Name: WILBER HERNANDEZ

UCID: wh3

Preferred email address: [wh3@njit.edu](mailto:wh3@njit.edu)

OPTION# 6

# **AWS EMR - ENVIRONMENT CONFIGURATION DETAILS**

AWS EMR - Release label:emr-5.7.0

Spark: Spark 2.1.1

Hadoop 2.7.3

YARN

Ganglia 3.7.2

Zeppelin 0.7.2

# **PROGRAMMING LANGUAGE**

Scala

# **CLUSTER**

1-master

2-Core

# **CLUSTER NAME IN AWS EMR**

recommendation

# **INSTANCE TYPE**

m4.large

4 vCPU, 8 GiB memory, EBS only storage

EBS Storage:32 GiB

# **SSH CONNECTION INFORMATION**

hadoop@ec2-52-15-188-233.us-east-2.compute.amazonaws.com

# **DATASET SOURCE**

<https://grouplens.org/datasets/movielens/>

MovieLens 1M Dataset

Stable benchmark dataset. 1 million ratings from 6000 users on 4000 movies. Released 2/2003.

# **STEPS**

1-packaged Scala code into Java JAR file using SBT

2-upload dataset to AWS S3 bucket (s3://movierecommendation/) for easy access by EMR cluster

Install sbt on the cluster by,

curl https://bintray.com/sbt/rpm/rpm | sudo tee /etc/yum.repos.d/bintray-sbt-rpm.repo

sudo yum install sbt

(see instructions on <http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Linux.html>)

Create file: movierecommendation.sbt

To help complile the Scala code into a JAR file, http://spark.apache.org/docs/2.1.1/quick-start.html

name := "movierecommendation"

version := "1.0"

scalaVersion := "2.11.7"

libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.1"

libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.1.1"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.1.1"

3-Create this directory structure in the AWS EMR cluster with the files in them,

[hadoop@ip-172-31-43-135 SparkAWS]$ find .

.

./movierecommendation.sbt

./src

./src/main

./src/main/scala

./src/main/scala/movierecommendation.scala

[hadoop@ip-172-31-43-135 SparkAWS]$

4-run the sbt package command at the cli that will compile Scala code down to Java bytecode and will pull all the dependencies from Apache Spark (ex: MLLIB). When done, it will create the JAR file of the Scala code. When done creating the JAR file, the CLI output will look like –

[success] Total time: 10 s, completed Jul 29, 2017 9:08:37 PM

[hadoop@ip-172-31-43-135 SparkAWS]$ ls

movierecommendation.sbt project src target

[hadoop@ip-172-31-43-135 SparkAWS]$ cd target/

[hadoop@ip-172-31-43-135 target]$ ls

scala-2.11 streams

[hadoop@ip-172-31-43-135 target]$ cd scala-2.11/

[hadoop@ip-172-31-43-135 scala-2.11]$ ls

classes movierecommendation\_2.11-1.0.jar resolution-cache

[hadoop@ip-172-31-43-135 scala-2.11]$ ^C

[hadoop@ip-172-31-43-135 scala-2.11]$

The JAR file will be needed to run code in the cluster.

5-Load the dataset into the AWS S3 bucket (s3://movierecommendation/))

# **HOW TO ACCESS AWS EMR CLUSTER AND RUN SCRIP:**

1-Connect via SSH to,

hadoop@ec2-52-15-188-233.us-east-2.compute.amazonaws.com

**Use the PK file embedded on this document below to connect via SSH (note: no login username needed)**



2-Type the following to run the script to the movie recommendation output

spark-submit movierecommendation\_2.11-1.0.jar

This output is based on the fictitious user whose preference or trained data is (see Scala code):

Rating(0,260,5), //Star Wars

Rating(0,329,5), //Star Trek Gens

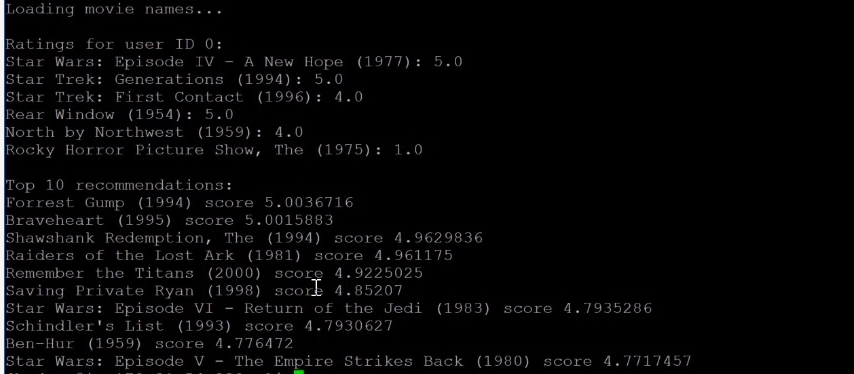
Rating(0,1356,4), //Star Trek First Cont

Rating(0,904,5), //Rear Wind

Rating(0,908,4), //North by NW

Rating(0,2657,1) //Rocky Picture Show

3- the output of this script will look like,



# **SCALA CODE**

package com.sparkaws

import org.apache.spark.ml.recommendation.ALS

import org.apache.spark.sql.SparkSession

import scala.io.Source

import java.nio.charset.CodingErrorAction

import scala.io.Codec

import org.apache.spark.sql.functions.\_

class Movierecommendation {

//Rating class

case class Rating(userId: Int, movieId: Int, rating: Float)

//Passing a line of movies.dat to Rating

def parseRating(str: String): Rating = {

val fields = str.split("::")

assert(fields.size == 4)

Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat)

}

//Load map of movie IDs to movie names

//From disk to memory

def loadMovieNames() : Map[Int, String] = {

//Handle character encoding issues:

implicit val codec = Codec("UTF-8")

codec.onMalformedInput(CodingErrorAction.REPLACE)

codec.onUnmappableCharacter(CodingErrorAction.REPLACE)

//Create map of Ints to Strings, to populate from movies.dat

var movieNames:Map[Int, String] = Map()

//Update movies.dat if stored someplace other than current directory

val lines = Source.fromFile("/home/hadoop/movies.dat").getLines()

for (line <- lines) {

var fields = line.split("::")

if (fields.length > 1) {

movieNames += (fields(0).toInt -> fields(1))

}

}

return movieNames

}

def main(args: Array[String]) {

//Set SparkSession

val spark = SparkSession.builder.appName("ALSExample").getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

//print map of movie ID's to movie names in memory

println("Loading movie names...")

val nameDict = loadMovieNames()

import spark.implicits.\_

// s3://movierecommendation/ml-1m/ratings.dat

//load ratings or large data to train recommendation model with

val ratings = spark.read.textFile("s3://movierecommendation/ml-1m/ratings.dat").map(parseRating)

//count number of ratings each movie for later use

val ratingCounts = ratings.groupBy("movieId").count()

//apply ALS or Alternating Least Squares recommender with fictitious user parameters

val als = new ALS().setRank(8).setMaxIter(10).setRegParam(0.1).setSeed(1234).setUserCol("userId").setItemCol("movieId").setRatingCol("rating")

//making a new user id who likes scifi and classics

val newUserRatings = Array(

Rating(0,260,5), //Star Wars

Rating(0,329,5), //Star Trek Gens

Rating(0,1356,4), //Star Trek First Cont

Rating(0,904,5), //Rear Wind

Rating(0,908,4), //North by NW

Rating(0,2657,1) //Rocky Picture Show

)

val newUserRatingsDS = spark.sparkContext.parallelize(newUserRatings).toDS()

//Add user into ratings to train ALS algorithm

val allRatings = ratings.union(newUserRatingsDS)

//Train ALS recommender model

val model = als.fit(allRatings)

//Build dataset of movies fictitious user has not seen

//that have been rated more than 25 times

val moviesIveSeen = newUserRatings.map(x => x.movieId)

val unratedMovies = ratings.filter(x => !(moviesIveSeen contains x.movieId))

val myUnratedMovies = unratedMovies.map(x => Rating(0, x.movieId, 0)).distinct()

val myUnratedMoviesWithCounts = myUnratedMovies.join(ratingCounts, "movieId")

val myPopularUnratedMovies = myUnratedMoviesWithCounts.filter(myUnratedMoviesWithCounts("count") > 25)

//Predict ratings on each movie

val predictions = model.transform(myPopularUnratedMovies)

//Print ratings for the user with movie titles

println("\nRatings for fictitious user or user ID 0:")

for (rating <- newUserRatings) {

println(nameDict(rating.movieId) + ": " + rating.rating)

}

//taking 10 movies with highest rating predictions and print them

println("\nTop 10 recommended movies:")

for (recommendation <- predictions.orderBy(desc("prediction")).take(10)) {

println( nameDict(recommendation.getAs[Int]("movieId"))

+ " score " + recommendation.getAs[String]("prediction"))

}

//Stop SPARK session

spark.stop()

}

}