Graph Processing on Spark

Description

The purpose of this project is to develop a graph analysis program using Apache Spark.

This project must be done individually. No copying is permitted. **Note: We will use a system for detecting software plagiarism, called**[**Moss (Links to an external site.)**](http://theory.stanford.edu/~aiken/moss/)**, which is an automatic system for determining the similarity of programs.** That is, your program will be compared with the programs of the other students in class as well as with the programs submitted in previous years. This program will find similarities even if you rename variables, move code, change code structure, etc.

Note that, if you use a Search Engine to find similar programs on the web, we will find these programs too. So don't do it because you will get caught and you will get an F in the course (this is cheating). Don't look for code to use for your project on the web or from other students (current or past). Just do your project alone using the help given in this project description and from your instructor and GTA only.

Platform

As in the previous projects, you will develop your program on SDSC Comet. Optionally, you may use IntelliJ Idea or Eclipse to help you develop your program, but you should test your programs on Comet before you submit them.

Using your laptop to develop your project

If you'd prefer, you may use your laptop to develop your program and then you would need to test it and run it on Comet.

To install the project:

cd

wget http://lambda.uta.edu/cse6331/project5.tgz

tar xfz project5.tgz

To compile and run project5:

cd project5

mvn install

~/spark-1.5.2-bin-hadoop2.6/bin/spark-submit --class Partition --master local[2] partition.jar small-graph.txt

Setting up your Project on Comet

Login into Comet and download and untar project5:

wget http://lambda.uta.edu/cse6331/project5.tgz

tar xfz project5.tgz

chmod -R g-wrx,o-wrx project5

Project Description

You are asked to re-implement Project #3 (graph partitioning) using Spark and Scala. **Do not use Map-Reduce**. An empty project5/src/main/scala/Partition.scala is provided, as well as scripts to build and run this code on Comet. **You should modify Partition.scala only**. Your main program should take the text file that contains the graph (small-graph.txt or large-graph.txt) as an argument.

The graph can be represented as RDD[ ( Long, Long, List[Long] ) ], where the first Long is the graph node ID, the second Long is the assigned cluster ID (-1 if the node has not been assigned yet), and the List[Long] is the adjacent list (the IDs of the neighbors). Here is the pseudo-code:

var graph = /\* read graph from args(0); the graph cluster ID is set to -1 except for the first 5 nodes \*/

for (i <- 1 to depth)

graph = graph.flatMap{ /\* (1) \*/ }

.reduceByKey(\_ max \_)

.join( graph.map( /\* (2) \*/ ) )

.map{ /\* (3) \*/ }

/\* finally, print the partition sizes \*/

where the mapper function (1) takes a node ( id, cluster, adjacent) in the graph and returns (id,cluster) along with all (x,cluster) for all x in adjacent. Then the join joins the result with the graph (after it is mapped with function (2)). The join returns an RDD of tuples (id,(new,(old,adjacent))) with the new and the old cluster numbers. In function (3) you keep the old cluster of it's not -1, otherwise you use the new.

Note that in the Comet script, both the Spark local and distributed modes use 2 RDD partitions. This means that in order to get 10 centroids, we need to get 5 from each partition (the first 5 of each partition).

You can compile Partition.scala using:

run partition.build

and you can run it in local mode over the small graph using:

sbatch partition.local.run

Your result should be the same as the small-result.txt. You should modify and run your programs in local mode until you get the correct result. After you make sure that your program runs correctly in local mode, you run it in distributed mode using:

sbatch partition.distr.run

This will work on the moderate-sized graph and will print the results to the output. It should be the same as large-solution.txt.

What to Submit

Submit the zipped project5 directory, which must contain the files:

project5/src/main/scala/Partition.scala

project5/partition.local.out

project5/partition.distr.out