

Submitting Spark Applications

Starting an interactive session:

1. Start an interactive session asking three nodes, using the ***start_interactive*** script . This script requests by default one node from the y15 reservation **for an hour**:
 - a. Modify ***./start_interactive.sh***. script with your reservation number and walltime.

```
#qsub -IVl select=3:ncpus=36,walltime=01:00:00,place=scatter:excl -A y15  
-q <reservation number> -j oe
```

This will give you an interactive session into a node (e.g. node **r1i3n0**) and you will see something like this:

```
[USERNAME@r1i3n0~]$ ./start_interactive.sh  
qsub: waiting for job 399686.indy2-login0 to start  
qsub: job 399686.indy2-login0 ready  
[USERNAME@r1i3n0~]$
```

Starting a Spark cluster :

2. Start the spark cluster using the ***start_spark*** script. It will configure a spark cluster, with the master running in a node, and three workers (with 72 cores per worker) running in each reserved node. Remember to use **source** for exporting the environment variables into your PATH.

```
[USERNAME@r1i3n0~]$ source ./start_spark.sh  
3 node(s) assigned  
Autoloading gcc/6.2.0  
starting org.apache.spark.deploy.master.Master, logging to  
/lustre/home/y15/rosaf3/spark-2.4.0-bin-hadoop2.7/logs/spark-  
rosaf3-org.apache.spark.deploy.master.Master-1-r1i3n0.out  
r1i3n0: starting org.apache.spark.deploy.worker.Worker, logging to  
/lustre/home/y15/rosaf3/spark-2.4.0-bin-hadoop2.7/logs/spark-  
rosaf3-org.apache.spark.deploy.worker.Worker-1-r1i3n0.out  
r1i3n14: starting org.apache.spark.deploy.worker.Worker, logging  
to /lustre/home/y15/rosaf3/spark-2.4.0-bin-hadoop2.7/logs/spark-  
rosaf3-org.apache.spark.deploy.worker.Worker-1-r1i3n14.out  
r1i3n13: starting org.apache.spark.deploy.worker.Worker, logging  
to /lustre/home/y15/rosaf3/spark-2.4.0-bin-hadoop2.7/logs/spark-  
rosaf3-org.apache.spark.deploy.worker.Worker-1-r1i3n13.out  
starting org.apache.spark.deploy.history.HistoryServer, logging to  
/lustre/home/y15/rosaf3/spark-2.4.0-bin-hadoop2.7/logs/spark-  
rosaf3-org.apache.spark.deploy.history.HistoryServer-1-r1i3n0.out
```

Note: In this example, master node is r1i3n0

Submitting a Spark Application to the Spark Cluster – using cluster mode:

3. Submit the **SparkPi** application (which is an example that comes with spark) to your Spark Cluster that you have started by using **spark_submit_SparkPi** script, indicating your **master node** as the first argument (e.g. r1i3n0), which corresponds to the node assigned in your interactive session.

```
./spark_submit_SparkPi.sh r1i3n0
```

4. Check the result inside the \$HOME/spark-2.4.0-bin-hadoop2.7/work/ directory. The result will be a driver directory corresponding with this application:
 - a. more \$HOME/spark-2.4.0-bin-hadoop2.7/work/driver-20190106121316-0001/stdout | grep Pi → Pi is roughly 3.1403511403511404
5. To monitor the spark applications in your spark cluster, just launch another terminal session and run something like this:

- a. ssh **USER**@login.cirrus.ac.uk -L8080:MASTER_NODE:8080
- b. Web browser → localhost:8080

Note: In our example, MASTER NODE is r1i3n0. This node could be different each time you start an interactive session.

Memory in use: 751.7 GB Total, 0.0 B Used
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 1 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20190106115330-10.148.0.219-38018	10.148.0.219:38018	ALIVE	72 (0 Used)	250.6 GB (0.0 B Used)
worker-20190106115339-10.148.0.246-44268	10.148.0.246:44268	ALIVE	72 (0 Used)	250.6 GB (0.0 B Used)
worker-20190106115339-10.148.0.248-39230	10.148.0.248:39230	ALIVE	72 (0 Used)	250.6 GB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

Running Drivers (0)

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
---------------	----------------	--------	-------	-------	--------	------------

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190106115555-0000	Spark Pi	215	1024.0 MB	2019/01/06 11:55:55	rosaf3	FINISHED	3 s

Completed Drivers (1)

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class
driver-20190106115552-0000	2019/01/06 11:55:52	worker-20190106115339-10.148.0.246-44268	FINISHED	1	200.0 GB	org.apache.spark.examples.SparkPi

Application: Spark Pi

ID: app-20190106123522-0000
 Name: Spark Pi
 User: rosa3
 Cores: Unlimited (215 granted)
 Executor Limit: Unlimited (3 granted)
 Executor Memory: 1024.0 MB
 Submit Date: 2019/01/06 12:35:22
 State: FINISHED

Executor Summary (3)

ExecutorID	Worker	Cores	Memory	State	Logs
------------	--------	-------	--------	-------	------

Removed Executors (3)

ExecutorID	Worker	Cores	Memory	State	Logs
2	worker-20190106123354-10.148.0.193-45231	71	1024	KILLED	stdout stderr
1	worker-20190106123354-10.148.0.217-32989	72	1024	KILLED	stdout stderr
0	worker-20190106123355-10.148.0.231-39996	72	1024	KILLED	stdout stderr

Tuning Resource Allocations:

- We can also modify the number of cores to use per worker/executor, by using the `--total-executor-cores` in our spark-submission script.

For example, if we decide to use 72 cores for running this spark application:

```
>> spark-2.4.0-bin-hadoop2.7/bin/spark-submit --verbose --class org.apache.spark.examples.SparkPi
--master spark://r1i3n0:7077 --deploy-mode cluster --total-executor-cores 72 spark-2.4.0-bin-
hadoop2.7/examples/jars/spark-examples_2.11-2.4.0.jar 10
```

Note: You will have to modify this script to replace the master node (`--master spark://MASTER NODE:7077`), with the node where you master is running.

We can see that the spark cluster will run the application using the 3 workers/executors, but using only 24 cores per worker/executor.

Application: Spark Pi

ID: app-20190106123838-0001
 Name: Spark Pi
 User: rosa3
 Cores: 72 (72 granted, 0 left)
 Executor Limit: Unlimited (3 granted)
 Executor Memory: 1024.0 MB
 Submit Date: 2019/01/06 12:38:38
 State: FINISHED

Executor Summary (3)

ExecutorID	Worker	Cores	Memory	State	Logs
------------	--------	-------	--------	-------	------

Removed Executors (3)

ExecutorID	Worker	Cores	Memory	State	Logs
2	worker-20190106123355-10.148.0.231-39996	24	1024	KILLED	stdout stderr
1	worker-20190106123354-10.148.0.217-32989	24	1024	KILLED	stdout stderr
0	worker-20190106123354-10.148.0.193-45231	24	1024	KILLED	stdout stderr

Submitting a Spark Application to the Spark Cluster – using client mode:

- We can also submit the wordcount.py application stored in `Spark_Applications` folder, using the **client deployment mode**.

In cluster mode, the Spark driver runs inside an application master process which is managed by cluster-manager on the cluster, and the client can go away after initiating the application. **In client mode**, the driver runs in the client process, and the application master is only used for requesting resources from spark cluster.

```
>> export PYSPARK_PYTHON=/lustre/sw/anaconda/anaconda3-5.1.0/bin/python3
```

```
>> $SPARK_HOME/bin/spark-submit --verbose --master spark:// r1i3n0:7077 --deploy-mode client  
wordcount.py $HOME/spark-2.4.0-bin-hadoop2.7/README.md
```

Note: You will have to modify this script to replace the master node (`--master spark://MASTER NODE:7077`), with the node where you master is running.

You can replace `$HOME/spark-2.4.0-bin-hadoop2.7/README.md`, by other FILE that you will like to use as an input for counting the words in it.

Important: Currently, the **standalone mode** does not support **cluster mode** for **Python applications**