SPARK and MAP-REDUCE

Overview

1. MapReduce

MapReduce is a programming engine for processing and generating large data sets with a parallel, distributed algorithm on a cluster of computer.

MapReduce is composed of several components, including :

JobTracker -- the master node that manages all jobs and resources in a cluster

TaskTrackers -- agents deployed to each machine in the cluster to run the map and reduce tasks

JobHistoryServer -- a component that tracks completed jobs, and is typically deployed as a separate function or with JobTracker

2. SPARK

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

Spark serves several languages Scala, Python, R and Java.
Apache Spark is a powerful open source processing engine built around speed, ease of use, and sophisticated analytics.

MapReduce vs Spark

MAPREDUCE	SPARK
Mainly restricted for java developers	Java , Scala, python, R, SQL closure
Boiler plate coding	Conciseness
No interactive shell	REPL(Read evaluate print loop)
Disk based, performance is slow	Memory based
Only for batch processing	Batch as well as interactive processing
Not optimized for iterative algorithm	Best for iterative algorithms
No graph processing	Graph processing is supported

USE CASES

Below are some use cases and scenarios that will explain the benefits and advantages of Spark over MapReduce.

Some scenarios have solutions with both MapReduce and Spark, which makes it clear as to why one should opt for Spark when writing long codes.

Scenario 1: Simple word count example in MapReduce and Spark

The same code in MapReduce.

MapReduce

~~~~~~~~~

Step 1: Create a text file on which processing is to be done.

```
hadoop fs -mkdir -p /user/$USER/input
```

Step 2: Copy the text file from local file system to hdfs

```
hadoop fs -copyFromLocal sample.txt input
```

Step 3: Create wordcount file

```
package wc;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.*;

public class WordCount extends Configured implements Tool {
 public static void main(String args[]) throws Exception {
  int res = ToolRunner.run(new WordCount(), args);
  System.exit(res);
 }
  public int run(String[] args) throws Exception {
  Path inputPath = new Path(args[0]);
  Path outputPath = new Path(args[1]);
  Configuration conf = getConf();
  Job job = new Job(conf, this.getClass().toString());
  FileInputFormat.setInputPaths(job, inputPath);
  FileOutputFormat.setOutputPath(job, outputPath);

  job.setJobName("WordCount");
  job.setJarByClass(WordCount.class);
  job.setInputFormatClass(TextInputFormat.class);
```

```
 job.setOutputFormatClass(TextOutputFormat.class);
 job.setMapOutputKeyClass(Text.class);
 job.setMapOutputValueClass(IntWritable.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);

 job.setMapperClass(Map.class);
 job.setCombinerClass(Reduce.class);
 job.setReducerClass(Reduce.class);
 return job.waitForCompletion(true) ? 0 : 1;
}
 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,
  Mapper.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 value : values) {

     sum += value.get();

    }

    context.write(key, new IntWritable(sum));

   }

 }

}
```

Step 4: Execute the jar file

jar cf wordcount.jar WordCount*.class

hadoop jar wordcount.jar WordCount input output

Step 5: Check the output from the two partitions

hadoop fs -tail output/part-r-00000 | tail > sample-tail.out

Spark

~~~~~~

Step 1: Open up the spark-shell (Scala or python)

Step 2: In scala shell

```
val rdd1=sc.textFile("sample.txt")
val rdd2=rdd1.flatMap(line => line.split( ))
val rdd3=rdd2.map(word => (word,1))
val rdd4=rdd3.reduceByKey((v1,v2)=>(v1+v2))
rdd4.collect()
rdd4.saveAsTextFile("/user/input/wordcount")
```

VERDICT

The 100 lines of code of a simple Word Count Program have been limited to just less than 10 lines. It shows the efficiency of Spark and ease in code.

Scenario 2: Three csv files have been given,

EmployeeName.csv with field (id, name)

EmployeeManager.csv(id,managerName)

EmployeeSalary.csv (id,salary)

Data:

EmployeeManager.csv

E01,Vishnu

E02,Satyam

E03,Shiv

E04,Sundar

E05,John

E06,Pallavi

E07,Tanvir

E08,Shekhar

E09,Vinod

EmployeeName.csv

E01,Lokesh

E02,Bhupesh

E03,Amit

E04,Ratan

E05,Dinesh

E06,Pavan

E07,Tejas

E08,Sheela

E09,Kumar

E10,Venkat

EmployeeSalary.csv

E01,50000

E02,50000

E03,45000

E04,45000

E05,50000

E06,45000

E07,50000

E08,10000

E09,10000

E10,10000

Steps: Open up scala shell using spark-shell command

```
val manager = sc.textFile("EmployeeManager.csv")
val m = manager.map(x=>(x.split(',')(0),(x.split(',')(1))))
val name = sc.textFile("EmployeeName.csv")
val n = name.map(x=>(x.split(',')(0),(x.split(',')(1))))
val salary = sc.textFile("EmployeeSalary.csv")
val s = salary.map(x=>(x.split(',')(0),(x.split(',')(1))))
val joined =n.join(s).join(m)
val result=joined.map(x=>(x._1, x._2._1._1, x._2._1._2 )).toString()
```

Scenario 3: To find the occurrence of a particular name from a file

Find the name Richard occurs how many times.

Data

~~~~~

Employee.txt

E01,Lokesh

E02,Bhupesh

E03,Amit

E04,Richard

E05,Dinesh

E06,Pavan

E07,Tejas

E08,Sheela

E09,Kumar

E10,Richard

E011,Lokesh

E012,Bhupesh

E013,Amit

E014,Richard

E015,Dinesh

E016,Pavan

E017,Tejas

E018,Sheela

E019,Kumar

E020,Richard

Steps: Open up the spark shell

```
val data=sc.textFile("Employee.txt")
val rdd1=data.map(x=>x.split(',')).filter(x=>x(1).contains('Richard'))
val rdd2=rdd2.map({case x=>x(1)->1}).reduceByKey(_+_)
rdd2.count()
rdd2.saveAsTextFile("user/data/name")
```

Scenario 3: Sanfranciso Police Department publishes the crime data

Following are fields in the dataset:

incidentnum,category,description,dayofweek,date,time,pddistrict,resolution,address,X,Y, pdid

What are the top 5 addresses with most incidents?

Steps: Open up the scala shell

```
val pair = sc.textFile("sample.txt").map(l=>l.split(',') .map(w=>(w(0),w(8)))
val count=pair.reduceByKey(v1,v2=>v1+v2).groupByKey()
System.out.println(count.take(5))
```

Scenario 4: Sanfranciso Police Department publishes the crime data

Following are fields in the dataset:

incidentnum,category,description,dayofweek,date,time,pddistrict,resolution,address,X,Y, pdid

What are the top 5 districts with most incidents?

Steps: Open up the scala shell

```
val pair = sc.textFile("sample.txt").map(l=>l.split(',') .map(w=>(w(0),w(6)))
val count=pair.reduceByKey(v1,v2=>v1+v2).groupByKey()
System.out.println(count.take(5))
```

Scenario 5: Sanfranciso Police Department publishes the crime data

Following are fields in the dataset:

incidentnum,category,description,dayofweek,date,time,pddistrict,resolution,address,X,Y, pdid

What are the top 10 resolutions?

Steps: Open up the scala shell

```
val counts = sc.textFile("sample.txt").map(l=>l.split(',').map(w=>(w(7),1))
.reduceByKey(v1,v2=>v1+v2).sortByKey()

System.out.println(counts.take(10))
```

Scenario 6: Sanfranciso Police Department publishes the crime data

Following are fields in the dataset:

incidentnum,category,description,dayofweek,date,time,pddistrict,resolution,address,X,Y, pdid

What are the top 10 categories of incidents?

Steps: Open up the scala shell

```
val counts = sc.textFile("sample.txt").map(l=>l.split(',').map(w=>(w(1),1))
        .reduceByKey(v1,v2=>v1+v2).sortByKey()
System.out.println(counts.take(10))
```

Scenario 7:

Airport authority of USA have provided their data in Airport.csv file with following details:

airports.csv have fields[iata, Airport, City, State, Country, Latitude, Longitude]

Make a Spark application for:

(a)**count of total number of airports in USA**

(b) **to list all the airports in state CO**

(c) **generate text file in following keyvalue pair (state,(airports))**

Scenario 8:

A database from an organization provides some details about the employees working there in a text file which is comma separated format NamesText.txt file

NamesText.txt have fields [YEAR,FirstName,County,Gender,Count]

Make a Spark application for:

(a) **to count total lines in file**

(b**) to count distinct names**

(c) **to find total of count for every first name**

(d) **to find how many users have firstname "DAVID"**

(e**) to find how many Male and Female employees are**

Scenario 9: **To remove the contents present in file1, from file2**

Data

~~~~~

Context.txt

Hello and welcome to Hadoop Classes.

Today we will study about Apache Spark.

This will be an interesting Session.

Thank you.

Remove.txt

Hello, an, we

Steps: Open up the scala/python shell using spark-shell or pyspark

```
val context=sc.textFile("Context.txt")

val remove=sc.textFile("Remove.txt")

val c=context.flatMap(x=>x.split(' ')).map(w=>w.trim)

val r=remove.flatMap(x=>x.split(',')).map(w=>w.trim)

val sub=c.subtract(r)

sub.collect

sub.saveAsTextFile("/user/data/omit")
```
~~~~~

Scenario 10: A data file of employee with name, gender and salary is given

Data.txt

~~~~~~~~~~

Vishnu, Male,53000

Satyam,Male,67000

Shiv,Male,700000

Sundar,Male,45000

Anna,Female,27000

Pallavi,Female,56000

Tango,Male,34000

Samantha,Female,42000

Vinay,Male,45000

Raj,Male,23000

Steps: Open up the shell using pyspark or spark-shell Using scala shell

```
val rdd=sc.textFile("data.txt")
 val byKey=rdd.map({case(name,gen,sal)=>(name,gen) -> sal})
 val sum=byKey.reduceByKey(_+_)
  sum.collect()
```