MMDS Project2 Report

Zheng Da (5100309039) yumaoshu@gmail.com
Zhao Yilong (5100309060) berzjackson@gmail.com
Li Wei (5092029004) liwei606@gmail.com

July 2, 2013

1 Background

1.1 Hadoop

1.1.1 Single-Node cluster

Befor running the Multi-Node clusters, we first set up the single node cluster on each of our three computer by the following procedures:

- Prerequisites
  - Java-7-OpenJDK
  - Adding a dedicated Hadoop user
  - Configure SSH (copy the rsa to others)
- Hadoop
  - Installation
  - Update $HOME/.bashrc (add export HADOOP_HOME and JAVA_HOME)
  - Configuration (hadoop-env.sh and x-site.xml)
  - Formatting the HDFS filesystem via the NameNode
  - Starting each single-node culster
  - Stopping each single-node culster
  - Running the MapReduce job for test

1.1.2 Multi-Node clusters

After setting up sigle-node cluster, we continue to set up the multi-node clusters.

1.1.3 Configuration

- Configure the conf/masters and conf/slaves file on master
- On all machines, configure the conf/x-site.xml by changing the `fs.default.name` parameter, which specifies the NameNode(the HDFS master) host and port.
- Since we have three nodes available, so we set `dfs.replication` to 3.

1.1.4 Formatting

Before we start our new multi-node cluster, we must format Hadoops distributed filesystem (HDFS) via the NameNode.

1.1.5 Starting the multi-node cluster

1. We begin with starting the HDFS daemons: the NameNode daemon is started on **master**, and DataNode daemons are started on all slaves (here: **master** and **slave**).
2. Then we start the MapReduce daemons: the JobTracker is started on **master**, and TaskTracker daemons are started on all slaves.

1.1.6 Stopping the multi-node cluster

Like starting the cluster, stopping it is done in two steps. The workflow however is the opposite of starting.

1. We begin with stopping the MapReduce daemons: the JobTracker is stopped on master, and TaskTracker daemons are stopped on all slaves.
2. Then we stop the HDFS daemons: the NameNode daemon is stopped on master, and DataNode daemons are stopped on all slaves.

1.2 PageRank

Pagerank is a link analysis algorithm that assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set.

A transition matrix need to be generated. And the pagerank will be calculated iteratively according to the value of pagerank in the last iteration until convergence. However if the matrix and the vector of the pagerank is to large to be stored and calculated in one single machine, we need to calculate the pagerank in a parallel and distributed way, where we need hadoop.

1.3 MapReduce

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. We can specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

In the pagerank algorithm, there are many ways to implement the pagerank in a map reduce style. Our group come up with three ideas on how to implemente the map reduce model of pagerank, details will be elabrated on next section.

2 Algorithms and Implementing details

We propose three methods to implement the pagerank algorithms in the map reduce model. Our goal is to enhance the efficiency of the MapReduce process leveraging the characteristics of Hadoop step by step. Additionally, these methods also differ with each other on their way to address the dead end problem.

2.1 Method 1

2.1.1 Overview

In this method, we implemented the Page-Rank algorithm with two map-reduce phases.

To describe it briefly, the first map-reduce phase will compute how page-rank values are distributed across the graph along with the edges in the this graph. Meanwhile, all the page-rank values that are not distributed via the edges, that is, page-ranks values of those dead-ends, are accumulated as a value called M, which stands for missing.
The second map-reduce phase is a map-only phase, as all it does is to evenly distribute this value M to all web pages in the graph. Of course, this computation is more than just a simple addition as we also consider the random-teleport factor in this phase.

2.1.2 MapReduce function

1. Mapper1

   - Input key-value pair: $< K_1, [V_1, V_2, ..., V_N] >$
   - Output key-value pair: $< V_1, PR[K_1]/N >$
   - K1 is the source web page while $[V_1, V_2, ..., V_N]$ are pages linked to by K1. V1 is one of the page that is linked to by K1, and PR[K1] is the page-rank value of K1. In this mapper, we distribute the page-rank value of a given web page along its out links evenly. Meanwhile, if N is 0, that is if K1 is a dead-end, we emit a special key-value pair, which is $< 0, PR[K_1] >$. This key-value pair will later be used to compute the value M, which is the sum of page-rank values that are lost in all those dead-ends.

2. Reducer1

   - Input key-value pair: $< K_1, [PR_1, PR_2, ..., PR_N] >$
   - Output key-value pair: $< K_1, [PR_{TOTAL}] >$
   - K1 is a web page while $[PR_1, PR_2, ..., PR_N]$ are page-rank values that are distributed to it via its in links. PR_{TOTAL} is simply the sum of [PR1, PR2, ..., PRN]. In this reducer, we sum up all the page-rank values that are distributed to a particular web page. What's special is that for the key-value pair with key 0, we get the sum of page-rank values is exactly the missing page-rank value M.

3. Mapper2

   - Input key-value pair: $< K_1, PR >$
   - Output key-value pair: $< K_1, PR' >$
   - This mapper is gonna take into account two effect: the distribution of the missing page-rank value M and the random teleport with factor beta. For a given page with page-rank value PR, we compute the new page-rank value PR' using $PR' = (1 - \beta) * 1/|N| + \beta * (M/|N| + PR)$.

4. Reducer2

   - Input key-value pair: $< K_1, V_1 >$
   - Output key-value pair: $< K_1 V_1 >$
   - This is simply an identity reducer as nothing needs to be done about all the key-value pairs.

2.1.3 Further discussion & Additional Discovery

There are some additional improvement and tricks:

- It can be obviously seen that it is possible to implement an in-mapper combiner in this mapper. We implemented a hash table to accumulate the sum page-rank values of a particular page, so that the total number of key-value pairs emitted in the phases will be greatly reduced.

- Instead of implementing an identity reducer in the second phase, we use the '-D mapred.reduce.tasksp=0' command to tell Hadoop that there is no need to reducer at all so that the sorting and shuffling will not be called, which makes whole process more efficient.

- The missing page-rank value M that are computed by the reducer of the first phase must be accessible to the mapper of the second phase. To make this possible, we make the reducer to output the accumulate with the key 0 to a special file. This is actually really tricky as this file is not in the HDFS, but on the local file system. Even trickier is that we don't know which machine in the cluster is gonna be the one who run this reducer and generate this file. To deal with this, we use shell command to synchronize this file across the cluster. Since this file only contain a value of M, it will not have negative impact on the performance of the algorithm.

2.2 Method 2

2.2.1 Overview

In this method, we treat deadends as vertexes with self-loop with the following consideration:

1. Self-loop can make the PageRank vector normalized.
2. Without self-loop, every deadend will end up with the same $PR = (1 - \beta)/N$.
3. It is natrual for every page to have a self-loop.

2.2.2 MapReduce function

Firstly, we modify the raw data into the following format with MapReduce process.
$"K_1 : PR[K_1]V_{11}V_{12}...V_{1N}"$

1. Mapper1

   Mapper 1 accept file from two distinct path, the raw data and the page rank value.

   For raw data:

   - Input key-value pair: $< line\#, "K_1 : V_{11}, V_{12}, ..., V_{1N}" >$
   - Output key-value pair: $< K_1, "V_{11}, V_{12}, ..., V_{1N}" >$

   For page rank value:

   - Input key-value pair: $< line\#, "K_1 PR[K_1]" >$
   - Output key-value pair: $< K_1, "PR[K_1]" >$

2. Reducer1
   - Input key-value pair: $< K_1, ["PR[K_1]", "V_{11}, V_{12}, ..., V_{1N}" >$
   - Output key-value pair: $< K_1, " : PR[K_1], V_{11}, V_{12}, ..., V_{1N}" >$

3. Mapper2
   - Input key-value pair: $< K_1, [PR[K_1], V_1, V_2, ..., V_N] >$
   - Output key-value pair: $< V_1, PR[K_1]/N >, < V_2, PR[K_1]/N >, ..., < V_N, PR[K_1]/N >$

4. Reducer2
   - Input key-value pair: $< K_1, [PR_1, PR_2, ..., PR_N] >$
   - Output key-value pair: $< K_1, [PR_{TOTAL}] >$

2.2.3 Further discussion & Additional Discovery

- For each iteration, MapReduce1 will perform 1.3G disk read and 1.1G disk write; MapReduce2 will perform 1G disk read and 300MB disk write.
  There are 2.3G disk read and 1.4G disk write in total. We need about (5 + 7 = 12) mins for each iteration.(3 computers)

- Optimization(Double the speed)
  The raw data is too big to store in memory; while the PR vector is OK to cache. We cached the PR vector into the memory by overwriting "setup" method in Mapper and with the help of "Distributed Cache" class. After this optimization, we eliminated MapReduce1.

- For each iteration, there are 1.3G disk read and 0.3G disk write. We need about 7 mins for each iteration.(3 computers)

2.3 Method 3

2.3.1 Overview

After searching for some factors that can affect the efficiency of Hadoop map reduce process, we find that there are two major factors that we can leverage to enhance the efficiency: network transmission and disk access time. So in this method, we aim to devise a way that can take advantage of the inner characteristers of hadoop to address the efficiency problem. So firstly, we add combiner in the mapper to reduce the network transmission time since there would be less pairs need to be transported from the mapper to the reducer. Notice that here we are able to add a combiner because the combiner only do the add or sum operation, which is commutative and associative. Secondly, we hope that the output of a mapper can be used in a reducer that is of the same node of the mapper, which requires the output of a mapper contain all the input pagerank so that the network transmission time can be largely reduced. To achieve this goal, we need to store the input document which contain the transfer matrix in a inverted index way. Thirdly, the pagerank vector v can be stored in the hdfs cache other than the disk or to be transported everytime to promote the disk access and the network transmission efficiency.

Generally speaking, this method firstly run an initialization function in map reduce model only once since the initialization aims to convert the input file with the format – source1: dest11 dest12 dest13... to a format – dest1: source11 degree[source11] source12 degree[source12] source13 degree[source13]...After this initialization, the pagerank algorithm will be iteratively runned in a map reduce style. This pagerank algorithm tackle with the dead end problem by adding the taxitation $1 - \beta$. So the sum of all the values of the pagerank will be less than 1. However, it won't affect the accuracy of the rank for each page.

Additionally, the sort of the pagerank results is also implemented in map reduce model. Which leverages the merge sort in the mapper.

2.3.2 MapReduce function

1. Mapper_for_Init
   - Input key-value pair: $< source_1, [dest_{11}, dest_{12}, ..., dest_{1n_1}] >$
   - Output key-value pair: $< dest_{11}, source_1\ n_1 >, < dest_{12}, source_{1n_1}\ n_1 >$
   - Suppose n_i represent the output degree of the $source_i$, here we only take $i = 1$ for example. Actually the input file is still different from the given file of the teacher. In the original file, the dead end won't appear as source. So we have preprocess the input file to add the dead end in the file with an empty string after the colon. On the hand, if a page is a source page, which only have links out with no links in, then their will be no pair contain this source page after the emission. So here we also emit the pair with $< source, "" >$, where the value is an empty and will be handled in the reducer. Details can be seen in the source code.
2. Combiner_for_init
   - Here the combiner performs the same with the reducer, so it can leverage the Reduce.class directly.
3. Reducer_for_init
   - Input key-value pair: $< dest_1, [source_{11}\ n_{11}, source_{12}\ n_{12}, ..., source_{1s}\ n_{1s}] >$
   - Output key-value pair: $< dest_1, \text{Text\_Concatenation}[source_{11}\ n_{11}, source_{12}\ n_{12}, ..., source_{1s}\ n_{1s}] >$
   - Here the reduce function returns a Text of the concatenation of each value. Following the discussion in the Mapper_for_Init, if the value of the pair is an empty string in the output file, it means it is a source page, hence every page no matter dead end or the source page won't be missed in the newly generated file. This output file contains the inversed index of each page, the key-value pair generated from this file in the mapper will contain all the income pagerank information, so their will be less network transmission cost for the mapper to reducer transportation if the reducer and the mapper are in the same node.

4. Mapper_for_PageRank
   - Input key-value pair: $< dest_1, \text{Text\_Concatenation}[source_{11}\ \ n_{11}, source_{12}\ \ n_{12}, ..., source_{1s}\ \ n_{1s}] >$
   - Output key-value pair: $< dest_1, v[source_{11}]/n_{11} >, < dest_1, v[source_{1s}]/n_{1s} >$
   - Here v is the pagerank vector stored in the memory, initialized with 1/5716808 the total number of the page, the vector v can be accessed by both mapper and reducer.
5. Combiner_for_PageRank
   - Here the combiner performs mostly likely to the reducer, but they still have difference. The combiner won't have the product with β and add the taxitation, it only returns the sum of the pagerank.
6. Reducer_for_PageRank
   - Input key-value pair: $< dest_1, \{pr[source_{11}], pr[source_{12}], \ldots pr[source_{1s}]\} >$
   - Output key-value pair: $< dest_1, sum > \{pr[source_{11}], pr[source_{12}], \ldots pr[source_{1s}]\} * \beta + (1 - \beta)/5716808$
   - Here $pr[source_{11}]$ is the pagerank value calculated by $v[source_{11}]/n_{11}$. Also it can be the sum of several pagerank whose destination is $dest_1$ since the combiner has summed up the pagerank.
7. Mapper_for_Sort
   - Input key-value pair: $< page_i, pr[page_i] >$
   - Output key-value pair: $pr[page_i], < page_i >$
   - Since the mapper will sort the pairs by key, so the pagerank value is set to be the key.
8. Reducer_for_Sort
   - Input key-value pair: $pr[page_i], \{< page_i >\}$
   - Output key-value pair: $< page_i, pr[page_i] >$

2.3.3 Further discussion & Additional Discovery

- In this method, an additional process "Initialization" is added. Undoubtedly it will introduce some cost to this method. However, since this initialization process only performs once and the pagerank algorithm will performs many iterations, the preprocess of the input file, namely the initialization process, make some sense.
- Considering why the inverted index input file will make the pagerank algorithm run faster, we can think about the process after mapper and before reducer. The output data of the mapper will be hashed to the reducer according to their keys. This process need the data to be transported from the mapper to reducer, which will cost a network transmission time. If the output pairs of the mapper will be reducer in the same node with mapper, the transmission time will be eliminated. Hence, we propose the inverted index file preprocess. Since the pairs of the same *dest* will be generated in the same mapper (possiblly not all), as in the file the *dest* also acts as the key, then these pairs will be hashed to the same reducer. Then we achieves the data process from the mapper to reducer is in the same node.
- The advantage of combiner is to reduce the dist access and the network transmission time. The output pairs of the mapper will be stored in the memory to be hashed and sorted. However, the memory is limilted and most of the time the pairs will be stored in the disk which introduces a cost for disk access, so the amount of the output pairs need to be reduced. Also the pairs will be transmitted to their hashed reducer, so less pairs will also make the network transmission time less. Hence the combiner can decrease the time cost efficiently.

2.4 Running the PageRank algorithm

We run the PageRank algorithm with our three laptops connected in **Ethernet**. We have implemented method 1 in `python` and method 2 and 3 in `java`. We have tested these three methods with $\beta = 0.85$. Method 1 and 2 have run 10 iterations and method 3 run 1 iteration. The results of top 100 pages with their pagerank value is shown in the evaluation section.

3 Evaluation of the results

3.1 Experiment results

The results of the pagerank value and the the top 100 pages is shown in the Table 1, this result is of method 2 with 10 iterations.

Table 1: Top 100 pagerank results

Title	**PageNO.**	**PageRank value**
United_States	5302153	0.002209359512533595
2007	84707	0.0014099756187816366
2008	88822	0.0013586752911541287
Geographic_coordinate_system	1921890	0.001252922460569376
United_Kingdom	5300058	0.0010108623085304637
2006	81615	8.667520417343134E-4
France	1804986	7.333235280577378E-4
Wikimedia_Commons	5535280	7.243712330737046E-4
Wiktionary	5535664	6.575311781782034E-4
Canada	896161	6.497972071116346E-4
2005	79583	6.175394797034801E-4
England	1601519	6.04170137022616E-4
Biography	687324	6.003675121433222E-4
Germany	1948883	5.849412865615936E-4
United_States_postal_abbreviations	5308545	5.513193837099443E-4
Australia	505135	5.290595901260609E-4
English_language	1603276	5.179609256461322E-4
June_15	2640611	5.105182434006226E-4
World_War_II	5596267	5.069626191212584E-4
Japan	2497500	4.851001627208616E-4
List_of_U._S._postal_abbreviations	2995510	4.698288994040508E-4
Europe	1650573	4.6395048577019444E-4
India	2370447	4.495919584862455E-4
2004	77935	4.365862330905947E-4
Italy	2437900	4.0352218346139177E-4
Music_genre	3492254	3.9811381035332824E-4
Race_and_ethnicity_in_the_United_States_Census	4141787	3.953123078342447E-4
Internet_Movie_Database	2401294	3.914101947105265E-4
Record_label	4189168	3.867804976124913E-4
Biological_classification	687618	3.7901573106808537E-4
Plural	3988566	3.697000556629321E-4
London	3072654	3.6441091104582257E-4
Area	434174	3.558258463646112E-4
Russia	4351989	3.432613626657996E-4
Population_density	4015997	3.415211394681675E-4
Spain	4696900	3.4139709102744113E-4
Binomial_nomenclature	686242	3.3976054603903553E-4
2003	76573	3.3825309476962233E-4
Latin	2876077	3.33368205361725E-4
		Continued on next page

Table 1 – continued from previous page

Title	PageNO.	PageRank value
Digital_object_identifier	1386743	3.2836159171656355E-4
New_York_City	3603437	3.260144282843706E-4
Time_zone	5115901	3.2214797814768785E-4
Association_football	478879	3.190712858005955E-4
Website	5492723	3.0740112818381253E-4
Roman_Catholic_Church	4302220	2.918457959723607E-4
Poland	3997849	2.908705156933415E-4
2001	74165	2.902807318686929E-4
China	1033539	2.8499064090490206E-4
2002	75323	2.813074123832398E-4
Public_domain	4089591	2.808081904160274E-4
Netherlands	3587465	2.74686261186023E-4
Abbreviation	181909	2.704229217258718E-4
Scotland	4490320	2.6165279555028317E-4
Population	4015913	2.6161208775414227E-4
French_language	1840972	2.589164700179258E-4
Sweden	4856540	2.5676818291969224E-4
California	880698	2.520635536584991E-4
World_War_I	5596263	2.505056217871289E-4
New_York	3603035	2.500990390338934E-4
2000	72989	2.4878115467777797E-4
List_of_countries	3013310	2.47699176958141E-4
Personal_name	3915373	2.4312406586861934E-4
Paris	3850181	2.4292837774973755E-4
Soviet_Union	4693429	2.3975272884239332E-4
German_language	1947988	2.3789130826065748E-4
New_Zealand	3607033	2.363433967299526E-4
Daylight_saving_time	1319777	2.3456331548606248E-4
North_America	3662151	2.3149698774290568E-4
Animal	380090	2.299485296509869E-4
Romania	4306671	2.2861551322422984E-4
Square_kilometre	4720025	2.2769133823496025E-4
Football_(soccer)	1781824	2.2461380734733279E-4
1999	66877	2.2161057395909652E-4
Greek_language	2041772	2.2115578077715637E-4
Brazil	774931	2.2111528222793216E-4
Mexico	3335081	2.2053538146535806E-4
Switzerland	4861926	2.1595171638175798E-4
Television	4936083	2.1489330678116527E-4
Metre	3331185	2.0868650562245222E-4
Africa	229601	2.0623857813657851E-4
Elevation	1569176	2.0464817986939917E-4
Norway	3674975	2.026817061552012E-4
Record_producer	4189215	1.9720928561684927E-4
Film	1741340	1.925418414446025E-4
Ireland	2415295	1.9225898614669298E-4
Asia	472437	1.9048289547735065E-4
1998	65954	1.865082136027623E-4
South_Africa	4681238	1.8644933357465882E-4
January_1	2496661	1.8553297486415774E-4
Washington,_D.C.	5477621	1.8519996803721453E-4
Greece	2040703	1.8451070483226916E-4
Mathematics	3266921	1.8398252041072012E-4
Belgium	630330	1.829003324727559E-4

Continued on next page

Table 1 – continued from previous page

Title	PageNO.	PageRank value
Arabic_language	421957	1.8182917213543215E-4
Politician	4003069	1.806624622206276E-4
Square_mile	4720054	1.8040146968770313E-4
Spanish_language	4698528	1.7999730420057268E-4
Russian_language	4353393	1.7918988859125923E-4
Austria	508777	1.7762837274857283E-4
Portugal	4023070	1.7672069933923073E-4

The comparasion of the three methods for the top 100 pagerank results is listed in the Table 2. From left to right, the page number and the pagerank value are of method1, method2, method3 in order.

Table 2: Top 100 pagerank results in our 3 methods

Method1 page	Method1 pr	Method2 page	Method2 pr	Method3 page	Method3 pr
5302153	0.00222766	5302153	0.00220936	5308545	0.00304369
84707	0.00142018	84707	0.00140998	5302153	0.00228777
88822	0.00136995	88822	0.00135868	1921890	0.00205034
1921890	0.00126278	1921890	0.00125292	687324	0.00158252
5300058	0.00101725	5300058	0.00101086	88822	0.00123752
81615	0.000873097	81615	0.000866752	84707	0.00118954
1804986	0.000736552	1804986	0.000733324	5300058	0.000859021
5535280	0.000728076	5535280	0.000724371	3492254	0.000852523
5535664	0.000657204	5535664	0.000657531	1804986	0.000829547
896161	0.000655401	896161	0.000649797	4189168	0.00082092
79583	0.000622667	79583	0.000617539	687618	0.000787974
1601519	0.00060847	1601519	0.00060417	1601519	0.000740804
687324	0.000603935	687324	0.000600368	896161	0.00064562
1948883	0.00058831	1948883	0.000584941	3915373	0.000625695
5308545	0.000555066	5308545	0.000551319	81615	0.000625467
505135	0.000533199	505135	0.00052906	2401294	0.000605807
1603276	0.000518677	1603276	0.000517961	2370447	0.000543413
5596267	0.000509996	2640611	0.000510518	686242	0.000507369
2497500	0.000489657	5596267	0.000506963	1948883	0.000492182
2995510	0.000473783	2497500	0.0004851	505135	0.000490056
1650573	0.000466185	2995510	0.000469829	79583	0.000481599
2370447	0.000450751	1650573	0.00046395	2497500	0.000479041
77935	0.000439959	2370447	0.000449592	4813259	0.000464092
2437900	0.000404997	77935	0.000436586	5394902	0.000427467
4141787	0.000404123	2437900	0.000403522	4189215	0.000418115
3492254	0.000403592	3492254	0.000398114	1781824	0.000403935
2401294	0.000396123	4141787	0.000395312	4003069	0.000401881
4189168	0.000392232	2401294	0.00039141	4306671	0.00038888
687618	0.000382841	4189168	0.00038678	1603276	0.000387215
3988566	0.000369928	687618	0.000379016	5115901	0.000349128
3072654	0.00036702	3988566	0.0003697	1355876	0.000347963
434174	0.000360483	3072654	0.000364411	5535664	0.000347502
4015997	0.000346056	434174	0.000355826	1920395	0.000342364
4351989	0.00034455	4351989	0.000343261	5243336	0.000337211
4696900	0.000343421	4015997	0.000341521	77935	0.000336512
686242	0.00034329	4696900	0.000341397	4936083	0.000330175
76573	0.000341008	686242	0.000339761	2437900	0.000328246
2876077	0.000330859	76573	0.000338253	1650573	0.000328227

Continued on next page

Table 2 Top 100 pagerank results in our 3 methods

Method1 page	Method1 pr	Method2 page	Method2 pr	Method3 page	Method3 pr
3603437	0.000329289	2876077	0.000333368	275656	0.000325794
1386743	0.00032727	1386743	0.000328362	1165553	0.000318766
5115901	0.000325344	3603437	0.000326014	5492723	0.000312477
478879	0.000324582	5115901	0.000322148	380090	0.000307072
5492723	0.000309588	478879	0.000319071	3072654	0.00030619
3997849	0.000294498	5492723	0.000307401	2330913	0.000296159
74165	0.000292704	4302220	0.000291846	5535280	0.000295846
4302220	0.000291798	3997849	0.000290871	5596267	0.000295287
1033539	0.000285522	74165	0.000290281	3997849	0.000290544
75323	0.000283542	1033539	0.000284991	76573	0.00028208
4089591	0.000279796	75323	0.000281307	4015997	0.000278895
3587465	0.00027617	4089591	0.000280808	4089591	0.000266646
181909	0.000271761	3587465	0.000274686	3988566	0.00026379
4015913	0.000264625	181909	0.000270423	3603437	0.000259466
4490320	0.000263218	4490320	0.000261653	206622	0.000250157
1840972	0.000259066	4015913	0.000261612	1386743	0.000243902
4856540	0.000258426	1840972	0.000258916	880698	0.000243299
880698	0.000254562	4856540	0.000256768	4696900	0.000241915
3603035	0.000252411	880698	0.000252064	74165	0.00024015
3013310	0.000251944	5596263	0.000250506	5185303	0.000239888
5596263	0.00025169	3603035	0.000250099	1569176	0.000237826
72989	0.000250873	72989	0.000248781	3674975	0.000236988
3915373	0.00024456	3013310	0.000247699	75323	0.000233066
3850181	0.000244044	3915373	0.000243124	4351989	0.000226244
4693429	0.000240635	3850181	0.000242928	1033539	0.000225824
1947988	0.000238106	4693429	0.000239753	1741340	0.000224639
3607033	0.000237986	1947988	0.000237891	4856540	0.000223306
1319777	0.000237362	3607033	0.000236343	4490320	0.000222973
3662151	0.000233375	1319777	0.000234563	313376	0.000220645
380090	0.000232213	3662151	0.000231497	3979494	0.000220505
4720025	0.000230844	380090	0.000229949	434174	0.000217225
4306671	0.000230207	4306671	0.000228616	5601388	0.000211767
1781824	0.000227862	4720025	0.000227691	3607033	0.000210878
66877	0.000223598	1781824	0.000224614	774931	0.000210079
774931	0.000222613	66877	0.000221611	3603035	0.000207484
3335081	0.000222278	2041772	0.000221156	4482375	0.000205752
2041772	0.000219714	774931	0.000221115	72989	0.000203615
4861926	0.000217575	3335081	0.000220535	380733	0.000201126
4936083	0.000217422	4861926	0.000215952	181909	0.000200852
3331185	0.000210254	4936083	0.000214893	4015913	0.000200658
1569176	0.000207339	3331185	0.000208687	3587465	0.000194781
229601	0.000206792	229601	0.000206239	66877	0.000190431
3674975	0.000203975	1569176	0.000204648	1045817	0.000190017
4189215	0.000200073	3674975	0.000202682	1319777	0.000189898
1741340	0.00019489	4189215	0.000197209	821412	0.000181231
2415295	0.000193422	1741340	0.000192542	3992956	0.000180736
472437	0.000191198	2415295	0.000192259	1142648	0.000175147
65954	0.000188122	472437	0.000190483	2415295	0.000172417
4681238	0.000187424	65954	0.000186508	4487654	0.000171222
2496661	0.000187003	4681238	0.000186449	65954	0.000170158
5477621	0.000186809	2496661	0.000185533	4611853	0.000169328
2040703	0.000185093	5477621	0.0001852	1773129	0.000168003
630330	0.000183908	2040703	0.000184511	3331185	0.000167596
4720054	0.000182497	3266921	0.000183983	4288238	0.000167194

Continued on next page

Table 2 Top 100 pagerank results in our 3 methods

Method1 page	Method1 pr	Method2 page	Method2 pr	Method3 page	Method3 pr
4003069	0.0001813	630330	0.0001829	3013310	0.00016718
3266921	0.000180469	421957	0.000181829	3335081	0.000164056
4698528	0.000180205	4003069	0.000180662	3850181	0.000161522
421957	0.000180134	4720054	0.000180401	5252582	0.000159251
4353393	0.000179159	4698528	0.000179997	1198405	0.000156433
508777	0.000178651	4353393	0.00017919	964099	0.000149442
4023070	0.000177537	508777	0.000177628	1597989	0.000149073
2496936	0.000175277	4023070	0.000176721	1187884	0.000148183

The PageRank results with the top 100 pages ranked by the number of in-links is listed in Table: 3.

Table 3: Top 100 pagerank results with in link methods

Page NO.	In links
5302153	374934
1921890	294604
88822	286409
84707	266614
687324	154656
81615	146336
5300058	139325
3492254	129952
4189168	123784
1804986	123553
79583	120091
1601519	118170
4015997	99908
896161	99651
1948883	95366
2401294	94442
687618	90263
5115901	89733
434174	87527
77935	82344
1569176	81765
505135	78810
2497500	76037
4189215	72790
3013310	71323
5535280	70096
1603276	69408
5492723	67103
5596267	66648
2437900	65493
686242	63158
2370447	61883
76573	61812
3072654	60232
1650573	58290
1781824	57880
4015913	57294
3997849	56369

Continued on next page

Table 3 Top 100 pagerank results with in link methods

Page NO.	In links
4720054	56046
5394902	55594
4813259	54720
74165	54170
964099	53843
1319777	53170
75323	52661
2496936	51493
1201051	51344
4696900	48750
3603437	47314
72989	47034
880698	44717
66877	43949
4089591	43588
1078771	43318
3603035	42672
5303198	42394
1165553	42379
5308545	41051
4351989	40209
313376	39648
2330913	39591
4003069	38845
3240494	38804
275656	38711
5295862	38067
65954	37396
1355876	37377
4490320	36847
1033539	36816
3906702	36762
380090	36599
1386743	36492
4031749	36307
3331185	36016
1921845	35465
4856540	35363
1198405	34780
4002307	34400
3712833	33965
4611853	33937
2768574	33781
65064	33567
3283930	33518
967347	32786
4936083	32785
206622	32521
5275322	32460
5275367	32222
1741340	31934
5303187	31916
774931	31656
1717039	31474

Continued on next page

Table 3 Top 100 pagerank results with in link methods

Page NO.	In links
64233	31447
5524334	31374
3850181	31315
3587465	31152
4935409	30995
3607033	30936
3335081	29414
5596263	29076

3.2 Anslysis of the results

1. The comparision table of the three methods

   - From the tables above, we can find that in Table 2, the three methods have the similar pagerank results basically. The method1 and method2 are especially close to each other, but they may be different from the standard pagerank results which leverages the method in our textbook because that method 1 and method 2 all devise their new way to cope with the dead end and make the sum of the pagerank value to be 1. Although these two methods use the different way to adress the dead end problem, the rank won't be changed and they are still very useful.
   - In method 3, we can see that it has a bigger difference with the other two methods, the main reason lies in that it is only be run with 1 iteration. And we can check that the general order has been generated successfully. In this method, the way to tackle with dead end is the same with our textbook, so the sum of all the pagerank value will be less than 1.
   - All of these three method are run with the parameter $\beta = 0.85$.

2. The comparision to the rank with the in link amount

   - We can compare the results in the Table: **??** with the results in the Table: 2. We can see that the rank of the in links method is close with the rank results of our methods. The reason is that, most significant pages, with the important pages link to, may also have a large number of pages link to. On the other hand, if a page has large number of pages link to, it will have higher possibility to get a higher pagerank value. So compute the number of in links is also a way to evaluate the significance of a page, and the pagerank algorithm is an improvement to the in links method which will have a more accurate and reasonable pagerank value.

4 Contribution of each member

In this project, the three of us all participated in the discussion, searching for useful references, devising innovative and efficient methods, implementing the pagerank algorithms for different methods and finishing the report. Each of us has made great contribution to the accomplishment of the hadoop project in these tired and tensive days. Moreover, through the heated discussion and practical implimentation all of us feel that we have gained deeper and better undertanding of what we have learned in class about hadoop, mapreduce, distributed system and pagerank algorithm. We have tried our best to learn, think, create and implement in these limited days.

5 Reference

1. MapReduce indexing strategies: Studying scalability and efficiency, Richard McCreadie, Craig Macdonald, Iadh Ounis

2. Shuffle in MapReduce, http://langyu.iteye.com/blog/992916

3. How partitioning, collecting and spilling work in MapReduce, http://grepalex.com/2012/09/24/map-partition-sort-spill/

4. Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer