Getting Started with *Spark*

Shadi Ibrahim
March 30th, 2017

MapReduce has emerged as a leading programming model for data-intensive computing. It was originally proposed by Google to simplify development of web search applications on a large number of machines.

Apache Spark is a powerful open source processing engine for Hadoop data built around speed, ease of use, and sophisticated analytics. It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010.

Spark enables applications in Hadoop clusters to run up to 100x faster in memory, and 10x faster even when running on disk. It lets you quickly write applications in Java, Scala, or Python. It comes with a built-in set of over 80 high-level operators. And you can use it interactively to query data within the shell. Sophisticated Analytics. In addition to simple "map" and "reduce" operations, Spark supports SQL queries, streaming data, and complex analytics such as machine learning and graph algorithms out-of-the-box. Better yet, users can combine all these capabilities seamlessly in a single workflow.

The goal of this TP is to study the implementation and the operation of the Spark Platform. We will see how to deploy the platform, alongside Hadoop. We will run simple examples using the MapReduce paradigm.

Exercise 1: Installing Spark platform on your local machine alongside your existing Hadoop

The goal of this exercise is to learn how to set up and configure a single-node Spark installation so that you can quickly perform simple operations using Hadoop MapReduce and the Hadoop Distributed File System (HDFS).

Question 1.1

Download the Hadoop platform from (https://spark.apache.org). Extract the contents of spark-1.1.0-bin-hadoop1-2.tar.gz in your home. To run Spark, we assume that you have successfully deployed and configured your Hadoop platform

To make Spark aware of your Hadoop configuration, go to your Spark home directory, in `conf/spark-env.sh.template` add:

```
HADOOP_CONF_DIR=../hadoop/conf
```

Now save the file as `spark-env.sh`.

Then you will need to add your workers to the slaves file, go to your Spark home directory, in `conf/slaves` add:

```
xxx.rennes.grid5000.fr
yyy.rennes.grid5000.fr
```

To be able to view the web UI of an application, set spark.eventLog.enabled to true before starting the application, go to your Spark home directory, in `conf/spark-defaults.conf.template` add:

```
spark.eventLog.enabled true
```

Now save the file as `spark-defaults.conf`.

Question 1.2

We will now start the platform, and start all daemons:

- First starting HDFS:
  - Format a new distributed-filesystem:
    $ `bin/hadoop namenode -format`

  - Start the HDFS daemons:
    $ `bin/start-dfs.sh`
  - A nifty tool for checking whether the expected HDFS processes are running is `jps`.

- Starting Spark master and workers:

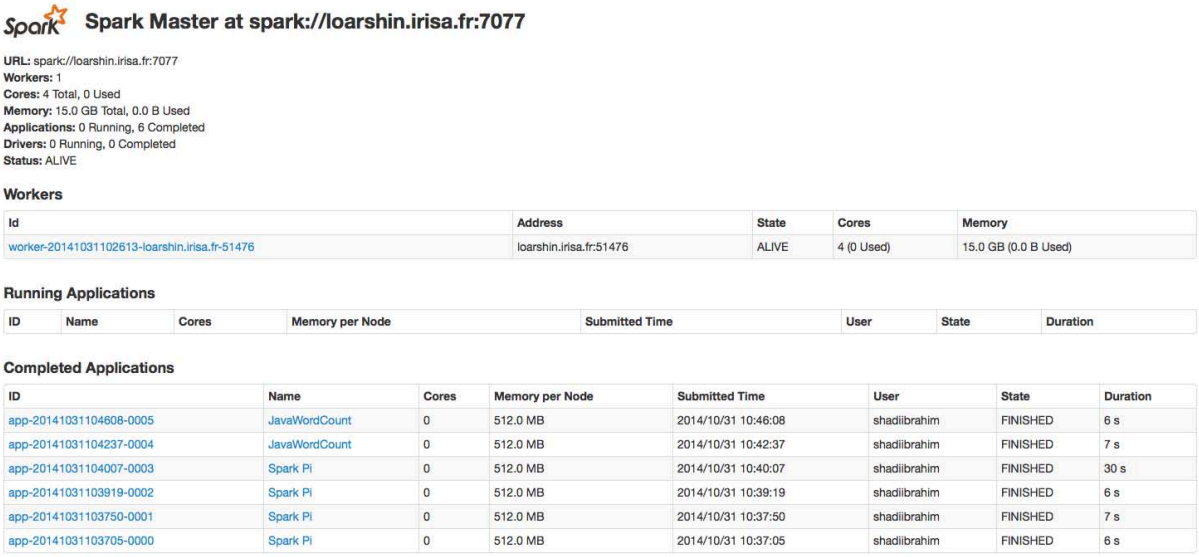
  - Start the Spark daemons:
    $ `sbin/start-all.sh`
  - A nifty tool for checking whether the expected Spark processes are running is `jps`.

Question 1.3

To Browse the web interface for the NameNode and the JobTracker; by default they are available at:

- Master - http://Master:8080/
- Worker - http://worker:8081/

The Master and Worker GUI should look like this:



Spark Master at spark://loarshin.irisa.fr:7077

URL: spark://loarshin.irisa.fr:7077
Workers: 1
Cores: 4 Total, 0 Used
Memory: 15.0 GB Total, 0.0 B Used
Applications: 0 Running, 6 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Id	Address	State	Cores	Memory
worker-20141031102613-loarshin.irisa.fr-51476	loarshin.irisa.fr:51476	ALIVE	4 (0 Used)	15.0 GB (0.0 B Used)

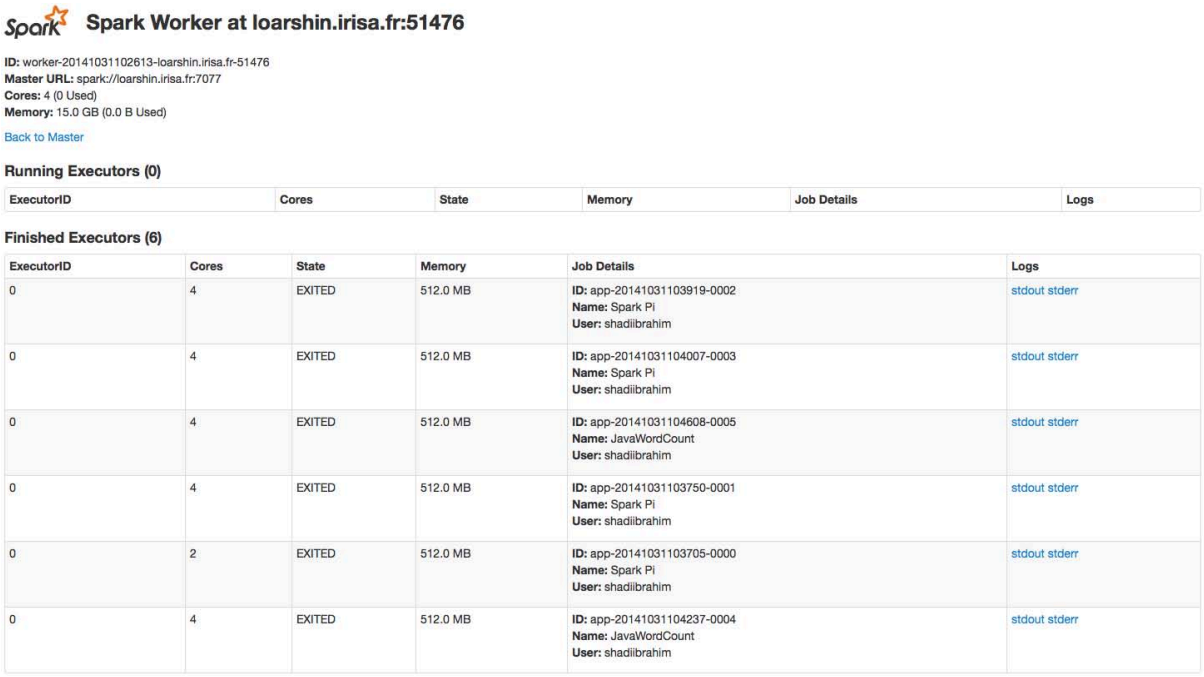
Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Completed Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20141031104608-0005	JavaWordCount	0	512.0 MB	2014/10/31 10:46:08	shadiibrahim	FINISHED	6 s
app-20141031104237-0004	JavaWordCount	0	512.0 MB	2014/10/31 10:42:37	shadiibrahim	FINISHED	7 s
app-20141031104007-0003	Spark Pi	0	512.0 MB	2014/10/31 10:40:07	shadiibrahim	FINISHED	30 s
app-20141031103919-0002	Spark Pi	0	512.0 MB	2014/10/31 10:39:19	shadiibrahim	FINISHED	6 s
app-20141031103750-0001	Spark Pi	0	512.0 MB	2014/10/31 10:37:50	shadiibrahim	FINISHED	7 s
app-20141031103705-0000	Spark Pi	0	512.0 MB	2014/10/31 10:37:05	shadiibrahim	FINISHED	6 s

Figure 1: Spark Master



Spark Worker at loarshin.irisa.fr:51476

ID: worker-20141031102613-loarshin.irisa.fr-51476
Master URL: spark://loarshin.irisa.fr:7077
Cores: 4 (0 Used)
Memory: 15.0 GB (0.0 B Used)

Back to Master

Running Executors (0)

ExecutorID	Cores	State	Memory	Job Details	Logs

Finished Executors (6)

ExecutorID	Cores	State	Memory	Job Details	Logs
0	4	EXITED	512.0 MB	**ID:** app-20141031103919-0002 **Name:** Spark Pi **User:** shadiibrahim	stdout stderr
0	4	EXITED	512.0 MB	**ID:** app-20141031104007-0003 **Name:** Spark Pi **User:** shadiibrahim	stdout stderr
0	4	EXITED	512.0 MB	**ID:** app-20141031104608-0005 **Name:** JavaWordCount **User:** shadiibrahim	stdout stderr
0	4	EXITED	512.0 MB	**ID:** app-20141031103750-0001 **Name:** Spark Pi **User:** shadiibrahim	stdout stderr
0	2	EXITED	512.0 MB	**ID:** app-20141031103705-0000 **Name:** Spark Pi **User:** shadiibrahim	stdout stderr
0	4	EXITED	512.0 MB	**ID:** app-20141031104237-0004 **Name:** JavaWordCount **User:** shadiibrahim	stdout stderr

Figure 2: Spark Worker

Question 1.4

To stop all the daemons running on your machine:
In Hadoop directory:
$ `bin/stop-all.sh`
In Spark directory:
$ `sbin/stop-all.sh`

Exercise 2: Running your first MapReduce program

The goal of this exercise is to execute two MapReduce examples, typically used for benchmarking, which come with the default Spark distribution.

Question 2.1

Run the Pi estimator:

- `spark-submit --class org.apache.spark.examples.SparkPi --master spark://yourMasterNode:7077 (check the Master GUI) --num-executors NUMBER (number of executors) --executor-memory (Amount of memory to use per executor process e.g., 512m) --executor-cores X (number of cores -- concurrent tasks-- per executor) ./lib/spark-examples-1.1.0-hadoop1.0.4.jar samples`

Run the Pi estimator with 10000 sample.

Question 2.2

Run the wordcount example:

- `spark-submit --class org.apache.spark.examples.JavaWordCount --master spark://yourMasterNode:7077 (check the Master GUI) --num-executors NUMBER (number of executors) --executor-memory (Amount of memory to use per executor process e.g., 512m) --executor-cores X (number of cores -- concurrent tasks-- per executor) ./lib/spark-examples-1.1.0-hadoop1.0.4.jar hdfs://namenode:9000/input`

Use the 2 GB data set and check the output (notice that the output is printed on your screen).

To write the output to HDFS, copy the `spark-example.jar` to the lib directory then run the following command:

- `spark-submit --class org.apache.spark.examples.JavaWordCount --master spark://yourMasterNode:7077 (check the Master GUI) --num-executors NUMBER (number of executors) --executor-memory (Amount of memory to use per executor process e.g., 512m) --executor-cores X (number of cores -- concurrent tasks-- per executor) ./lib/spark-example.jar hdfs://namenode:9000/input hdfs://namenode:9000/output`

Use the 2 GB data set and check the output in the HDFS.

Question 2.3

Run Wordcount using the 2 GB data set and check the output in the HDFS. Use three different scenarios (1 core, two cores and 3 cores)
See the execution times, what can you observe?

Exercise 3: Running your Spark application using the Scala shells

To start one of the shell applications, run one of the following commands:
In Spark directory:
$ `bin/spark-shell`

```
scala> val myfile = sc.textFile("hdfs://namenode:8020/path-to-input")
scala> val counts = myfile.flatMap(line => line.split(" ")).map(word => (word, 1)).
       reduceByKey(_ + _)
scala> counts.saveAsTextFile("hdfs://namenode:8020/path-to-output")
```

Figure 3

Question 3.1

Run Wordcount using the 2 GB data set and check the output in the HDFS.

Exercise 4: Running your Spark application using the Python shells

To start one of the shell applications, run one of the following commands: In Spark directory:
$ `bin/bin/pyspark`

```
>>> myfile = sc.textFile("hdfs://namenode:8020/path-to-input")
>>> counts = myfile.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1)).
    reduceByKey(lambda v1,v2: v1 + v2)
>>> counts.saveAsTextFile("hdfs://namenode:8020/path-to-output")
```

Figure 4

Question 4.1

Run Wordcount using the 2 GB data set and check the output in the HDFS.