HW5 MapReduce

1. Read the MapReduce paper attached, answer the following questions:
   a) The results of mapping are shuffled and send to reducers, which could cost a lot of network traffic? Propose a way to significantly reduce the network traffics.
   b) If a mapper or reducer task is too slow which would make the whole map-reduce task very slow? Propose an approach to address this issue.

2. Implement a program that counts the out-degree of all vertices in a graph using MapReduce, and then find the top-20 biggest out-degree vertices also using MapReduce.
   There are 3 graphs to be count. (case1, case2, case3)

   Note:
   For case1, just find top-2 biggest out-degree vertices.

3. Treating the top-20 vertices as the frontier (sometimes referred to as 'search key'), implement the Breadth-first search(BFS) algorithm using MapReduce. And then output the distance of all vertices that can be searched by the frontier.

   Note:
   BFS: https://en.wikipedia.org/wiki/Breadth-first_search
   When you output the distance, please sort them by the distance in ascending order. The distance of the frontier is 0.
   Please consider the condition of convergence of the BFS algorithm.

Please submit your assignment containing your report, code and the result.
Please describe your solution in detail in your report. Besides, please tell me how to run your program successfully.

Java should be the native language of MapReduce. However, if you don't like Java, you may write MapReduce programs in Python, the following link gives the tutorial on MapReduce in Python:
http://hadoop.apache.org/docs/r1.2.1/streaming.html
http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

The command may be '*hadoop jar /opt/hadoop/jars/hadoop-streaming-2.6.0-cdh5.14.0.jar …*'.

[Graph]

G = (V, E)

V is the set of all vertices in the directed graph G.

E is the set of all edges in the directed graph G.

$e = (u, v)$ $u, v \in V$ $e \in E$

It means there is an edge that from u to v.

out-degree(u) = amount of edges that start from u.

For example, there are 4 edges in a graph: (1,2) (1,3) (2,3) (1,3)
So, out-degree(1) = 3, out-degree(2) = 1, out-degree(3) = 0.
Note: Treat the duplicated edge as another edge.

[Code]

You will be given 2 codes using Java language, one is MyWordCount.java, the other is OutDegree.java.

MyWordCount.java is a completed program that you can compile and run directly, this program counts the words in the input file. You can use it to learn about MapReduce. The output file's format will be "key value" ("word counts" such as "aaa 2") per line. I will show more details about the output in section **Sample**.

OutDegree.java is an uncompleted program that counts the out-degree of vertices in a graph. You will need to fill in 2 functions: map() and reduce(). Besides, in order to find the top 20 biggest out-degree vertices, you may need to add some classes or some jobs aiming to sort the output ("key value") by the value.

For the BFS algorithm, you need to design it all by yourself. Besides, you may need to reorganize the mode or the format of how the graph is stored.

[HDFS & Data]

The MapReduce program need to read the input file from hdfs and write the output file to hdfs.

The graph data is located at the directory:

/user/qinwei/graph_data

There are 5 files in it:

temp.txt. Used as the sample to test MyWordCount.java.
edges.txt. Used as the sample to test OutDegree.java.
case1: |V|=9 (1~9) |E|=100
case2: |V|=999986 (1~999986) |E|=10000000
case3: |V|=9999921 (1~9999921) |E|=100000000

case1 & case2 & case3 are the 3 directed graphs for this assignment.

The format of these 3 files is "a u v w".

It means there is an edge from u to v, and its weight is w.
We use u for part 1 of this assignment.
We use u and v for part 2 of this assignment.

[Sample]

1. MyWordCount:

   Input:

   File: temp.txt, a file consists of 4 lines' word bellowing.

   ```
   aaa
   bbb
   ccc
   aaa
   ```

   Output:

   File: res_wc/part-r-00000

   ```
   aaa  2
   bbb 1
   ccc  1
   ```

2. Graph:

   Input:

   File: edges.txt

   ```
   a 1 2 0
   a 3 4 0
   a 5 1 0
   a 2 4 0
   a 4 5 0
   a 2 5 0
   a 2 3 0
   a 3 2 0
   ```

Output:

```
After part 1 (format: vertex_ID      out-degree):

     2   3
     3   2
     5   1
     4   1
     1   1
```

```
After part 2 using top-2 vertices as the frontier to do BFS
       (format: vertex_ID   distance):

     3   0
     2   0
     5   1
     4   1
     1   2
```