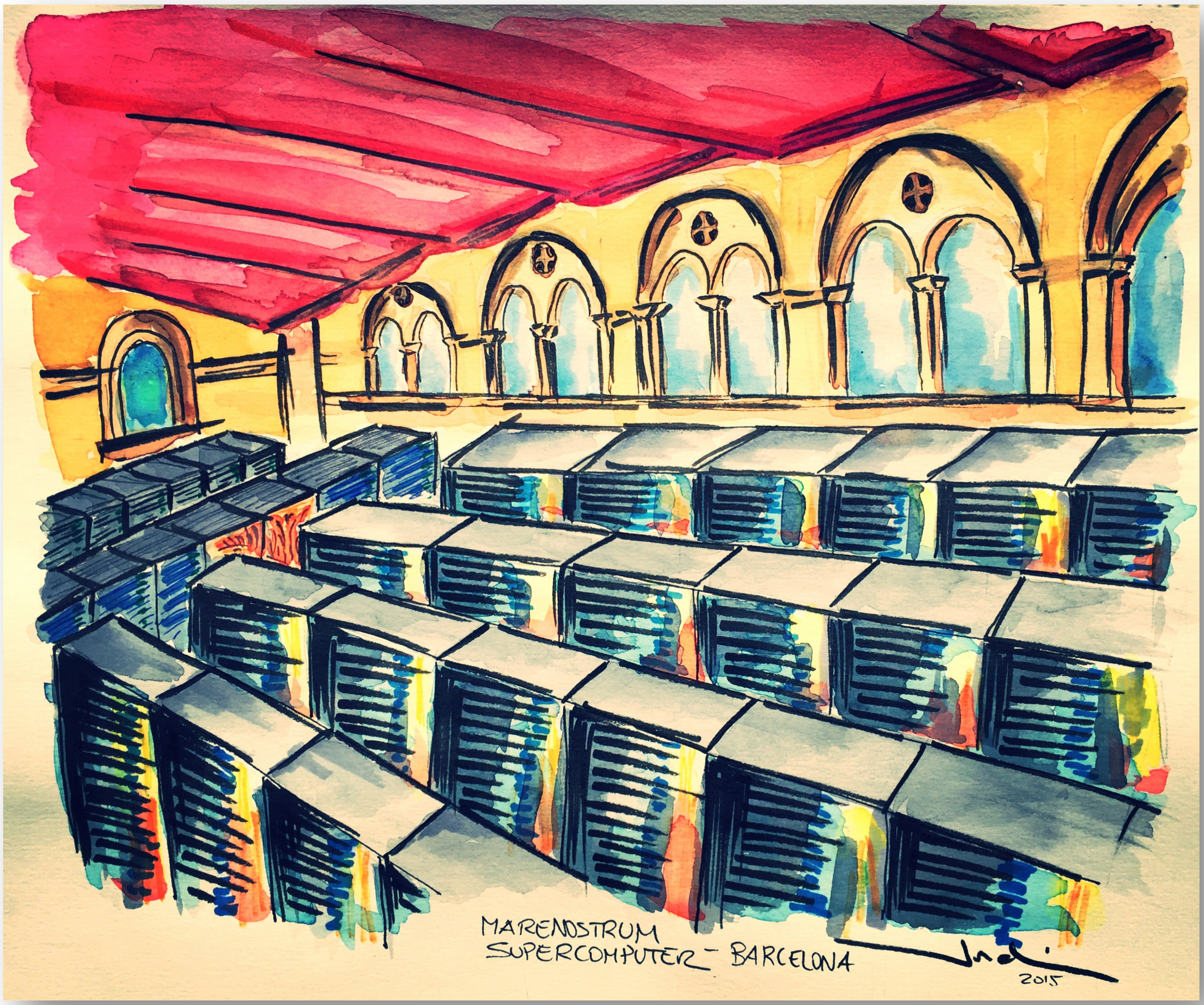
Hands-on 11:

Spark Deployment and Performance Evaluation on the Marenostrum III



SUPERCOMPUTERS ARCHITECTURE

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS

Barcelona, Fall 2015

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Hands-on 11

Spark Deployment and Performance Evaluation on the Marenustrum III

SUPERCOMPUTERS ARCHITECTURE

Master in innovation and research in informatics

(Specialization High Performance Computing)

UPC Barcelona Tech & Barcelona Supercomputing Center

Version 1.0 - 14 July 2015

Authors:
Albert Calvo - albert.calvo.ibanez@est.fib.upc.edu
Ruben Tous - ruben.tous@ac.upc.edu
Anastasios Gounaris - gounaria@csd.auth.gr
Jordi Torres - jordi.torres@bsc.es

Table of Contents

1 Hands-on content

This hand-on was created to help people start using Big Data Analysis with Spark on Mare Nostrum III. To complete this Hands-on successfully you may be familiar with the Linux shell and some basics of Scala.

Enjoy!

2 Before starting

2.1 Create a workspace in your laptop

As the previous Hands-on, part of the work will be done in you laptop. We suggest to use git to ease the development of this hands-on.

Before you start using Git, you have to make it available on your computer. Even if it's already installed, it's probably a good idea to update to the latest version. You can either install it as a package or via another installer, or download the source code and compile it yourself. Please, visit this web site and follow the installation instructions: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

We suggest to get a copy of SA-MIRI Git repository. The command you need is git clone. Git receives a full copy of nearly all data that the server has.

```
$ git clone https://github.com/jorditorresBCN/SA-MIRI.git
```

2.2 Maven

Maven is a Java tool, so you must have Java installed in order to proceed, with the same version as Marenostrum III. To know it you can run the following commands:

```
$ module load java/latest
$ java –version
```

For Desember 2015 course edition: **version Oracle SDK 1.8**

First, download Maven from https://maven.apache.org/download.cgi, and follow the installation instructions from https://maven.apache.org/install.html.

After that, type the following in a terminal or in a command prompt:

```
$ mvn --version
```

It should print out your installed version of Maven.

2.3 Project Structure

A good praxis in large software projects is to keep the code well organized and deal with decencies. One tool to achieve this objectives is Apache Maven.

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information[1].

A boilerplate has been created to start coding spark in less time as possible in this Hands-on:

```
$ cd ~/SA-MIRI/Hands-on-11
$ ls boilerplate/
README.md
pom.xml
src
target
```

this boilerplate contains the project structure and the configuration file that contains information to build the project and all decencies needed to build your project (to create the .jar file).

Clone the boilerplate repository in your workspace or download it in your local directory as a “pom.xml”

Assume that your project could have the following project structure:

```
/src
/src/main

/src/main/scala

/src/main/scala/bsc

/src/main/scala/bsc/Helloworld.scala
```

Exercise 1: Try to understand the pom.xml file and change it property for your

1 More Info: https://maven.apache.org/

project if necessary (not required to include answer in the report lab)

2.4 Check your connection to Marenostrum III

Use your login username and its associated password to get into the cluster through one of the following login nodes in Marenostrum III:

- mn1.bsc.es
- mn2.bsc.es
- mn3.bsc.es

Remember that you must use Secure Shell (ssh) tools to login into or transfer files into the cluster in order to enable secure logins over an insecure network.

```
$ ssh sam14030@mn1.bsc.es
Password:
```

3 Start coding

3.1 Create HelloWorld.scala program

The file *Helloworld.scala* contains the main class of the object to compile. At this point you can simply write a println("Hello World") or start coding real spark applications.

The following example contain a *wordcount* program that must be stored in

```
~/src/main/scala/bsc/HelloWorld.scala
```

Note that you have to upload a dataset in your profile and you need to create this workload file too.

```
package com.examples;
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
object Helloworld {
 def main(arg: Array[String]) {
      val conf = new SparkConf().setAppName("Simple Wordcount")
      val sc = new SparkContext(conf)
      //Your Spark code goes here
      val input ="/home/sam14/sam14030/datasets/dataset.txt"
     val output_dir = "/home/sam14/sam14030/wordcount"
     val logData = sc.textFile(input, 2).cache()
     val count = logData.flatMap(line => line.split(" "))
               .map(word => (word, 1))
               .reduceByKey(_ + _)
    count.saveAsTextFile(output_dir)
   }
}
```

(*) Note: I used my path /sam14/sam14030. You need to indicate yours.

3.2 Compilation with Maven

Run the following command in the folder that you create the pom.xml file.

```
$ mvn clean package
```

If everything went well the [artifactId]-[version].jar has been created in /target folder and see the following output:

```
. . .

[INFO] Building jar: /Users/jorditorres/Dropbox/github/SA-
MIRI/Hands-on-11/boilerplate/target/spark.helloworld-
1.0.0.jar
[INFO] ----------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------------
```

3.3 Upload the JAR using scp

Scp allows you to upload a file into Mare Nostrum Infrastructure. The usage is very simply, if you are not familiar with this tool the following example can help you to understand how it works.

```
$ scp target/spark.helloworld-1.0.0.jar sam14030@mn1.bsc.es:/home/sam14/sam14030/
```

4 Your first spark code in Marenostrum III

We assume that you have uploaded the jar file in your account. Now, in order to submit a spark job in Marenostrum we need to do some steps before using the *Spark for Marenostrum module* (Spark4MN), a module that enables data-intensive Spark workloads on MareNostrum.

Spark4MN runs over IBM LSF Platform workload manager, but it can be ported to clusters with different managers (e.g., Slurm) and does not rely on Spark cluster managers, like Apache Mesos and Hadoop YARN. Spark4MN is also in charge to manage the deployment of any additional resource Spark needs, such as a service-based distributed file system (DFS) like HDFS. Spark4MN is a collection of bash scripts with different commands. We will only use one command, which deploys all the Spark cluster's services, and executes the user applications.

4.1 Load the Spark module

Remember from a previous hands-on that Marenostrum III are using a environment modules package that provides a dynamic modification of a user's environment. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically, in a clean fashion.

First step is to be sure that we don't have any loaded module. Purge all previously loaded modules to ensure a clean execution

```
$ module purge
```

2. Load the module spark4mn, this package configure the shell to use spark:

```
$ module load spark4mn
```

4.2 Create a configuration file

3. Create an application configuration file according to the template available in the folder `/apps/SPARK4MN/3.1.0/doc/conf/` .

Some of the relevant options in the spark4mn configuration template are:

```
JOB_NAME=[STRING]
WORKING_DIR=[PATH]
WALLCLOCK=[INT]
[…]
SPARK_NNODES=[INT]
SPARK_NWORKERS_PER_NODE=[INT]
SPARK_NCORES_PER_WORKER=[INT]
SPARK_WORKER_MEM_SIZE=[INT]
SPARK_WORKER_AFFINITY=[CORE|SOCKET|…]
SPARK_NETWORK_INTERFACE=[IB|ETH]
DFS_MODULE=[HDFS|CASSANDRA|NONE]
DFS_SHARE_NODES=[BOOL]
[similar to Spark above]
ADDITIONAL_MODULES=[STRING,…]
ADDITION
AL_CLASSPATH=[PATH:PATH:…]
ADDITIONAL_LIBRARIES=[PATH:PATH:…]


[…]
```

It's recommended to copy the configuration file to $HOME. Instead, you'll have problems with permissions. This configuration file allows changing several parameters in order to tune the execution.

```
$  cp  /apps/SPARK4MN/3.1.0/doc/conf/spark4mn_job.conf.template
./job.conf
```

We suggest to modify only compulsory parameters: JAR file for each main problem to be executed and entry class for each main problem to be executed:

```
DOMAIN_JAR_1="/home/sam14/sam14030/spark.helloworld-1.0.0.jar";
DOMAIN_ENTRY_POINT_1="bsc.Main";
```

Warning: Not allowed blanks spaces before or after the equal sign

4.3 Submit the Spark job

As we explained, LSF is the utility used at Marenostrum III for batch processing support, so all jobs must be run through it. You can use spark4mn module to submit the job indicated in the configuration file as follow:

```
$ spark4mn job.conf
```

If everything works fine we will see the job (running or pending) in the queue with the command

```
$ bjobs
```

4.4 Have a look at execution trace files

After you submit a spark job, the output of the execution and metrics has been created and you can find it in the following files:

myspark_[jobID] .metrics: this file contains the time elapsed of the master and the workers.

myspark_[jobID]_logs: this folder contain some files with can be helpful to see the behavior of every worker along the cluster:

> *domain_1.log*: contain info about the creation of the Spark Environment
>
> *job.err*: contain info about errors, if exists.
>
> *job.out*: summary of the execution (CPU time, Max Memory, Max Processes ...)
>
> *resource_details.log*: information about the resources requested it may be equal to the values in the myspark.conf file.
>
> *spark/master.log and spark/worker_[1...N].log*: output generated during the execution at master and the slaves.

Exercise 2: Have a look of all these files. Include in your report the time that requires your program to be finished and detailed time exection of workers and master.

Exercise 3: Execute again the same program with two more workloads: (a) workload that has the double of data that the previous one and (b) a workload with only two words. Compare the results and include the results in your report.

5 Benchmarking Kmeans in Marenostrum III

As you know MLlib supports k-means clustering, one of the most commonly used clustering algorithms that clusters the data points into predefined number of clusters, used in a previous Hands-on. Because you are already familiar with this algorithm we will use it in this hands-on to benchmarking Marenostrum III.

5.1 Environment setup

In order to execute the k-means in Marenostrum III you have to proceed in the same way with the previous workcount example. However, in order to use Mlib library, you have to add a new dependency in the pom.xml

```
<depencies>
…
 <dependency>
       <groupId>org.apache.spark</groupId>
       <artifactId>spark-mllib_2.10</artifactId>
       <version>1.3.0</version>
         <scope>provided</scope>
     </dependency>
</depencies>
```

5.2 Program

Now you can, code the algorithm of kmeans in the main function. Enclosed some suggestions about the code.

```
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.linalg.Vectors

val numClusters = X
val numIterations = Y
val clusters = KMeans.train(rdd, numClusters, numIterations)

//Evaluate clustering by computing WithinSetSumofSquared Errors
val WSSSE = clusters.computeCost(parsedData)
println("Within Set Sum of Squared Errors = " + WSSSE)
```

5.3 Workload

In order to simplify this hands-on, we suggest to generate the data. However, if you are interested to analyse some specific data you can use scp command to transfer it to Marenostrum III.

In the SA-MIRI github you could find the program *DataGenerator* witch contain functions to create a random data. Import the file to the current project and create an rdd with these random data.

I suggest to use

```
val rdd = DataGenerator.generateKMeansVectors(sc, points,
numCols, numCenters, numPartitions, seed)
```

where numCenters is used to enforce that in the dataset exist clusters. This number must be equal to the number of clusters that we are using later. Other important parameter is numPartitions that indicates the number of RDDS that we will use.

5.4 Starting code

Enclosed you will find some code that could help you to create your code. Remember that first of all you need to import all the packages required

```
package bsc;

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

import org.apache.spark.mllib.clustering.{KMeansModel, KMeans}
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.rdd.RDD
```

Starting code to speed up and finish the hands-on in today lab. Please, produce your own code.

```
object Main {

  def main(args: Array[String]) {

    if (args.size < 1) {
        println("Error: Input size")
        System.exit(1)
    }

    //Variables for build the dataset
    val points        = args(0).toLong
    val numCols       = args(1).toInt
    val numCenters    = args(2).toInt
    val numPartitions = args(3).toInt
    val seed          = args(4).toInt

    //Variables for kmeans
    val numClusters   = args(5).toInt
    val iterations    = args(6).toInt

    val appName = "Kmeans"
    val conf = new SparkConf()
        .setAppName(appName)
    val sc = new SparkContext(conf)

     val rdd = DataGenerator.generateKMeansVectors(sc, points,
               numCols, numCenters, numPartitions, seed)

    val kmeans = KMeans.train(rdd, numClusters, iterations)

    val WSSSE = kmeans.computeCost(rdd)


    val CS = kmeans.k
      println("Num of clusters created" + CS)

    val CC = kmeans.clusterCenters
      for ( x <- CC ) {
        println("ClusterCenters" + x)
      }

    sc.stop()
    }
}
```

Note: In order to use the Scala arguments you should use job.conf variables DOMAIN_PARAMETERS_[[1-9]+]="[STRING, ...]"

Now we are ready to start your performance evaluation.

5.5 *Report Lab Performance evaluation*

Now it is time for evaluating the performance of k-means on Marenostrum.

I suggest executing the code with different workload size, different number of workers and different number of cores.

Exercise 4: Create and summit a Lab Report. Your Lab Report should contain the data that reinforce your conclusions.

Acknowledgement: Thank you to Carlos Tripiana and Cristian Morales from BSC Operations department for his invaluable help preparing this hands-on.