Readme

Liwei Han(lh567), Mengyuan Yang(my437), Yiwen Wang(yw748)

File Structure

Task 1: Simple Computation of Page Rank
Simple: My_counter.java, PageRank.java, TrustMapper.java, TrustReducer.java
Task 2: Blocked Computation of Page Rank
Block: My_counter.java, PageRank.java, TrustMapper.java, TrustReducer.java
Task 3: Jacobi *vs* Gauss-Seidel
JG: My_counter.java, PageRank.java, TrustMapper.java, TrustReducer.java
Task 4: Random Block Partition
Random: My_counter.java, PageRank.java, TrustMapper.java, TrustReducer.java

Input File

1. Prefilter:

filter.txt:
Use netid: lh567; rejectMin = 0.765 * 0.9; rejectLimit = 0.765 * 0.9 + 0.01;
Total number of edges: 1228211

2. Prefiltered and formated file:

filter_format.txt : (input file for the mapreduce)
Each line works as <linenum srcnode pagerank destnodes>. Note that line number, source node, pagerank and destination nodes are separated by a space while within destination nodes, each node is separated by a comma, such like: <0 0 0.1 1,2,3>. If a node has no destination node, it has the form like <linenum srcnode pagerank>, such like: <0 0 0.1>.

Implemention of Solution

1. SimplePageRank:

mapper: the mapper processes the input line one by one and emits key-value pairs in the form of either <nodeID, node> or <neighbournodeID, incomingpagerank>, where node is the information of a whole line with adding a special sign “@”, and incomingpagerank is the pagerank when equally distributed among the source node's outgoing nodes. Therefore incomingpagerank= pagerank/sizeof(outgoing nodes).

reducer: the reducer receives all the key - value pairs corresponding to a given nodeID and it identifies whether the value is imcomingpagerank or node by checking the sign “@”.
Then the reducer computes the page rank of this node by adding all the incoming pagerank. Also it computes the residual value for this node and add it to the hadoop counter - residual; Finally, it emits the key - value pair in the form of <nodeID, node>.

running: The main method is responsible for initiating MapReduce jobs. We run a total of 5 mapreduce passes. The output for each pass i serves as input file for pass i+1. When a job is finished, we get the total residual value from hadoop counters and then compute and print the average residual.

2. BlockedPageRank:

Mapper: There are 3 tasks for the mapper: (a) Assign a blockid to each node; (b) Find the nodes which are belong to BE ; (c) Find the nodes which are belong to BC.
BE = { <u, v> | u B u v } = the Edges from Nodes in Block B
BC = { <u, v, R> | u B v B u v R = PR(u)/deg(u) }

Implementation of task (a): Using an integer array to store all the blocks' range, and find the corresponding blockID for a given nodeID.

Implementation of task (b): For each source node, check the blockID for its destination nodes, if both have the same blockID, then emit <blockID, value>, where value is in the form of : "e" + source nodeID + "/" +destination nodeID. "e" is used to to identify it by reducer.

Implementation of task (c): For each source node, check the blockID for its destination nodes, if their blockIDs are different, then emit <blockID, value>, where blockID is the destination node's blockID and value is in the form of: "c" + source nodeID + "/" + incoming pagerank. "c" is used to identify it by reducer.

Finally, mapper emits the key-value pair of <nodeID, node>.

Reducer:
In the reducer, we totally use five HashMaps and one ArrayList to store all the information we need. The arraylist store all the nodeID of this block and five HashMaps are: hm_c, hm_e, init, old and nev.

hm_c: store all the nodes' information belong to BC. The key is nodeID, the value is an arraylist which involve all the incoming pagerank from other blocks for this node.

hm_e: store all the nodes' information belong to BE. The key is nodeID, the value is an arraylist which store all the nodesIDs that has an outlink to this node.

init: store all the nodes' initial information belong to this block. The key is nodeID, the value is node's information, i.e. nodeID, pagerank, and destination nodes.

old: store this block's nodes information and used in the iteration to get the pagerank value of this node.

nev: store this block's nodes information and used in the iteration to temporarily store the new pagerank value of this node.

The iteration over the block is implemented as follows:
The termination condition is either iterates twenty loops or the average residual of this bloc is below 0.001.

For each nodes in this block, check the hm_c and hm_e and do the computation according to the pseudocode provided by the handout.

Finally, compute the average residual and check whether it meets the termination condition.

After complete the iteration, record the iteration loops in the hadoop counter.

running: The implementation of main method is similar to the simple pagerank. The difference is that instead of using fixed number of iteration to terminate the iteration of job pass, we use the conditon that let average residual value < 0.001. Also, after each job finished, we compute and print the total and average loops of this pass in each reducer, and the average residual value of this pass.

3. RandomPageRank:

The implemention of randomPageRank is almost the same as blockedPageRank. The only difference lies in the way of block partition, that is, for RandomPageRank we call function blockIDofNode(long nodeid).
BlockedPageRank depends on a well defined block range which is immutable and for each input we find the exact blockID according to the nodeid.
Random Partition: The way we locate random blockid is that we find the hashcode of nodeID which is unique and random for every nodeID and have it fall into the range of 68 by mod 68.

4.Jacobi vs Gauss-Seidel:

The overall implementation is similar to the blocked pagerank, except that in the reducer class, we use the new pagerank value of nodes rather than the old one.

Results:

Results are put into results the folder in each project. The results folder contains the screenshot of clusterDetails, output buckets, results and the PageRank values for the two lowest-numbered Nodes in each Block. The PageRank values for the two lowest-numbered Nodes in each Block are saved as txt file.

Conclusion:

1. RandomPageRank and BlockedPageRank: the elapsed time for blocked pagerank is 16 minutes and the number of pass is 7, while the elapsed time for random pagerank is 27 minutes and the number of pass is 21. So we can conclude that the speed of blocked pagerank to converge is faster than random pagerank.

2. Jacobi vs Gauss-Seidel: the elapsed time for blocked pagerank is 16 minutes and the number of pass is 7, while the elapsed time for Gauss-Seidel pagerank is 16 minutes and the number of pass is 6. Although the time for these two algorithm is similar, in the first Gauss-Seidel's pass we can see that average loops(10) is smaller than the BlockedPageRank's pass(16).