ISOM 3370 FINAL PROJECT

GROUP MEMBER: CHAN, POK WAH, QIN SHENGHAO, ZHANG YICHEN

PARALLEL IMPLEMENTATION

1) Create 2 instances with: ami-49cb5931, with spark installed

master

worker1 --- for HDFS, Spark

1.1) switch user

$ sudo su - bigdata

2) Set Spark Configuration

Copy the Configuration template

$ cp /home/bigdata/spark/conf/slaves.template /home/bigdata/spark/conf/slaves

Modify the file

$ vim /home/bigdata/spark/conf/slaves

change the content "localhost" to:

worker1

3) Set the hostname (sshconf.sh)

$ ./sshconf.sh

Set the private IP for master and worker

4) start spark daemons

$ spark/sbin/start-all.sh

4.1) Please check whether spark "worker" daemon is running:

$ ssh worker1

In worker1:

$ jps

=>>> worker

if no, run:

$ start-slave.sh spark://master:7077

check again:

$ jps

then exit back to master

5) set HDFS configuration

$ vim $HADOOP_CONF_DIR/workers

change the content "localhost" to:

worker1

$ hdfs namenode -format

$ start-dfs.sh

Upload datafiles:

Use WinSCP to send ratings.csv to instance on AWS under ~/examples/pagerank/data

Remove the header of ratings.csv for future processing

$ sed 1d ratings.csv > noheader.csv

$ hadoop fs -mkdir -p /user/yzhangec

$ hadoop fs -copyFromLocal ~/examples/pagerank/data /user/yzhangec

$ hadoop fs -ls /user/yzhangec/data

6) Run spark-shell

$ spark-shell --master spark://master:7077 --executor-memory 512m

7) In spark-shell:

The focusUser variable will be changed every time for the specific user. (i.e. 3, 9, 33, 39, 90)

```
val file = sc.textFile("hdfs://master:9000/user/yzhangec/data/ noheader.csv");
val fileData = file.map(_.split(",")).map(t => (t(0).toInt, t(1).toInt, t(2).toDouble));
val likeUM = fileData.filter(x => x._3 > 3).map(t => (t._1, t._2)).groupByKey;
val likeMU = fileData.filter(x => x._3 > 3).map(t => (t._2, t._1)).groupByKey;

var focusUser = 3; //this will be changed every time for specific user's output
var relevance = fileData.map(data => data._2).distinct.map(temp => (temp, 1.0 / 9125));

var conMU = likeMU.join(relevance).flatMap {case (movieID, (userID, relevance)) => userID.map(dest
=> (dest, relevance / userID.size))};
```

```
var similarity = conMU.reduceByKey(_ + _).map(x => if(x._1 == focusUser) (x._1, x._2 * 0.8 + 0.2)
else (x._1, x._2 * 0.8));

var conUM = likeUM.join(similarity).flatMap {case (userID, (movieID, similarity)) =>
movieID.map(dest => (dest, similarity / movieID.size))};

//the above part will not actually be computed, they are used to define vars to avoid recursive
defining errors in the recursion

for (i <- 1 to 10) {
    conMU = likeMU.join(relevance).flatMap {
        case (movieID, (userID, relevance)) => userID.map(dest => (dest, relevance / userID.size))
    };
    similarity = conMU.reduceByKey(_ + _).map(x => if(x._1 == focusUser) (x._1, x._2 * 0.8 + 0.2)
else (x._1, x._2 * 0.8));
    conUM = likeUM.join(similarity).flatMap {
        case (userID, (movieID, similarity)) => movieID.map(dest => (dest, similarity /
movieID.size))
    };
    relevance = conUM.reduceByKey(_ + _);
};

val result = relevance.sortBy(x => (x._2, x._1), false).take(20);
sc.parallelize(result).repartition(1).saveAsTextFile("hdfs://master:9000/user/yzhangec/output/out_
3");
```

8) Get the result from Hadoop:

$ hadoop fs -get hdfs://master:9000/user/yzhangec/output

ALTERNATIVE IMPLEMENTATION BY PYSPARK

We also implement the PageRank Algorithm by PySpark. The detailed code is attached in file ./PySpark/MovieRecommendation.py

We use pandas package to deal with csv file data.

The data are processed with RDD approach.

The output is the recommended movies for all users.