CS 6240 Parallel Data Processing in MapReduce - 02
Homework 5
Dheeraj Joshi

Design Discussion

By taking into consideration the representation of inlinks for a page, I have taken the sparse matrix approach. Hence, I only emit inlinks and contributions for pages with inlinks and do not replicate 1/|V| across the matrix M. Instead, I store a flag in the column Vector R for dangling nodes and emit the pages with their inlink indexes and contributions(1/C(j)).

For version A: M[i, *] = (j,1/C(j))….
R[i, 1] = PageRank, isDangling

For version B: M[i, j] = (j,1/C(j))
R[i, -1] = PageRank,isDangling
R[i, 0] = delta

where i is the set of pages,
j is the set of inlinks
and isDangling is a flag to denote pages with no outlinks

Version A: Row By Column

For version A, I initially parse the given Wikipedia dataset and form a consolidated set of pages along with their outgoing links. This **parsing job** takes into consideration the sink nodes as well as assigns a unique index for forming the matrix later. Note that, the mapping of index to a page exists in the output of this job itself.

```
public void map(record){
        Emit(page,outlinks)
}
//Initialize counter ID to 0
public void reduce(page,List<Outlinks> list){
        id =id +1;
        For(v in list){
                Emit(page,id+outlinks)
        }
}
```

This parsing job output is then read in by a **Matrix M constructing job** which emits every page as <PageName:"N,1"> where N denotes a node and 1 denotes the index in the matrix and also emits every outgoing page as <OutlinkPageName:"A,1,0.2"> where A denotes an inlink , 1 denotes the page from which it arrives and 0.2 is the contribution from that page. These records are emitted with page name as key and therefore, the reducer gets the set of inlinks as well as the node record. We then form a row for our matrix as "2: (1,0.2)\t(..)" where 2 is the page index, 1 is an inlink with contribution of 0.2 of it's own PageRank. For pages with no inlinks, we simply emit "3: ()". I also, emit the initial

rank for all pages as "R,1,1/<total # of pages>, isDangling" into a separated directory. This forms my **column vector R** for PR computation.

```
public void map(record){
        emit(page,(N,id,0,isDangling))
        for(l in outlinks){
                emit(l,(A,id,outlinks.size))
        }
        emit(page,R,id,1/pageCount,isDangling)
}

public void reduce(page,List<inlinks> list){
        emit(page,inlinks)
}
```

This Matrix M is then read by a PageRank map-only job which also loads the R vector into cache. Note that, for the first iteration we pick up the initial PR values from the job above and for consecutive iterations we pick up the output of the last iteration. This job loads the R vector into cache and also, accumulates the delta contribution using the isDangling flag. Once the cache is populated into a hash map of PR values, we simply read every record from Matrix M denoting a row, and retrieve the corresponding inlink R values from the column vector. We multiply these PR values with their respective contributions and add the delta along with the "Alpha factors" to computer the new PageRank for the page row index. This value is then emitted as "R,1,newPR,isDangling" and passed as cache input to the next iteration.

```
//Load Initial R for 1st iteration/R output from 2nd iteration onwards
//Accumulate Delta
Public void map(record){
        For(l in inlinks){
                newPR += inlinkContribution * R[inlinkIndex]
        }
        newPR += delta/pageCount;
        newPR = (alph/pageCount) + (1-alpha)* newPR
        emit(R,pageIndex,0,newPR,isDangling)
}
```

After 10 iterations, the output vector R is read by the **Top K records job.** It simply reads every page index and provides a local top k in the map phase. These local top k page indexes are then read by the reducer to form a global top k set of page indexes with their PR values. Now inorder to map the indexes back to their page names, we simply read the output of the parsing job into cache and populate a hash map of (index, page name) pairs. This ensures that for every page index in the global top k set, we can retrieve it's corresponding page name and write to the final output.

```
//Map
public void map(){
        add to treemap(page,pageRank)
        if(map.size>100)
                delete first record
}
public void cleanup(){
        //Emit all records to reducer
}

//Reduce
public void reduce(){
        add to treemap(page,pageRank)
                if(map.size>100)
                        delete first record
}
public void cleanup(){
        //Emit all records to output
}
```

To sum it up, I have the following set of jobs:

- ✓ *1 Job for Parsing and Mapping Indexes to Page Names*
- ✓ *1 Job for forming the Matrix M and initial R vector.*
- ✓ *1 Map-only Job for computing the new R vector. (x10 iterations)*
- ✓ *1 Job for accumulating the global top 100 pages.*

Version B: Column By Row

For version B, I parse and create "page to index" mapping same as the version A does above.

```
public void map(record){
        Emit(page,outlinks)
}
//Initialize counter ID to 0
public void reduce(page,List<Outlinks> list){
        id =id +1;
        For(v in list){
                Emit(page,id+outlinks)
        }
}
```

This parsing job output is then read in by a **Matrix M constructing job** which emits every page as <PageName:"N,1"> where N denotes a node and 1 denotes the index in the matrix and also emits

every outgoing page as <OutlinkPageName:"A,1,0.2"> where A denotes an inlink , 1 denotes the page from which it arrives and 0.2 is the contribution from that page. These records are emitted with page name as key and therefore, the reducer gets the set of inlinks as well as the node record. We then form a column for our matrix as "2: (1,0.2), 3: (1,0.3), 4: (1,0.8)" where 2,3 and 4 are the page indexes and 1 is an outlink with contribution of 0.2,0.3 and 0.8 of it's own PageRank respectively. For pages with no outlinks, we simply emit "5: (),6: ()". I also, emit the initial rank for all pages as "R,1,-1,1/<total # of pages>, isDangling" into a separated directory. This forms my row vector R for PR computation.

```
public void map(record){
        emit(page,(N,id,0,isDangling))
        for(l in outlinks){
                emit(l,(A,id,outlinks.size))
        }
        emit(page,R,id,-1,1/pageCount,isDangling)
}

public void reduce(page,List<inlinks> list){
        for(l in inlinks){
                emit(pageindex,inlinkindex,contribution)
        }
}
```

This Matrix M is then read by a PageRank map job which also loads the R vector into cache. Note that, for the first iteration we pick up the initial PR values from the job above and for consecutive iterations, we pick up the output of the last iteration. This job loads the R vector into cache and also, accumulates the delta contribution using the isDangling flag. Once the cache is populated into a hash map of PR values, we simply read every inlink from Matrix M denoting a column, and retrieve the corresponding row of R values from the column vector. We multiply these PR values with their respective contributions and emit <"R,pageIndex,inlinkIndex,PR Contribution,isDangling"> with page index as key. We also emit the accumulated delta to the reducer with col index as 0. The reducer then accumulates all such PR contributions for a page and simply adds them. The reducer also takes the delta denoted by col index 0 and adds it to the sum. We then multiply this sum by an alpha factor to get the new PR of the page. We emit this new PR value as "R,pageIndex,-1,newPR,isDangling".

```
//Load Initial R/R #1-10 into cache
//Accumulate Delta
//Map
Public void map(record){
        PRContribution= inlinkContribution * R[inlinkIndex]
        emit(R,pageIndex,0,delta,isDangling)
        emit(R,pageIndex,inlinkIndex, PRContribution,isDangling)
}
```

```
//Reduce
public void reduce(page,List<R> list){
        delta=0.0;
        for(PR in list){
                if(inlinkIndex==0)
                        delta = delta/pageCount;
                else{
                        inlinkSum += PR;
                }
        }
        newPR = (alph/pageCount) + (1-alpha)* (inlinkSum+delta)
        emit(R,pageIndex,-1,newPR,isDangling)
}
```

After 10 iterations, the output vector R is read by the **Top K records job.** It simply reads every page index and provides a local top k in the map phase. These local top k page indexes are then read by the reducer to form a global top k set of page indexes with their PR values. Now inorder to map the indexes back to their page names, we simply read the output of the parsing job into cache and populate a hash map of (index, page name) pairs. This ensures that for every page index in the global top k set, we can retrieve it's corresponding page name and write to the final output.

```
//Map
public void map(){
        add to treemap(page,pageRank)
        if(map.size>100)
                delete first record
}
public void cleanup(){
        //Emit all records to reducer
}
//Reduce
public void reduce(){
        add to treemap(page,pageRank)
                if(map.size>100)
                        delete first record
}
public void cleanup(){
        //Emit all records to output
}
```

To sum it up, I have the following set of jobs:

- ✓ *1 Job for Parsing and Mapping Indexes to Page Names*
- ✓ *1 Job for forming the Matrix M and initial R vector.*
- ✓ *1 Job for computing the new R vector. (x10 iterations) (Map Computers Inlink PR's, Reduce*

Adds Inlink PR's)

- ✓ *1 Job for accumulating the global top 100 pages.*

Performance Comparison

Version A:

11 Machines: (1 Master 10 Slaves)	**Start**	**End**	**Total Time**
Pre-processing time:	17:35:22	17:56:16	0:20:54
Time to make M and Initial R:	17:56:27	18:04:12	0:07:45
Time to run ten iterations of PageRank:	18:04:29	18:22:55	0:18:26
Time to find the top-100 pages:	18:23:24	18:24:16	0:00:52

6 Machines: (1 Master 5 Slaves)	**Start**	**End**	**Total Time**
Pre-processing time:	18:35:33	19:09:26	0:33:53
Time to make M and Initial R:	19:09:37	19:19:33	0:09:56
Time to run ten iterations of PageRank:	19:19:48	19:46:23	0:26:35
Time to find the top-100 pages:	19:46:52	19:47:48	0:00:56

Version B:

11 Machines: (1 Master 10 Slaves)	**Start**	**End**	**Total Time**
Pre-processing time:	0:38:03	0:58:58	0:20:55
Time to make M and Initial R:	0:59:08	1:00:35	0:01:27
Time to run ten iterations of PageRank:	1:00:51	1:27:50	0:26:59
Time to find the top-100 pages:	1:28:18	1:29:12	0:00:54

6 Machines: (1 Master 5 Slaves)	**Start**	**End**	**Total Time**
Pre-processing time:	2:03:00	2:37:03	0:34:03
Time to make M and Initial R:	2:37:13	2:39:26	0:02:13
Time to run ten iterations of PageRank:	2:39:42	3:24:37	0:44:55
Time to find the top-100 pages:	3:25:05	3:26:03	0:00:58

CS 6240 Parallel Data Processing in MapReduce – 02
Homework 5
Dheeraj Joshi

Speed Up:

Jobs	6 Machines	11 Machines
Matrix PR Calculation Version A	0:02:40	0:01:51
Matrix PR Calculation Version B	0:04:30	0:02:42
Adjacency List PR Calculation	0:02:36	0:01:42

As we observe from the times above, the Homework 3 version was faster than the matrix calculation version A time but only by a few seconds. The column by row version is rightly the slowest due to it's additional overhead of adding the multiplied PR contributions and requires an additional reduce phase for computation. There was a definite time gain in the Matrix Approach but it somewhat nullifies with the time used for loading of the cache as observed in the logs. The network transfers were higher in the adjacency list representation as compared to the matrix versions due to complete inlink page name sets being transferred along with delta. The major gain of the matrix version is that it does not need an additional iteration for accumulating delta. This can be done simultaneously and does provide easier computation. As an inference, by doing further analysis on number of computations and selecting a range partitioner with effective quintile factor dividing the matrix M into n blocks, we might be able to achieve better times.

Top -100 Record Comparisons:

EMR Output HW3		***EMR Output HW5***	
United_States_09d4	0.002481279	United_States_09d4	0.002622783
2006	0.001158024	2006	0.00122845
United_Kingdom_5ad7	0.001134892	United_Kingdom_5ad7	0.001203088
Biography	9.42E-04	Biography	9.82E-04
2005	8.64E-04	2005	9.17E-04
England	8.32E-04	England	8.80E-04
Canada	8.09E-04	Canada	8.56E-04
Geographic_coordinate_system	7.34E-04	Geographic_coordinate_system	7.72E-04

France	6.83E-04	France	7.25E-04
2004	6.79E-04	2004	7.20E-04
Australia	6.43E-04	Australia	6.80E-04
Germany	6.17E-04	Germany	6.54E-04
2003	5.54E-04	2003	5.87E-04
India	5.53E-04	India	5.83E-04
Japan	5.50E-04	Japan	5.83E-04
Internet_Movie_Database_7ea7	5.07E-04	Internet_Movie_Database_7ea7	5.33E-04
Europe	4.79E-04	Europe	5.09E-04
Record_label	4.69E-04	Record_label	4.91E-04
2001	4.60E-04	2001	4.87E-04
2002	4.55E-04	2002	4.83E-04
World_War_II_d045	4.49E-04	World_War_II_d045	4.78E-04
Music_genre	4.46E-04	Population_density	4.70E-04
Population_density	4.45E-04	Music_genre	4.67E-04
2000	4.38E-04	2000	4.65E-04
Italy	4.19E-04	Italy	4.46E-04
London	4.09E-04	Wiktionary	4.36E-04
Wiktionary	4.08E-04	Wikimedia_Commons_7b57	4.35E-04
Wikimedia_Commons_7b57	4.06E-04	London	4.35E-04
English_language	3.92E-04	English_language	4.18E-04

1999	3.83E-04	1999	4.06E-04
Spain	3.41E-04	Spain	3.63E-04
1998	3.36E-04	1998	3.56E-04
Russia	3.23E-04	Russia	3.44E-04
Television	3.18E-04	1997	3.37E-04
1997	3.18E-04	Television	3.36E-04
New_York_City_1428	3.15E-04	New_York_City_1428	3.35E-04
Football_(soccer)	3.10E-04	Football_(soccer)	3.26E-04
Census	3.07E-04	1996	3.24E-04
1996	3.05E-04	Census	3.24E-04
Scotland	3.04E-04	Scotland	3.22E-04
1995	2.92E-04	1995	3.10E-04
China	2.90E-04	China	3.09E-04
Scientific_classification	2.90E-04	Population	3.04E-04
Square_mile	2.88E-04	Square_mile	3.04E-04
Population	2.87E-04	Scientific_classification	3.04E-04
California	2.85E-04	California	3.02E-04
1994	2.74E-04	1994	2.91E-04
Public_domain	2.72E-04	Sweden	2.88E-04
Record_producer	2.71E-04	Public_domain	2.87E-04
Sweden	2.71E-04	Film	2.86E-04

Film	2.71E-04	Record_producer	2.84E-04
New_Zealand_2311	2.67E-04	New_Zealand_2311	2.83E-04
New_York_3da4	2.63E-04	New_York_3da4	2.79E-04
Marriage	2.61E-04	Netherlands	2.77E-04
Netherlands	2.60E-04	Marriage	2.76E-04
United_States_Census_Bureau_2c85	2.60E-04	1993	2.75E-04
1993	2.59E-04	United_States_Census_Bureau_2c85	2.75E-04
1991	2.56E-04	1991	2.72E-04
Politician	2.54E-04	1990	2.68E-04
1990	2.53E-04	1992	2.66E-04
1992	2.51E-04	Politician	2.65E-04
Album	2.49E-04	Album	2.61E-04
Actor	2.46E-04	Latin	2.60E-04
Ireland	2.43E-04	Actor	2.58E-04
Latin	2.43E-04	Ireland	2.58E-04
Per_capita_income	2.43E-04	Per_capita_income	2.56E-04
Studio_album	2.42E-04	Studio_album	2.52E-04
Poverty_line	2.38E-04	Poverty_line	2.51E-04
Km^2	2.37E-04	Km^2	2.49E-04
1989	2.32E-04	1989	2.47E-04
Norway	2.28E-04	Norway	2.41E-04

Website	2.26E-04	Website	2.39E-04
1980	2.22E-04	1980	2.35E-04
Animal	2.18E-04	Animal	2.29E-04
Personal_name	2.17E-04	Area	2.29E-04
Area	2.16E-04	1986	2.27E-04
1986	2.14E-04	Personal_name	2.26E-04
Poland	2.13E-04	Poland	2.26E-04
Brazil	2.13E-04	Brazil	2.26E-04
1985	2.11E-04	1985	2.24E-04
1987	2.10E-04	1987	2.23E-04
1983	2.09E-04	1983	2.22E-04
1982	2.08E-04	1982	2.21E-04
1981	2.07E-04	French_language	2.19E-04
1979	2.06E-04	1981	2.19E-04
1984	2.06E-04	1979	2.19E-04
1988	2.06E-04	1984	2.19E-04
World_War_I_9429	2.05E-04	World_War_I_9429	2.19E-04
French_language	2.05E-04	1988	2.19E-04
1974	2.05E-04	Paris	2.18E-04
Paris	2.05E-04	1974	2.18E-04
Mexico	2.03E-04	Mexico	2.16E-04

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 5
Dheeraj Joshi

USA_f75d	1.99E-04	19th_century	2.12E-04
1970	1.99E-04	1970	2.11E-04
19th_century	1.97E-04	January_1	2.11E-04
January_1	1.97E-04	USA_f75d	2.11E-04
1975	1.96E-04	1975	2.09E-04
White_(U.S._Census)_c45a	1.96E-04	1976	2.08E-04
1976	1.96E-04	Africa	2.08E-04
Africa	1.95E-04	South_Africa_1287	2.07E-04

Simple Output HW3		***Simple Output HW5***	
United_States_09d4	0.005096647	United_States_09d4	0.005189009
Wikimedia_Commons_7b57	0.004718697	Wikimedia_Commons_7b57	0.004806766
Country	0.003850755	Country	0.003940285
England	0.002709991	England	0.002752481
Water	0.002629478	Water	0.00268781
Animal	0.00250238	Animal	0.002554088
City	0.002464279	City	0.002510824
Germany	0.002316092	United_Kingdom_5ad7	0.002358647
United_Kingdom_5ad7	0.002311653	Germany	0.002350402
France	0.002282353	Earth	0.002324735
Earth	0.002272975	France	0.002323608

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 5
Dheeraj Joshi

Europe	0.001997076	Europe	0.002038097
Wiktionary	0.001722305	Wiktionary	0.001753884
English_language	0.001715019	English_language	0.001749677
Government	0.001694566	Government	0.001732345
Computer	0.001684341	Computer	0.00171684
India	0.001679737	India	0.001713171
Money	0.001632113	Money	0.001667384
Japan	0.00152435	Japan	0.001551691
Plant	0.001493884	Plant	0.00152356
Italy	0.001479378	Italy	0.001507433
Canada	0.001453231	Canada	0.001481407
Spain	0.001445718	Spain	0.001471124
Food	0.001396752	Food	0.001424687
Human	0.001383624	Human	0.001412097
China	0.001369317	China	0.001396715
People	0.00135389	People	0.001382249
Australia	0.00130614	Australia	0.001329854
Asia	0.001256408	Asia	0.001284436
Capital_(city)	0.001249584	Capital_(city)	0.001274268
Television	0.001243346	Television	0.001264997
Sun	0.001231573	Sun	0.00126021

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 5
Dheeraj Joshi

Number	0.001218313	Number	0.001243236
State	0.00121302	State	0.001240376
Sound	0.001208956	Sound	0.001235212
Mathematics	0.001207404	Science	0.001232543
Science	0.00120575	Mathematics	0.001231057
Metal	0.001168748	Metal	0.001192305
Year	0.001152649	Year	0.001177093
2004	0.001151063	2004	0.001173357
Language	0.001126971	Language	0.001150166
Russia	0.00112323	Russia	0.001146182
Wikipedia	0.001098128	Wikipedia	0.00112333
Religion	0.001077206	Religion	0.001098567
19th_century	0.001075074	19th_century	0.001096539
Music	0.001069534	Music	0.001087431
Scotland	0.00103549	Scotland	0.001054801
20th_century	0.001032669	20th_century	0.001053705
Greece	0.001030129	Greece	0.001049223
London	0.00101004	Latin	0.001029861
Latin	0.00100919	London	0.001027355
Greek_language	9.83E-04	Greek_language	0.001004357
Energy	9.76E-04	Energy	9.99E-04

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 5
Dheeraj Joshi

World	9.65E-04	World	9.86E-04
Centuries	9.57E-04	Centuries	9.76E-04
Culture	9.23E-04	Culture	9.45E-04
History	9.16E-04	History	9.36E-04
Liquid	8.95E-04	Liquid	9.15E-04
Netherlands	8.89E-04	Netherlands	9.06E-04
Planet	8.84E-04	Planet	9.05E-04
Light	8.81E-04	Light	9.02E-04
Society	8.81E-04	Society	9.01E-04
Scientist	8.70E-04	Atom	8.90E-04
Atom	8.70E-04	Wikimedia_Foundation_83d9	8.88E-04
Wikimedia_Foundation_83d9	8.69E-04	Scientist	8.88E-04
Image	8.68E-04	Image	8.88E-04
Law	8.68E-04	Law	8.86E-04
List_of_decades	8.61E-04	Geography	8.79E-04
Geography	8.58E-04	List_of_decades	8.79E-04
Africa	8.44E-04	Uniform_Resource_Locator_1b4e	8.62E-04
Uniform_Resource_Locator_1b4e	8.43E-04	Africa	8.61E-04
Turkey	8.25E-04	Turkey	8.45E-04
Inhabitant	8.24E-04	Inhabitant	8.30E-04
Capital_city	8.09E-04	Capital_city	8.23E-04

CS 6240 Parallel Data Processing in MapReduce - 02
Homework 5
Dheeraj Joshi

Plural	8.03E-04	Plural	8.22E-04
Electricity	7.96E-04	Electricity	8.14E-04
Poland	7.83E-04	Poland	7.97E-04
Building	7.81E-04	Building	7.97E-04
Car	7.80E-04	Car	7.95E-04
Book	7.78E-04	Sweden	7.92E-04
Sweden	7.76E-04	Book	7.91E-04
Biology	7.71E-04	Biology	7.87E-04
War	7.56E-04	War	7.71E-04
Chemical_element	7.54E-04	Chemical_element	7.68E-04
God	7.48E-04	God	7.61E-04
North_America_e7c4	7.42E-04	North_America_e7c4	7.56E-04
September_7	7.38E-04	September_7	7.55E-04
Website	7.30E-04	Website	7.46E-04
Nation	7.25E-04	Nation	7.43E-04
Politics	7.24E-04	Politics	7.40E-04
2006	7.21E-04	2006	7.33E-04
Fish	7.18E-04	Fish	7.32E-04
Species	7.18E-04	Species	7.31E-04
Mammal	7.08E-04	Mammal	7.22E-04
Island	7.04E-04	Island	7.18E-04

Portugal	7.03E-04	Portugal	7.17E-04
Gas	6.99E-04	Gas	7.16E-04
River	6.98E-04	River	7.12E-04
Switzerland	6.92E-04	Switzerland	7.06E-04
World_War_II_d045	6.89E-04	World_War_II_d045	7.02E-04

The output pages have certainly not changed but a few pages have swapped ranks and that is down to a minor change in the precision of their page ranks. A possible explanation for this could be the delta being used immediately in the matrix version as compared to the homework 3 version. Also, the loss of precision bits might be due to the order of operations and the absence of long bit conversion for Hadoop. Therefore, pages with very close page rank values have simply swapped ranks in the top 100.