Guide: Install Spark in Local Mode

Ignacio M. Llorente, Simon Warchol

v3.0 - 29 March 2021

Abstract

This is a guideline document to show the necessary actions to set up Apache Spark (2.x) on Ubuntu (18.04). Spark can be run using the built-in standalone cluster scheduler in the local mode. This means that all the Spark processes are run within the same JVM-effectively, a single, multithreaded instance of Spark. The local mode is used for prototyping, development, debugging, and testing. However, this mode can also be useful in real-world scenarios to perform parallel computation across multiple cores on a single computer. As Spark's local mode is fully compatible with the cluster mode; programs written and tested locally can be run on a cluster with just a few additional steps.

Requirements

- **First you should have followed the Guide "First Access to AWS"**. It is assumed you already have an AWS account and a key pair, and you are familiar with the AWS EC2 environment.

Acknowledgments

The author is grateful for constructive comments and suggestions from David Sondak, Charles Liu, Matthew Holman, Keshavamurthy Indireshkumar, Kar Tong Tan, Zudi Lin, Nick Stern, Dylan Randle, Hayoun Oh, Zhiying Xu and Zijie Zhao.

1. Spin up EC2 instance

1. Start an EC2 instance with Ubuntu 18.04. We strongly recommend an instance with at least 4 vCPUs (**m4.xlarge**) to be able to evaluate parallel implementation. Use your CS205-key
2. Connect to this instance w/ `ssh` (or putty on Windows), again using your CS205-key.

2. Install Java

- Java should be installed in the machine to run Apache Spark. The subsequent commands help quickly install Java in an Ubuntu machine. Follow through with the instructions to install java on your virtual machine.

```
$ sudo apt-add-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt install openjdk-8-jdk
```

- To check the Java installation is successful run following command in the terminal

```
$ java -version
openjdk version "1.8.0_282"
OpenJDK Runtime Environment (build 1.8.0_282-8u282-b08-0ubuntu1~18.04-b08)
OpenJDK 64-Bit Server VM (build 25.282-b08, mixed mode)
```

3. Install Scala

- Install Scala

```
$ sudo apt-get install scala
```

- To check the Scala installation is successful run following command in the terminal

```
$ scala -version
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
```

4. Install Python

- Install Python

```
$ sudo apt-get install python
```

- To check the Python installation is successful run following command in the terminal

```
$ python -h
```

5. Install Spark

- Download and untar the Apache Spark 2 distribution to `/usr/local/spark`

```
$ sudo curl -O
http://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-hadoop2.7.tgz
$ sudo tar xvf ./spark-2.2.0-bin-hadoop2.7.tgz
$ sudo mkdir /usr/local/spark
$ sudo cp -r spark-2.2.0-bin-hadoop2.7/* /usr/local/spark
```

6. Configure Environment

- Add `/usr/local/spark/bin` to `PATH` in `.profile`. To do so, add the following line to the end of the `~/.profile` file with a text editor, for example with `vi ~/.profile`

```
export PATH="$PATH:/usr/local/spark/bin"
```

- Execute `source ~/.profile` to update PATH in your current session
- If you are using an AWS VM, you should include the internal hostname and IP to `/etc/hosts` with a text editor, for example with `sudo vim /etc/hosts`
- In my specific case, the file contains:

```
$ cat /etc/hosts
127.0.0.1 localhost
172.30.4.210 ip-172-30-4-210
```

7. Test Installation

- This section will run an example in Spark's local mode. In this mode, all the Spark **tasks** are run within the same JVM (**executor**), and Spark uses multiple cores (threads) for parallel processing within the executor.
    - Spark creates one task for each partition in the RDD. In this code the number of `partitions` is given as an argument.
    - These parallel tasks are executed in parallel on a number of cores (threads). By default, the example uses a number of cores (threads) available on your system.

```
$ run-example SparkPi 10
```

In a m4.xlarge instance, the log shows how 10 tasks are created and scheduled on 4 cores (threads). Then you get the output as Pi is roughly 3.14634 along with the execution time in seconds. You should see the following line in the log:

```
19/01/02 00:19:25 INFO DAGScheduler: Job 0 finished: reduce at
SparkPi.scala:38, took 0.675346 s Pi is roughly 3.143955143955144
```

- Check the source code of `pi.py` and other examples in

```
/usr/local/spark/examples/src/main/python
```

8. Try pyspark

Apache Spark comes with an interactive shell for python, which is the easiest way to try Spark. The shell for python is known as “pyspark”. The pyspark shell is responsible for linking the python API to the spark core and initializing the Spark context.

- Upload to the VM the input.txt file with the ebook of Moby Dick used in the MapReduce labs
- Execute `pyspark`

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0
      /_/

Using Python version 2.7.17 (default, Feb 27 2021 15:10:58)
SparkSession available as 'spark'.
```

- Read the input file in a dataset, show the first element and count the total number of elements

```
>>> text_RDD = sc.textFile('/home/ubuntu/input.txt')
>>> text_RDD.first()
u'The Project Gutenberg EBook of Moby Dick; or The Whale, by Herman Melville'
>>> text_RDD.count()
22108
```

- Count the word frequency

```
>>> text_RDD.map(lambda x: x.replace(',',' ').replace('.',' 
').lower()).flatMap(lambda x: x.split()).map(lambda x: (x,
1)).reduceByKey(lambda x,y:x+y).collect()
```

- You can try any of the several pyspark tutorials available in the Internet

9. Submit a Spark Job

You have learned how to implement various spark RDD concepts in interactive mode using pyspark. However, data engineers cannot perform all the data operations in interactive mode every time. Once the data pipeline and transformations are planned and execution is finalized, the entire code is put into a python script that would run the same spark application in standalone mode.

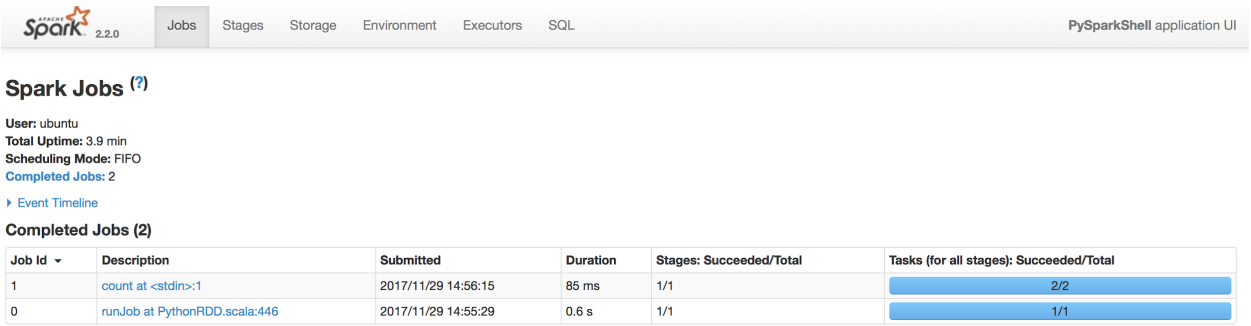
- Upload to the VM the Spark `wordcount.py` script and the `input.txt` file with the ebook of Moby Dick used in the MapReduce labs
- Check the source code of the `wordcount.py` script
- Submit the job

```
$ spark-submit wordcount.py
17/09/07 16:52:42 INFO SparkContext: Running Spark version 2.2.0
17/09/07 16:52:42 INFO SparkContext: Submitted application: WordCount
17/09/07 16:52:42 INFO SecurityManager: Changing view acls to: hadoop
17/09/07 16:52:42 INFO SecurityManager: Changing modify acls to: hadoop
17/09/07 16:52:42 INFO SecurityManager: Changing view acls groups to:
17/09/07 16:52:42 INFO SecurityManager: Changing modify acls groups to:
...
```

- When the program finishes, check file system again and look for the `output.txt` file (actually it is a folder contains the output files)

```
$ cat output.txt/*
('swimming', 1)
('seemed', 1)
('pilot', 1)
('told', 3)
('balaene', 1)
('more', 4)
('history', 3)
('man', 2)
```

- Every SparkContext launches a web UI, by default on port `4040`, that displays useful information about the application. You may find it useful to view results in the Spark UI, which is available at `http://localhost:4040` when running locally.



- In order to see the Spark UI, you must do the following:
  - Add an inbound rule to the instance that has the following properties:
    - Type: `Custom TCP Rule`
    - Port Range: `4040`
    - Source: `0.0.0.0/0`
  - Have pyspark running with the `pyspark` command
  - Open a local browser window to `http://localhost:4040`, replacing "`localhost`" in the URL with the public DNS of the instance
  - Repeat sample execution in Section 8 and monitor information, run the "Count the word frequency", go to the "Jobs" tab and refresh the page

10. Parallel Execution on a Single Node

In order to tune the parallel execution of a Spark application, we should adjust (1) the number of **Tasks** (level of parallelism in the application) created by Spark to process in parallel each RDD, and (2) the number of **Executors** (number of worker nodes in the parallel platform) and **Threads** (cores in each worker node) run in parallel by the underlying infrastructure to execute the tasks.

What is the number of independent tasks created by the application?

A **Task** is the unit of work that can be run on a partition of a distributed dataset. The unit of parallel execution is at the task level. All the tasks within a single stage can be executed in parallel.

Spark creates one task to process each RDD partition. RDD partitioning is a key property to parallelize a Spark application. A partition is a small chunk of a large distributed data set. Spark manages data using partitions that helps parallelize data processing with minimal data shuffle across the executors The number of partitions for a given RDD is:

- Given by the programmer (`parallelize`), or configured by `spark.default.parallelism` if none is given, for new created data.
- Determined by parent RDDs for RDDs produced as a result of transformation.
- Defined by the underlying file system if the RDD is created from a file. RDDs produced by textFile or hadoopFile have their partitions determined by the underlying MapReduce InputFormat that's used. The local file system creates one partition for each 32MB and the Hadoop file system creates one partition for each 128MB.

What is the number of executors provided by a platform?

An **Executor** is a single JVM process which is launched for an application on a worker node. Executor runs tasks and keeps data in memory or disk storage across them. Each application has its own executors. A single node can run multiple executors, and executors for an application can span multiple worker nodes.

When running Spark on a single node, there is a single executor to run the application that can be **multi-threaded** to use the multi-core parallelism. Spark properties control most application settings and are configured separately for each application. These properties can be set directly on a `SparkConf` passed to your `SparkContext`. `SparkConf` allows you to configure some of the common properties, like for example, the number of threads of the application. Running with `local[2]` means

two threads - which represents “minimal” parallelism, which can help detect bugs that only exist when we run in a distributed context or reduces execution time on multi-core systems.

This is configured in Java w/

```
val conf = new SparkConf().setMaster("local[2]").setAppName("Pi")
val sc = new SparkContext(conf)
```

or like so in pyspark

```
conf = SparkConf().setMaster("local[2]").setAppName("Pi")

sc = SparkContext(conf = conf)
```

In local mode, `setMaster` passed to Spark can be in one of the following formats

- `local`: Run Spark locally with one worker thread.
- `local[K]`: Run Spark locally with K worker threads (ideally, set this to the number of cores on your machines).
- `local[*]`: Run Spark locally with as many worker threads as logical cores on your machine.

Upload to the VM the Spark `pi.py` script, execute the code in local mode, and calculate the speedup for different number of threads (cores), from 1 to 4, by tuning the `SetMaster` property. For this simple code, this parameter sets both the partitions of the RDD and the number of simultaneous tasks.

Stop your instances when are done for the day to avoid incurring charges
Terminate them when you are sure you are done with your instance