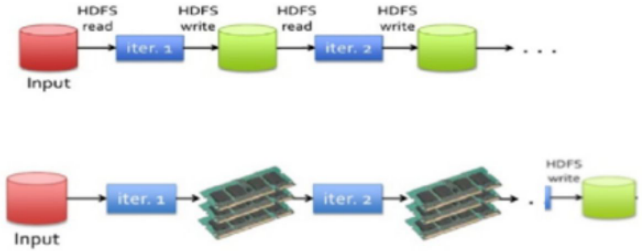
Comparaison avec Hadoop

Hadoop vs Spark

La plus grande différence, entre Hadoop Map Reduce et Spark

Hadoop Map Reduce utilise le disque pour stocker ses résultats intermédiaires
!!! 90% du temps à faire des opérations de lecture-écriture sur HDFS !!!

Spark a la possibilité d'exécuter les traitements en mémoire (RAM)
Avec Spark, les résultats intermédiaires sont sauvegardés dans la RAM
Spark est jusqu'à 100 fois plus rapide que Hadoop



Hadoop vs Spark

Hadoop vs Spark

Exemple de code sur Hadoop et sur Spark

```python
import sys

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    # create Spark context with necessary configuration
    sc = SparkContext("local","PySpark Word Count Exmaple")

    # read data from text file and split each line into words
    words = sc.textFile("D:/workspace/spark/input.txt").flatMap(lambda line: line.split(" "))

    # count the occurrence of each word
    wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)

    # save the counts to output
    wordCounts.saveAsTextFile("D:/workspace/spark/output/")
```

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

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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  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);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    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);
  }
}
```

Spark I

Les RDDs (Resilient Distributed Dataset)

RDD

RDD = Resilient Distributed Datasets

Resilient : Tolérant à la panne

Distributed : Le Dataset est distribué à travers les nœuds du cluster

Dataset : Ensemble de données stockées sur un stockage permanent (HDFS, Cassandra; etc.), en cache (RAM, RAM + disque, etc.) ou sur un autre RDD

Représente une collection partitionnée, immuable, résiliente et distribuée d'objets qui peuvent être manipulés en parallèle.

C'est l'abstraction de base de la donnée dans Spark.

RDD

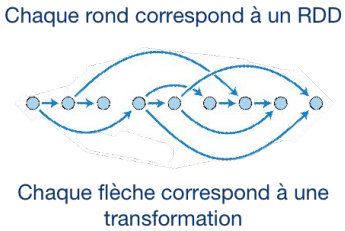
Les partitions sont distribuées à travers les workers du cluster
Chacune des partitions sera à la charge d'un executor lors des traitements.

Spark décide du nombre de partitions mais on peut modifier ce paramètre
Plus de partitions = plus de parallélisme,
Dans l'idéal, il faudrait que le nombre de partitions soit un multiple du nombre de cœurs du cluster.

Chaque RDD stocke le lineage graph (liste de dépendances avec les RDD parents)
avec la donnée initiale on est donc capable de recréer chaque RDD

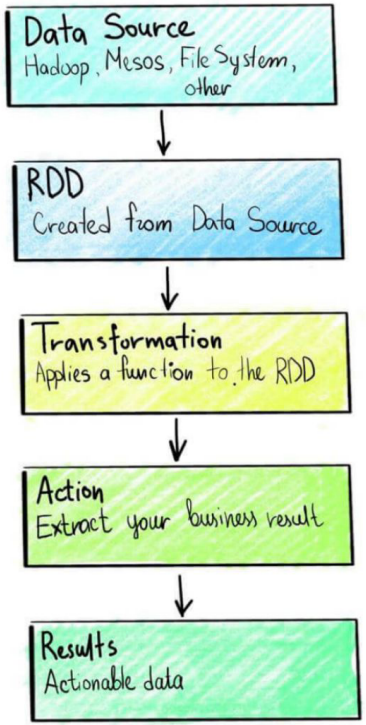
Définition DAG

Il s’agit d’une suite logique d’actions que l’on souhaite appliquer sur les données de manière parallélisée, cette suite est un graphe orienté acyclique, il est automatiquement créé par Spark.



A ne pas confondre avec le RDD lineage graph, le DAG comprend l’ensemble des RDD et des transformations.

RDD workflow



Spark I

RDDs:
Les opérations – Transformations et Actions

Opérations

Il existe 2 types d'opérations sur les RDD : les Transformations et les Actions

Une t**ransformation** est une opération qui transforme la donnée et qui retourne un nouveau RDD.

Une **action** est une opération qui retourne une valeur ou qui persiste la donnée ou encore qui affiche un résultat (ex: écriture dans le datalake, print, etc).

Un RDD est une collection immuable, c'est-à-dire qu'on ne peut pas le modifier.

Chaque transformation crée un nouveau RDD.

RDD: Transformations

map(func)	Retourne un nouveau RDD obtenu en appliquant la fonction func à chaque item du RDD de départ.
filter(func)	Retourne un nouveau RDD obtenu en filtrant sur les items pour lesquels la fonction func retourne True.
flatMap(func)	Similaire à map mais qui supprime l'imbrication des listes
union(otherDataset)	Retourne un RDD qui est l'union des items du RDD source et du RDD argument (otherDataset).
distinct()	Retourne un RDD qui est obtenu du RDD source en éliminant les doublons des items.

RDD: Actions

reduce(func)	Agréger les éléments du RDD en utilisant la fonction func (qui prend 2 arguments et retourne 1 résultat).
count()	Retourner le nombre d'items du RDD.
take(n)	Retourner les n premiers items du RDD.
first()	Retourner le premier item du RDD (similaire à take(1)).
countByKey()	Pour un RDD de type (clé, valeur), retourne l'ensemble de paires (clé, int) avec le nombre de valeurs pour chaque clé.
collect()	Retourner tous les items du RDD au programme driver. A utiliser seulement si le RDD a un volume faible (par ex, après des opérations de type filter très sélectives).
saveAsTextFile(path)	Écrit les items du RDD dans un fichier texte dans un répertoire du système de fichiers local, HDFS ou autre fichier supporté par Hadoop.

RDD: Exemples

On peut créer un RDD

- Avec sc.parallelize() sur une collection (listes, tuples, dictionnaires ...)
- Avec sc.textFile() sur un fichier existant, situé dans un HDFS en local ou dans une BDD : JDBC, Cassandra, HBase…
- En appliquant une transformation sur un autre RDD (avec filter, map…)

```
# Création d'un RDD à partir d'une collection :
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)
```

```
# Création d'un RDD à partir d'un fichier:
rdd = sc.textFile("data.txt")
```

```
# Création d'un RDD à partir d'un autre RDD :
new_rdd = rdd.map(row : row.lower())
```

sc est équivalent à SparkSession.SparkContext point d'entrée utilisée pour créer des RDD

RDD: Exemples

```
// Création d'un RDD à partir d'un fichier (transformation) :
sample_RDD = sc.textFile("sample.txt")

// Compte tous les éléments (action)
sample_RDD.count()

// Uppercase de tous les éléments (transformation) :
new_RDD = sample_RDD.map(lambda row : row.upper())

// Rapatrie et affiche tous les éléments (action) :
new_RDD.collect()
```

Notions importantes:

Lazy Evaluation & Caching

RDD: Lazy Evaluation

Lazy Evaluation : Une transformation d'un RDD n'est pas exécutée tant qu'il n'y a pas d'action !

- rdd = sc.textFile("spam.txt") Rien n'est exécuté à cette ligne
- filtered = rdd.filter(lambda row: "money" in row) Rien n'est exécuté à cette ligne
- filtered.take(10) Tout est exécuté à cette ligne

RDD: Caching

Lorsqu’un RDD est en cache, le résultat du RDD est conservé dans la mémoire des workers, il n’y a donc plus besoin de recalculer la chaîne à chaque fois. Ceci permet de gagner du temps au niveau de l'exécution des tâches.

Spark I

TP 1 Rdd sur Databricks

Databricks

Une plateforme unifiée pour faire du Machine Learning et du Data Engineering. Elle s’intègre bien avec les différents cloud providers et bien d’autres services.

Databricks- How To

1. Aller sur sur le site https://www.databricks.com/try-databricks pour s'inscrire
2. Choisissez l'option databricks-cloud-community
3. Créer un cluster dans la partie compute

4. Créer un nouveau dossier dans worskpace (en haut à gauche)
5. Dans ce dossier créer un notebook en utilisant l'option import
6. Importer le tp et attacher le cluster créé en 3. et c'est parti

Quiz Spark & RDD

1. Quels sont les différents modules de spark ?
2. A quels cas d'usages peut répondre spark ?
3. Citer quelques propriétés des rdds
4. Expliquer en quelque mots la notion de Lazy Evaluation