CS6240-SHUBHAM DEB

	Runtime on 6 m/c	Runtime on 11 m/c
Pre-Processing	2192877	1276559
Pagerank	1749635	1443167
Top-100	53565	38592

As expected the configuration with 11 machines will process all the operations faster than the configuration with 6 machines because of more parallelism.

1) Pre-processing
   Speedup = 2192877/1276559 = 1.71
2) Pagerank
   Speedup = 1749635/1443167 = 1.2123
3) Top-100
   Speedup = 53565/38592 = 1.38

Pre-processing shows faster speedup which shows good parallelism among the machines.
Pagerank also shows good speedup as we have used pagename as the key and the node class which consists of adjacency list as well as the pagerank in it which helped to easily retrieve previous pagerank as well as the list info from the file.
Top-100 is also faster as I am using the pagerank itself as the key to sort the top 100 pages in descending order.

	Data transferred from	Data transferred from

	Mapper to Reducer	Reducer to HDFS
Pagerank iteration 1	3236356314	3178539
Pagerank iteration 2	3236356314	3178539
Pagerank iteration 3	3236356314	3178539
Pagerank iteration 4	3236356314	3178539
Pagerank iteration 5	3236356314	3178539
Pagerank iteration 6	3236356314	3178539
Pagerank iteration 7	3236356314	3178539
Pagerank iteration 8	3236356314	3178539
Pagerank iteration 9	3236356314	3178539
Pagerank iteration 10	3236356314	3178539

As we see in the above table, the data transferred between the mapper and the reducer would remain the same. This is because in the mapper we just send the same set of data every time in each iteration:

In the mapper I just send two types of data:

1) The original node and it’s adjacency list and pagerank stored in GraphNode class.

2) The neighbors of the original node along with the contribution of the current node to the neighbors’ node.

Also since the data transferred between reducer and HDFS will remain the same because as I presume that since pagename is the key, it will only write unique pagenames to HDFS which will never change after each iteration as it is constant.

Pseudo Code for Parser

```
public void map(Key k,Value v){
	// Since the bz2 file is in the format pagename : full contents of the	//
page, we need to split on the delimiter ":"
	int delimiterIndex = value.indexOf[":"]
	page = value.substring(0,delimiterIndex)
	// Here we parse the links using WikiParser to remove any html  //
filenames which contain '~', we also remove the path name and
	// the .html suffix in the link
	// I modified the parser by removing the page if it is in the links
	// and also I added "&amp" instead of "&" to make it XML
	// compatible as "&" is an invalid XML
	links  = parse(value.substring(delimiterIndex+1))
	emit(page,links)
}

// I made a GraphNode class which has pagerank, list of outlinks, pagerank //
contribution from other nodes and a boolean saying whether the node is
// dangling or not
public void reduce(Key page, List<Links>){
	GraphNode node;
	dangling_nodes ← new Global Counter
	nodes ← new Global Counter

	boolean isDangling=false;
	if(links.size()==0) {
		dangling_nodes.increment(1);
```

```
            isDangling = true;
        }
        nodes.increment(1);

        node.setOutlinks(links);
        node.setPagerank(0);
        node.setIsDangling(isDangling);

        emit(page,node)
}
// We write it to the file in SequenceOutputFormat way so that I could
// write data in the form of key,value pairs
```

Pseudo code for Pagerank

```
// We run the pagerank job 10 times and we maintain a global counter
// "iteration"
class Mapper(Text, GraphNode,Text,GraphNode){
        public void setup(){
                iteration = conf.getInt("iteration");
                pagecount = conf.getDouble("nodes");
        }
        // Here key is the pagename and value contains the adjacency list
        // as well as the pagerank value of the pagename
        public void map(Text key, GraphNode value){
                if(iteration == 0){
                        value.setPagerank((double)1/pagecount);
                }
```

```
        GraphNode neigborNode;
        adj_size = value.getOutlinks().size();
        // we get the contribution of this node’s pagerank to all the
    // adjacent nodes in the graph
        for each link in the adjacency list of value:{
             neigborNode.setPagerankContribution(value
                                         .getPagerank()/adj_size);
            // we set isdangling true because initially they have no
            // outlinks
             neigborNode.setDangling(true);
            emit(link,neigborNode);
        }
        // we also emit the current pagename and it’s adjacency list to
        // the reducer
        emit(key,value);
    }
}

class Reducer(Text, GraphNode,Text,GraphNode){
    public void setup(){
        alpha = 0.15;
        iteration = conf.getInt(“iteration”);
        pagecount = conf.getDouble(“nodes”);
        // we define danglingNodeSum as a global counter
        danglingNodeSum = new Global counter;
        // if it is the initial iteration then it is 1/pages else it
        // will have the same formula of pagerank
```

```
        if(iteration == 0){
         pagerank_calc1 = 1/pagecount;
        }else{
        pagerank_calc1 = (alpha/pagecount)+(1-alpha)
                                *(danglingNodeSum/pagecount)
        }
}
public void reduce(Text key, Iterable<Graphnode1, GraphNode2...>){
        GraphNode gn;
        double pr;
        if(iteration == 0){
        // If it is the initial iteration, we don't need to calculate the
        // neighboring contributions so we just emit the node with the
        // pagerank as pagerank_calc1
        for each graphNode in the list of GraphNodes{
        // if the node is not a dangling node then set the outlinks
                if(!node.isDangling())
                        gn.setOutlinks(graphNode.getOutlinks());
        }

        // If it is the dangling node then we increment the dangling
        // node Pagerank sum
        if(gn.getOutlinks().size()==0)
                danglingNodeSum.increment(pr);
        // emit the node with the pagerank
        context.write(key,gn);
        }
```

```
            // For the other iterations we need to calculate the contributions
            // from other nodes and incorporate into the node’s pagerank
            else{
            // For each neighbor nodes we calculate the pagerank
           // contribution for each of those neighboring nodes
            for each graphNode in the list of GraphNodes{
                if(node.isDangling())
                    pr+=node.getPagerankContribution();
                else
                    gn.setOutlinks(graphNode.getOutlinks());
            }

            pagerank_calc2 =  pagerank_calc1 + (1-alpha)*pr;
            gn.setPageRank(pagerank_calc2);

            // if there are no outlinks we increment the danglingNodeSum
            if(gn.getOutlinks().size()==0)
                danglingNodeSum.increment(pr);

            context.write(key,gn);
          }
      }
}
```

Pseudo code for Top 100 pages

```
// Here we get all the pages with their respective pageranks as the input
```

```
public void map(Text key, GraphNode value){
        context.write(value.getPagerank(),key);
}

// Here since we are comparing pageranks, we keep the value of the
// pagerank as the key and the pagename as the value.
// But since we want the higher pageranks to come before, hence we
// implement a comparator

public void SortKeyComaparator(WritableComparable w1,WritableComparable
w2){
        return -1*(DoubleWritable)w1.compareTo((DoubleWritable)w2)
}

// since the pages are sorted by higher pageranks first, we output first 100  //
pages
class Reducer(DoubleWritable,Text,Text,DoubleWritable){
        int count = 1;
        public void reduce(DoubleWritable pr, Text key){
                while(count<=100){
                        context.write(key,pr);
                        i++;
                }
        }
}
```

LOCAL OUTPUT ON THE SIMPLE DATASET

1. United_States_09d4 0.00518721422549509
2. Wikimedia_Commons_7b57 0.004812413056087831
3. Country 0.003940635577847263
4. England 0.002756085830130238
5. Water 0.002685833170094715
6. Animal 0.002558192999367829
7. City 0.002512027634653502
8. United_Kingdom_5ad7 0.0023609725698112015
9. Germany 0.0023539436809666492
10. France 0.0023258816027411163
11. Earth 0.002319406858828818
12. Europe 0.002039363698098507
13. Wiktionary 0.0017574554185906706
14. English_language 0.001751737305517988
15. Government 0.001733348760395847
16. Computer 0.0017212965029798236
17. India 0.001712322508075945
18. Money 0.0016696359604339512
19. Japan 0.0015535074526837227
20. Plant 0.0015250917591044884
21. Italy 0.0015100500021195955
22. Canada 0.0014813746020729986
23. Spain 0.0014733985120154924
24. Food 0.0014262499290244696
25. Human 0.0014134928656453938
26. China 0.0013981070048756962
27. People 0.0013829302380043062

28. Australia 0.0013298278739984981

29. Asia 0.0012847640476336686

30. Capital_(city) 0.0012727199184232475

31. Television 0.0012680052338904409

32. Sun 0.0012549000241686367

33. Number 0.001241449861642872

34. State 0.0012401206507875891

35. Sound 0.0012378737894678075

36. Science 0.0012328602878170828

37. Mathematics 0.00123120280146496

38. Metal 0.0011911417771052531

39. 2004 0.00117480218091587З

40. Year 0.001173253461332767

41. Language 0.0011524014298679365

42. Russia 0.001146887054291973

43. Wikipedia 0.0011266202198632737

44. Religion 0.0011006540230750705

45. 19th_century 0.001097383718437955

46. Music 0.001091501796057004

47. Scotland 0.0010559529039673846

48. 20th_century 0.001054567359595846

49. Greece 0.001050515667020682

50. Latin 0.0010293581381843541

51. London 0.0010291771059294172

52. Greek_language 0.0010048749457338304

53. Energy 9.984289981905348E-4

54. World 9.868788493346706E-4

55. Centuries 9.771864771443005E-4
56. Culture 9.464525310446929E-4
57. History 9.373164705752353E-4
58. Liquid 9.129362615278531E-4
59. Netherlands 9.067027623807238E-4
60. Society 9.020811901364349E-4
61. Planet 9.011135535211027E-4
62. Light 9.008174520740959E-4
63. Wikimedia_Foundation_83d9 8.904386225730832E-4
64. Image 8.89414885691155E-4
65. Scientist 8.881650404557454E-4
66. Law 8.877171632864666E-4
67. Atom 8.855615355257471E-4
68. List_of_decades 8.795217406794E-4
69. Geography 8.792454292219923E-4
70. Uniform_Resource_Locator_1b4e 8.638164602742655E-4
71. Africa 8.598232128519124E-4
72. Turkey 8.452993560118422E-4
73. Inhabitant 8.319995841237841E-4
74. Capital_city 8.235485183638774E-4
75. Plural 8.221013519066274E-4
76. Electricity 8.127619080649534E-4
77. Poland 7.980292852575586E-4
78. Building 7.973488856814573E-4
79. Car 7.950745411464629E-4
80. Book 7.932027739159083E-4
81. Sweden 7.919163982334316E-4

82. Biology 7.886061152818689E-4
83. War 7.716169270295262E-4
84. Chemical_element 7.654796519964684E-4
85. God 7.621778795383236E-4
86. North_America_e7c4 7.558264449649417E-4
87. September_7 7.554605485111293E-4
88. Website 7.486586491137866E-4
89. Nation 7.426011465595062E-4
90. Politics 7.402011114426538E-4
91. Fish 7.335610698889521E-4
92. 2006 7.33315543085793E-4
93. Species 7.32584069228473E-4
94. Mammal 7.232745132439686E-4
95. Portugal 7.178387531326764E-4
96. Island 7.16897752557569E-4
97. Gas 7.133644315308763E-4
98. River 7.11416442799972E-4
99. Switzerland 7.069281457470969E-4
100. World_War_II_d045 7.025321250369043E-4

AWS OUTPUT ON THE FULL DATASET

1. United_States_09d4 0.002614705083135676
2. 2006 0.0012249412139240517
3. United_Kingdom_5ad7 0.0011995653788400816
4. Biography 9.786189840762173E-4
5. 2005 9.143200115993226E-4
6. England 8.774342287387843E-4

7. Canada 8.533839156991101E-4

8. Geographic_coordinate_system 7.690700492103422E-4

9. France 7.223354785013214E-4

10. 2004 7.176963083616064E-4

11. Australia 6.785788089955526E-4

12. Germany 6.521005838120747E-4

13. 2003 5.8570118848838788E-4

14. India 5.814390891608946E-4

15. Japan 5.811268812718626E-4

16. Internet_Movie_Database_7ea7 5.320313074569744E-4

17. Europe 5.07584217393557E-4

18. Record_label 4.903561159595094E-4

19. 2001 4.8558757559244306E-4

20. 2002 4.8147274044554287E-4

21. World_War_II_d045 4.7647065931006504E-4

22. Population_density 4.688402070029833E-4

23. Music_genre 4.6608484650288723E-4

24. 2000 4.6329775591381743E-4

25. Italy 4.438611561193308E-4

26. Wiktionary 4.347541413489517E-4

27. Wikimedia_Commons_7b57 4.3386399036844463E-4

28. London 4.334247649209811E-4

29. English_language 4.1678362765328633E-4

30. 1999 4.0475982262749063E-4

31. Spain 3.6157870457743284E-4

32. 1998 3.5528583677126854E-4

33. Russia 3.42585940753002E-4

34. 1997 3.3630517662944015E-4
35. Television 3.3543662104213877E-4
36. New_York_City_1428 3.336365678428803E-4
37. Football_(soccer) 3.2536867388613643E-4
38. 1996 3.226910571914064E-4
39. Census 3.224790012381231E-4
40. Scotland 3.2113796020123627E-4
41. 1995 3.0924450223409803E-4
42. China 3.074955597495053E-4
43. Population 3.0346343685683393E-4
44. Scientific_classification 3.032208377473414E-4
45. Square_mile 3.0308847558190456E-4
46. California 3.0076298846529957E-4
47. 1994 2.8984942157450155E-4
48. Sweden 2.8664786318534857E-4
49. Public_domain 2.8610401419055316E-4
50. Film 2.8552770886628657E-4
51. Record_producer 2.834468540575585E-4
52. New_Zealand_2311 2.8231620593372E-4
53. New_York_3da4 2.780251410527708E-4
54. Netherlands 2.7566499523399223E-4
55. Marriage 2.749101096980525E-4
56. 1993 2.7399078863460304E-4
57. United_States_Census_Bureau_2c85 2.7359921878439584E-4
58. 1991 2.7108232197041465E-4
59. 1990 2.675202726284365E-4
60. 1992 2.655720329650476E-4

61. Politician 2.638885620006592E-4
62. Album 2.598842722812592E-4
63. Latin 2.592456915144872E-4
64. Actor 2.5758259578692095E-4
65. Ireland 2.572733042699561E-4
66. Per_capita_income 2.548039920362548E-4
67. Studio_album 2.5116220048486404E-4
68. Poverty_line 2.5033653153409916E-4
69. Km² 2.486859428254584E-4
70. 1989 2.461382286001225E-4
71. Norway 2.4004017368145692E-4
72. Website 2.3839000563683385E-4
73. 1980 2.3458035064773754E-4
74. Animal 2.2876079338834563E-4
75. Area 2.2850841314147466E-4
76. 1986 2.2634724651035692E-4
77. Personal_name 2.2546589606197895E-4
78. Poland 2.2528710219180616E-4
79. Brazil 2.249222631409396E-4
80. 1985 2.2335082264643338E-4
81. 1987 2.226445243116869E-4
82. 1983 2.2107899568893987E-4
83. 1982 2.204064679821838E-4
84. 1981 2.1864027023394807E-4
85. 1979 2.1862751556915306E-4
86. French_language 2.1852793546301796E-4
87. 1984 2.1812796761309258E-4

88. 1988 2.1792497900871934E-4

89. World_War_I_9429 2.1792358186184014E-4

90. 1974 2.172876096570386E-4

91. Paris 2.1720422249347392E-4

92. Mexico 2.1498294269582695E-4

93. 19th_century 2.1106113408370387E-4

94. 1970 2.106353217427266E-4

95. USA_f75d 2.1009070613468088E-4

96. January_1 2.1001560345061348E-4

97. 1975 2.0792678839032662E-4

98. 1976 2.0779359493636803E-4

99. Africa 2.0707635223766625E-4

100. South_Africa_1287 2.0667231407303497E-4

ANALYSIS

I think the pagerank order is reasonable due to:

1) USA has many importart inlinks having high pageranks on wikipedia such as Canada, Mexico, 2005,2006,2004,India,France which has contributed to the USA being the topmost page both on the local and the large dataset.

2) Similarly for 2004 being the second highest is due to the fact that USA which has the highest pagerank is linking to 2004 8 times which is 3 times more than 2005 and hence it is getting more share of the pagerank than any other page.

3) So the pattern here is that important pages contribute more to the pagerank. Hence I think that this output this reasonable.

4) On the local dataset the pagerank values of some pages will be slightly different because unlike the full dataset we just had a small number of pages

with their outlinks which meant that the important pages which were linking to other important pages might not be in the small dataset which might decrease it's pagerank value. So the full dataset ensured that the all the valuable links are covered.