CS ASSIGNMENT - 3
Satya Akhil Chowdary Kuchipudi
Section-1

Pre-Processing :

Pre-Processor uses the Parser provided by Joe. I just modified into map reduce program for the job chaining.

```
class PreProcessor {

        map(offset, line)
                emit(line.key, Bz2wikiparser(line))
}
```

No Reducer class for this program, since Mapper's output alone is sufficient for adjacency nodes calculation.

Classification :

Primary Nodes : Nodes that have outlines and are traversed in pagerank algorithm
Primary Dangling Nodes : Nodes that are traversed in pagerank algorithm and doesn't have any outlinks
Secondary Dangling Nodes : Nodes that are present in adjacency list but are not traversed yet in pagerank

A : [B, C, D]
B: []

Primary Nodes : A
Primary Dangling Nodes : B
Secondary Dangling Nodes : C, D

Page Rank Pseudo Code :

The pseudo code I implemented is bit varied from pseudo code discussed in the class, since we need to handle the dangling factor from primary dangling and secondary dangling nodes.

```
// Enumeration Counters for storing Total Nodes and Dangling Loss at each iteration
enum counters {
dangling_loss,
total_nodes
}


map (nid n, Node N) {

	// First Iteration, page rank is set to 1/V
	if(first iteration) {
		pagerank = 1/total_nodes;
	}
	// Remaining Iterations , pagerank is calculated from fie
	else {
		pagerank = N.pagerank;
		pagerank += dangling_loss;
	}

//  If primary dangling node, then emit directly, since no adjacency nodes, pagerank will be set
accordingly with boolean true to recognize primary and secondary.
if(N.adjacency_nodes_list is null)
	emit (nid, Node NewNode (pagerank, String[], true);
// Primary Nodes, containing adjacency list
else
	// Emit Node to construct graph at Reducer
	emit (Node NewNode (pagerank, N.adjacency_node_list, true))
	// Emit contribution to each adjacency node
	for each id in N.adjacency_node_list
		emit (nid, Node NewNode(pagerank/|adjacency_node_list, String[], false))
}


reduce(nid, [n1, n2, n3, n4 ….]) {

	totalpagerank = 0;
	new node = null;
```

```
        for all n in [n1, n2, n3 …] {
                // If primary node, then new node will be set, otherwise it will be null to
                differentiate primary and secondary nodes
                if(primary_node is true) {
                        newnode = n;
                }
                // keep updating total contributions to this current node
                else {
                        totalpagerank += n.pagerank ;
                }
        }

        // update pagerank to include dampening factor for random jump probability.
        dampening_factor = 0.15
        totalpagerank = dampening_factor/total_nodes + (1-dampening_factor)*totalpagerank;

        // Check for primary node or not
        if(node not null){
                // if no adjacency list, add to dangling loss the current pagerank for next iteration
                if(node.adjacency_node_list is null) {
                        update dangling_loss += node.pagerank;
                }
                //  Emit the primary node with updated fields and total pagerank
                emit(nid, totalpagerank + node.adjacency_node_list)
        }
        // if secondary node, update the contributions to dangling loss and skip writing the node
        else {
        update dangling_loss += totalpagerank
        }
}
```

Top K:

```
PageRankNode(nid, pagerank) {
        nid = nid,
        pagerank = pagerank
}

compareto(newnode){
        return Double.comapre(newnode.pagerank, this.pagerank)
}


new priority_queue,
map(nid, pagerank) {

        priority_queue.offer(new PageRankNode(nid, pagerank));
```

```
}

cleanup(){
        while (i < 100)
                priority_queue.poll();
                emit(null, nid+pagerank)
}

keysortcomparator(key) {
}

reduce(null, [p1, p2, p3 …]) {
        for(p in [p1, p2, p3 …] and count <100) {
                emit (p1.nid, p1.pagerank)
                count++
        }
}
```

Map Shuffle bytes :

Iteration 1 : 1718564125
Iteration 2 : 2180996160
Iteration 3 : 2182225331
Iteration 4 : 2182776901
Iteration 5 : 2182578674
Iteration 6 : 2183100785
Iteration 7 : 2183010668
Iteration 8 : 2183229125
Iteration 9 : 2183039653
Iteration 10 : 2182994439

Reduce HDFS Bytes :

Iteration 1 : 1205475680
Iteration 2 : 1205444304
Iteration 3 : 1205415490
Iteration 4 : 1205413225
Iteration 5 : 1205413763
Iteration 6 : 1205421779
Iteration 7 : 1205414067
Iteration 8 : 1205422014
Iteration 9 : 1205416150
Iteration 10 : 1205417119

Yes, they change over time. The main reason being the values of page ranks keep changing across all iterations. It doesn't stay constant over iterations unless convergence is achieved. Another point to consider is change is not much substantial, it's just minor change due to byte changes between each page rank iterations.

The first iteration's Reduce shuffle is less, considering there is no calculation of dangling loss, happening on mapper phase of first iteration.

Performance Comparison :

1st Configuration Times :

Pre-Processing : 32 min 8 sec
Iteration 1 : 2 min 26 sec
Iteration 2 : 2 min 29 sec
Iteration 3 : 2 min 28 sec
Iteration 4 : 2 min 30 sec
Iteration 5 : 2 min 27 sec
Iteration 6 : 2 min 34 sec
Iteration 7 : 2 min 36 sec
Iteration 8 : 2 min 39 sec
Iteration 9 : 2 min 30 sec
Iteration 10 : 2 min 32 sec
Total Iterations : 25 min 11 sec
Top100 : 55 sec

2nd Configuration Times :

Pre-Processing : 18 min 53 sec
Iteration 1 : 1 min 32 sec
Iteration 2 : 1 min 35 sec
Iteration 3 : 1 min 34 sec
Iteration 4 : 1 min 32 sec
Iteration 5 : 1 min 30 sec
Iteration 6 : 1 min 34 sec
Iteration 7 : 1 min 29 sec
Iteration 8 : 1min 27 sec
Iteration 9 : 1 min 33 sec
Iteration 10 : 1 min 35 sec
Total Iterations : 15 min 21 sec
Top100 : 41 sec

Speedup = time seq/parallel.
Speed-up ratio (config1/config2) for Pre-Processing = 1.777
Speed-up ratio (config1/config2) for PageRank = 1.66
Speed-up ratio (config1/config2) for Top-K = 1.3

Top-K is not that efficient considering at the end all values have to calculated at single reducer. It's limits the capability of parallel computing. On the other hand, Pre-Processor has the highest speed-up ratio among all. The main reason behind this is, its perfect candidate for parallel computing considering none of the parts have any relation with other parts. This avoids shuffling across and avoids the use of reducer. Mapper phase alone is sufficient for Pre-processor. Page-Rank is intermediate, it's parallelism is not as bad as Top-K, but it's not as good as Pre-processor, since lot of shuffling has to happen between mappers and reducers and also different computations are related.

Top-100 Records :

United_States_09d4``0.0032494768828390604
2006``0.0015211616784715051
United_Kingdom_5ad7``0.0014900544362796333
Biography``0.0012453996105379601
2005``0.0011359189621972803
England``0.001093198055542417
Canada``0.0010620751451987727
Geographic_coordinate_system``9.819325719866722E-4
France``8.978060204010888E-4
2004``8.930179746383137E-4
Australia``8.4458563765572E-4
Germany``8.11060617306622E-4
2003``7.288399602462435E-4
India``7.255692167358884E-4
Japan``7.221949687545183E-4
Internet_Movie_Database_7ea7``6.683844552995707E-4
Europe``6.315251268580295E-4
Record_label``6.231847472798687E-4
2001``6.050657948632988E-4
2002``5.993002507685148E-4
Population_density``5.96006468150725E-4
World_War_II_d045``5.910906435713141E-4
Music_genre``5.869549096587867E-4
2000``5.76872313131371E-4
Italy``5.518090705669867E-4
Wikimedia_Commons_7b57``5.449154509907504E-4
Wiktionary``5.437791258095735E-4
London``5.387350359598977E-4
English_language``5.157695487882644E-4
1999``5.036218787043425E-4
Spain``4.4983467094282216E-4
1998``4.425670554548414E-4
Russia``4.250527392066299E-4
Television``4.197381530982238E-4
1997``4.189525462529053E-4
New_York_City_1428``4.145931960634428E-4
Football_(soccer)``4.10545286137929E-4
Census``4.064766755730564E-4
1996``4.018896987095552E-4
Scotland``4.00045213355645E-4
Square_mile``3.9340291358111484E-4
1995``3.849959196817877E-4
Scientific_classification``3.832626713394041E-4
Population``3.818679693223933E-4

China``3.817483287958019E-4
California``3.747400396608793E-4
1994``3.610791165591881E-4
Record_producer``3.608266963508694E-4
Public_domain``3.5859111065952046E-4
Sweden``3.57584989189876E-4
Film``3.571479044600305E-4
New_Zealand_2311``3.51568321292626E-4
United_States_Census_Bureau_2c85``3.482010208106093E-4
New_York_3da4``3.466865093415732E-4
Marriage``3.4573413119541574E-4
Netherlands``3.422356575574636E-4
1993``3.4153676825714993E-4
Studio_album``3.4036924335538514E-4
Politician``3.3811473534645683E-4
1991``3.3762470335418524E-4
1990``3.332995087661662E-4
Album``3.3085981112225484E-4
1992``3.307204812964279E-4
Per_capita_income``3.276864360797222E-4
Actor``3.2440916020703666E-4
Latin``3.206917880294077E-4
Ireland``3.2054726678189147E-4
Poverty_line``3.1428098804422424E-4
1989``3.0661841285004157E-4
Norway``3.003940964978517E-4
Website``2.973075792858006E-4
1980``2.9240794280670945E-4
Area``2.909950210718015E-4
Personal_name``2.886046807014996E-4
Animal``2.884647037837323E-4
1986``2.822230934383184E-4
Poland``2.8185843785508353E-4
Brazil``2.816551521605622E-4
1985``2.7848382804162864E-4
1987``2.7772202233817395E-4
1983``2.7565830209025263E-4
1982``2.750011827858954E-4
1981``2.726916160815193E-4
1979``2.725532226823511E-4
1984``2.72099436913142E-4
1988``2.717854453214809E-4
1974``2.7063394170145113E-4
World_War_I_9429``2.705317433735057E-4
French_language``2.704607315922989E-4
Paris``2.7033011702352417E-4
Mexico``2.6786642008563903E-4
1970``2.623263733721317E-4
USA_f75d``2.6226497351249523E-4

19th_century``2.6094263063126537E-4
January_1``2.5990639808903555E-4
1975``2.5903109822293896E-4
1976``2.589160435322116E-4
White_(U.S._Census)_c45a``2.587514622617928E-4
Africa``2.580087958943384E-4
South_Africa_1287``2.57229261489965E-4