In order to launch DAAL spark samples you need to run spark on yarn.

How to install and run Hadoop & yarn can be found [here](https://medium.com/@yogeshdarji99/installing-multinode-hadoop-cluster-in-yarn-mode-86741c3df45b).

How to launch spark on yarn can be found [here](https://spark.apache.org/docs/latest/running-on-yarn.html).

**How to run**

In order to use DAAL, you need to follow a few simple steps

**Step 1: scala installation**

First you need to install scala and properly configure the environment

Download and install scala and add the necessary jars to the CLASSPATH env variable.

1. # download and install scala
2. wget <https://downloads.lightbend.com/scala/2.11.12/scala-2.11.12.rpm>
3. sudo rpm -i scala-2.11.12.rpm
4. export SCALA\_JARS=/usr/share/scala/lib/scala-library.jar
5. # adding needed jars to CLASSPATH env variable
6. declare -a jars=(
7. "spark-core\_"
8. "spark-sql\_"
9. "spark-catalyst\_"
10. "spark-mllib\_"
11. "hadoop-common-"
12. "jackson-annotations-"
13. "breeze\_"
14. "breeze-macros\_"
15. "hadoop-mapreduce-client-common-"
16. "hadoop-mapreduce-client-core-"
17. "spark-tags\_"
18. "spark-unsafe\_"
19. "log4j-"
20. )
22. **for** i **in** "${jars[@]}"
23. do
24. export CLASSPATH=`sudo find / -name "${i}\*.jar" | head -n 1`:$CLASSPATH
25. done

If needed, correct scala version in this script (scala samples validated on scala 2.11.11).

**Step 2: downloading and setting up daal environment**

If DAAL is already installed and the environment is configured in correct way, then just download DAAL Spark samples.

1. # download daal from the main repository
2. git clone https://github.com/intel/daal.git
3. # go to the scala spark directory
4. cd daal/samples/scala/spark

Download the latest DAAL release and customize your environment.

1. url="https://github.com/intel/daal/releases/download/2019\_u4/l\_daal\_oss\_p\_2019.4.007.tgz" # insert link to last daal release here
2. name="l\_daal\_oss\_p\_2019.4.007"
3. # download and unzip daal release
4. wget $url
5. tar -xzf "${name}.tgz"
6. # download daal from the main repository
7. git clone https://github.com/intel/daal.git
8. # create a working directory
9. mkdir daal\_release\_lnx
10. mv $name/daal\_prebuild/linux/\* daal\_release\_lnx/
11. mv daal/samples daal\_release\_lnx/daal/
13. cd daal\_release\_lnx/daal
14. # setting up the build environment
15. source bin/daalvars.sh intel64
17. cd samples/scala/spark

**Step 3: configurating tbb threads**

DAAL uses TBB to achieve high parallel performance. TBB is a widely used C++ library for shared memory parallel programming and heterogeneous computing (intra-node distributed memory programming).

In the case of spark within each spark executors, the machine learning algorithms are computed using TBB. Unfortunately, TBB threads in spark executors are currently not configurable. This means that you need to set them yourself. You can do this as follows

1. sudo find / -name spark-defaults.conf

if the cluster is started on aws emr, then the path should look something like this:

/etc/spark/conf.dist/spark-defaults.conf

**Caution, you need sudo to edit the file.**

And in the found file you need to add:

1. spark.task.cpus 8

**Step 4: starting test samples**

In order to verify that everything works, you can run tests, using launcher from attachment.

Firstly, remove default launcher:

1. rm launcher.sh

Instead of the removed launcher take the launcher from <https://raw.githubusercontent.com/intel/daal/2019_u4/samples/scala/spark/launcher.sh>.

It corresponds to the DAAL prebuild, that used in this article.

Make the launcher executable.

1. chmod 744 launcher.sh

Start samples:

1. ./launcher.sh intel64

This will run the tests on small test data.

When running the tests, the warning cat: /etc/alternatives/jre/release: No such file or directory may be displayed. This warning is triggered by scalac and can be ignored. It does not affect the result.

So, if all the tests were successful, you can proceed to write a test case.

Consider launching DAAL on Spark using the example of PCA.

**Step 1: adding test data**

Add test data.

Firstly, make sure, that you are in daal/samples/scala/spark directory.

If the tests were run, then the test data should already be in hdfs.

If not, then you can put test data on hdfs with the following command.

1. export sample=PCA
2. hadoop fs -mkdir -p /Spark/${sample}/data
3. hadoop fs -put ./data/${sample}\*.txt /Spark/${sample}/data/

**Step 2: test file creating**

Create a test file PcaTest.scala.

Lets import the necessary modules.

1. **package** DAAL
3. **import** org.apache.spark.SparkConf
4. **import** org.apache.spark.SparkContext
5. **import** daal\_for\_mllib.{PCA, PCAModel}
7. **import** org.apache.spark.mllib.linalg.Vectors
8. **import** org.apache.spark.mllib.linalg.Matrix
9. **import** org.apache.spark.mllib.linalg.distributed.RowMatrix
11. **import** java.io.\_

In the PcaTest.scala file, add a PCA call.

1. object PcaTest **extends** App {
2. val conf = **new** SparkConf().setAppName("Spark PCA")
3. val sc = **new** SparkContext(conf)
4. // Retrieve input data from TXT and load it into RDD
5. val data = sc.textFile("/Spark/PCA/data/PCA.txt")
6. val dataRDD = data.map(s => Vectors.dense(s.split(' ').map(\_.toDouble))).cache()
7. // Compute PCA decomposition with 10 principal components
8. val model = **new** PCA(10).fit(dataRDD)
9. // Print resulting eigenvectors, eigenvalues and the explained variance
10. println("Eigen vectors: " + model.pc.toString())
11. println("")
12. println("Eigen values: " + model.explainedVariance.toString())
13. sc.stop()
14. }

**Step 3: setting up environment**

Set up the environment.

1. export daal\_ia=intel64
2. export CLASSPATH=${SCALA\_JARS}:$CLASSPATH
3. export SHAREDLIBS=${DAALROOT}/lib/${daal\_ia}\_lin/libJavaAPI.so,${DAALROOT}/../tbb/lib/${daal\_ia}\_lin/gcc4.4/libtbb.so.2,${DAALROOT}/../tbb/lib/${daal\_ia}\_lin/gcc4.4/libtbbmalloc.so.2

**Step 4: building test project**

Build a project.

1. scalac -d PcaTest.jar PcaTest.scala <path-to-daal-spark-samples>/sources/PCA.scala

When running the tests, the warning cat: /etc/alternatives/jre/release: No such file or directory may be displayed. This warning is triggered by scalac and can be ignored. It does not affect the result.

**Step 5: launching test project**

Launch the project.

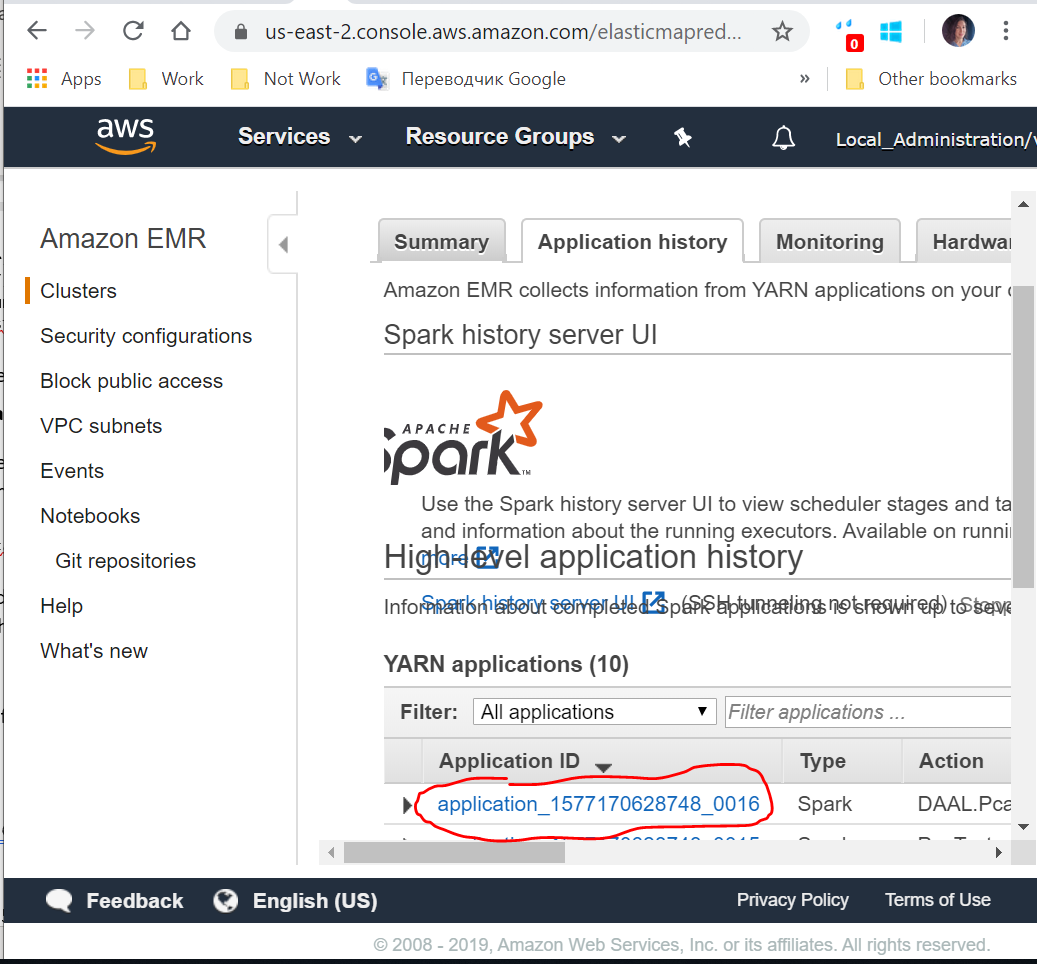
1. spark-submit --driver-**class**-path \"${DAALROOT}/lib/daal.jar:${SCALA\_JARS}\" --jars ${DAALROOT}/lib/daal.jar --files ${SHAREDLIBS} -v --master yarn-cluster --deploy-mode cluster --num-executors 12 --executor-cores 8 --executor-memory 15G --**class** DAAL.PcaTest PcaTest.jar

Description of the num-executors and executor-cores parameters and their correct settings described later in **How to configure a spark in correct way** chapter**.**

If everything is ok, then the message: final **status: SUCCEEDED** will be displayed in the spark logs, and you can get eigen values ​​and vectors using the command:

1. yarn logs -applicationId <application\_id> > pca\_logs.txt

Application id can be found from the list of applications, which can be obtained in Spark UI or application history tab, if the cluster was deployed on AWS EMR.



It will create pca\_logs text file, where eigen vectors and eigen values can be founded. Logs will look like this:

Eigen vectors:

0.25798164372921656 -0.4292620757958274 ... (10 total)

Eigen values:

[0.3779671568979385,0.1750187552716385,0.12002756104610995,…

**How to configure a spark in correct way**

**Num of executors and executors cores**

number of executors and cores per executor can be set with next conditions:

All cores of cluster >= (number of executors) × (cores per executor)

(Number of executors) / (Number of nodes) is integer (all nodes will have same amount of executors)

For example, if you have 5 nodes with 72 cores per node, you might set 20 executors with 18 cores per executor (it’s mostly better when number of nodes < number of executors).

In my case I have 6 nodes with 16 cores per node.

**Memory per executor**

In general the memory per executor can be calculated by the formula:

Total\_ram\_memory\_per\_node/num\_executors

However small overhead memory is also needed to determine the full memory request to YARN for each executor.

The formula for that overhead is 0.07 \* spark.executor.memory

Thus, the memory per executor can be calculated by the formula

Total\_memory/num\_executors -(Total ram per node memory/num\_executors\*0.07)

**Using Different Garbage Collector**

To improve performance, you can use different garbage collector. In the case of daal, G1 GC gave the best performance.

As experiments show, better performance can be achieved using the following g1 garbage collector settings:

* XX:MaxGCPauseMillis=50
* XX:ParallelGCThreads=node\_cores

To use it, add the argument conf:

1. spark-submit --driver-**class**-path "${DAALROOT}/lib/daal.jar:${SCALA\_JARS}" --jars ${DAALROOT}/lib/daal.jar --files ${SHAREDLIBS} -v --conf "spark.executor.extraJavaOptions=-XX:+UseG1GC -XX:MaxGCPauseMillis=50 -XX:ParallelGCThreads=16" --master yarn-cluster --deploy-mode cluster --num-executors 12 --executor-cores 8 --executor-memory 15G --**class** DAAL.PcaTest PcaTest.jar