

MapReduce: Data Distribution for Reduce

Graduate Group #10

Team member 1 (Heena Dave, 810917421, hdave@kent.edu)

Team member 2 (Bhavy Bhut, 810917417, bbhut@kent.edu)

1. Introduction

Hadoop is based on master-worker architecture. Master node is responsible for job scheduling and worker node runs tasks assigned by master node. Scheduling in master node is based on a hash function in hadoop architecture. By default, hash function is $Hash (Hashcode (intermediate\ key) \bmod number\ of\ Reducers)$. The default hash function is effective and provides load balance for uniformly distributed data. However, this hash function failed to achieve effective load balance for non-uniform data. For example, in basic Word Count Program, basic words (a, an, the, etc.) appear more frequently than other words so it imposes more work on corresponding reducers. Consider an example shown in below figure for letter count example.

a	b	c	a	b	d	c	a	B	d	a	a	a	b	c	e
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Hash function load	a: 6, d: 2	b: 4, e: 1	c: 3
Optimal load	a: 6	b: 4, d: 2	c: 3, e: 1

Above figure illustrates example of key assignment in MapReduce, it also depicts scenario of hash function load balancing and optimal load balancing for three reduce workers. Hash function assigns keys *a* and *d* to first machine, *b* and *e* to second machine and only *c* to third machine. As result first machine has maximum load of 8 while third machine has only load of 3. As shown in optimal load, hash function load balancing is not effective.

As in web index data, search engines and social sites like google, Facebook, twitter has more visitors than other sites. Now, if we want to calculate the number of times a website is visited then as per default hash function reducers with search engines and social sites has more data to process than other nodes. It will increase overall processing time as few reducers will sit idle until all the reducers produces final output value.

2. Project Description

MapReduce programming model performs tasks based on dividing work among two functions: Map and Reduce. The map function generates intermediate key-value pair while the reduce function performs the final output. For example, for URL-count example, map function generates key-value pairs where each URL being the key and value will be one for each key. Then, the reduce function performs the count operation. Here, between Map and Reduce function calls, there are some sorting operations done to

sort all the values of the same key together so that one reduce function can take all the intermediate key-value pairs belonging to same key.

The problem with this technique is, in real, the data is not evenly distributed. Hence, the data distribution for reduce function is not efficient in a case where there are some URLs being hit a lot and some URLs being hit only a few times. In this case, the reduce function handling URLs being hit more will be busy calculating the final count value and the reduce function handling URLs being hit rarely will be free as it will finish calculating the final value faster.

The project is about distributing data for reduce function effectively so that no reduce function takes a lot of time to calculate the final value while other reduce functions are idle.

Challenges to tackle this problem is that master node needs to distribute data to reduce workers in even manner to achieve effective load balancing. As the data for the same key is distributed between multiple reduce functions, there is one final reduce operation must be performed to calculate final value from previous reduce functions' values. The data distribution should be effective enough in such a way that even there is one more reduce function to be performed, the performance should still be efficient than the original MapReduce functionality.

We are two members working on this project: Heena Dave will be working on Algorithm design and load balancing and Project Report while Bhavy Bhut will work on Program Creation and Performance comparison between original MapReduce and data distribution of Reduce function.

3. Background

- Related papers (or surveys for graduate teams)
 - Hesham A. Hefny, Mohamed Helmy Khafagy, Ahmed M Wahdan, "Comparative Study Load Balance Algorithms for Map Reduce environment", International Journal of Applied Information Systems (IIAIS) – ISBN : 2249-0868 Foundation of Computer Science FCS, New York, USA
From this paper, we took reference of different load balance algorithms for map reduce environment. How they proposed various techniques to tackle skew problem in MapReduce environment.
 - Yanfang Le, Jiangchuan Liu, Funda Ergun, Dan Wang, "Online Load Balancing for MapReduce with Skewed Data Input", ISBN 978-1-4799-3360
This paper proposed an online load balancing technique, they used online minimum makespan to achieve optimal hash function which described in Section 1. They tried to schedule pairs with identical keys on same machine in MapReduce.

- YongChul Kwon, Magdalena Balazinska, Bill Howe and Jerome Rolia. “A Study of Skew in MapReduce Applications”

This paper provided study on load balancing problems. From this paper, we referenced common sources of skew in MapReduce applications and skew problems using real workloads.

- Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters”, Google, Inc.

This paper is the main paper for MapReduce where the concept of Map and Reduce functions are defined. So, we referenced MapReduce architecture and data distribution in MapReduce from this paper.

- Software Tools

- Library: Apache Hadoop which we use as a distributed data storage and Yarn for job scheduling
- Java: Java Sdk 6 or higher
- IDE: Eclipse IDE for source code
- DBMS: not required

- Required hardware

- Hardware requirement for Slaves

Workload pattern	Storage	Processor (1.8 GHz or higher recommended)	Memory (GB)	Network
Balanced	Based on input data	4	8	1 GB onboard
Compute intensive		8	16	
Storage heavy		4	8	

- Hardware requirement for Master (Namenode)

Workload pattern	Storage	Processor (1.8 GHz or higher recommended)	Memory (GB)	Network
Balanced	Based on input data	4	8	1 GB onboard
Compute intensive		4	8	
Storage heavy		4	8	

- Related programming skills

- Basic understanding of Hadoop File System
- Should be aware of MapReduce Framework for functionality of this project
- Object oriented programming
- Distributed Environment

4. Problem Definition

- Formal (mathematical) definitions of problems

The problem with MapReduce is that the data for Reduce function is not evenly distributed and it is distributed based on the key values. This works fine if the values for each key is almost equal and in that case, data distribution among reduce functions will be equal. On the other hand, the data in real world application is non-uniform. Non-uniform data means that few keys will have loads of values associated with it and few keys will have lesser values and others might have only selected values associated. Hence, certain reduce functions will take a lot of time to perform calculation of final values whereas others will finish faster and sit idle until all the reduce functions are done with the calculations. In any MapReduce task, until all reduce functions are not finished with their calculations, a new MapReduce cycle cannot be started. So, data distribution in MapReduce should be even. This will be helpful as all the reduce functions will finish their tasks almost simultaneously and so no reduce function will have too much load and no reduce function will sit idle.

The project is about making a programming model in which the Map function will act similar to MapReduce programming model but the data distribution for Reduce function will differ as to make data distribution more even among reducers considering the performance improvement.

For example, in letter count, the vowels a, e, i, o and u are referred more frequently than the consonants. So, the reducers handling vowels will take a lot more time than the ones handling consonant. Now, even if a reducer calculates count of 4-5 consonants together but still in some cases the data distribution will not be even.

- Challenges of tackling the problems

Although, the data distribution is an important factor, when the values of same key is distributed amongst different reducers, there is a final reducer call is needed to calculate final value from all the calculated values of reducers handling the same key. This extra function call will also take some amount of time and it needs to be considered in performance. Hence, the data distribution should occur only in some cases where there is a huge amount of difference between numbers of values for the same key.

- A brief summary of general solutions in your project

The data distribution for Reducer should be more uniform so that all the reducer functions finish their tasks almost simultaneously and no function will sit idle for long time. This will increase performance as well as utilize the resources efficiently. To do so, an algorithm should be designed in a way that will calculate total number of keys, total number of values and a ratio of total values / total reducers. Based on this ratio, reducer function will handle key-value pairs with closest ratio. The ratio will be calculated for each mapper output as in distributed system, the mapper output will be distributed too so

it would not be feasible to count total values across all the mappers. Also, input data is evenly distributed among mappers and so the mapper output will also be even in values and can be distributed among reducers.

For example, there are five keys for word count:

A: 10, B: 20, C: 30, D: 40, E: 1500

Now there are two mappers and two reducers that can perform the MapReduce operations:

Mappers generate key-value pairs for the keys assigned to them. Like,

<A: 1>
<A: 1>
<A: 1>
<A: 1>
.
.
.

And now there are two reducer functions available which can calculate count value for each key then the normal MapReduce will work as follows:

Mapper 1 Output: A: 5 (5 key-value pairs of <A, 1>), B: 15, C: 10, D: 20, E: 750

Mapper 2 Output: A: 5, B: 5, C: 20, D: 20, E: 750

Now Mapper 1 output will be evenly distributed among two reducers:

Total keys for Mapper 1: 5

Total values for Mapper 1: 800

Instead of distributing data among two reducers based on key, our model distributes based on total values/ total reducers which will be $800/2 \Rightarrow 400$ in this case.

Reducer 1 Input from Mapper 1:

A: 5
B: 15
C: 10
D: 20
E: 350

Reducer 2 Input from Mapper 1:

E: 400 (400 <E, 1> key-value pairs)

Similarly, Mapper 2 output will be distributed among two reducers.

As the mapper output will be sorted based on keys, the values with same key has higher chances of being executed on same reducer.

At last, final reducer will run, which will calculate following output:

A: 10, B: 20, C: 30, D: 40, E: 1500

5. The Proposed Techniques

- Framework (problem settings) and Algorithm

We have identified a problem with data distribution among reduce workers. As the data distribution in current MapReduce is key-based, some reducers process a lot of values than the others. Hence, we propose a technique to distribute load between reducers evenly.

Algorithm for our proposed technique is as below:

1. Take number of Mappers and number of Reducers from User.
2. Take Input Data.
3. Divide input data evenly between all the mappers.
4. Mapper generates intermediate key-value pairs as intermediate output.
5. Sort mapper output based on keys.
6. Take each intermediate mapper output (key-value pairs) and divide the output evenly among reducers.
7. Run final reducer to calculate final output key-value pairs based on all the reducer's output.

Fig 1: Data distribution algorithm

The algorithm requires number of mappers and number of reducers to be given by user. Then, we divide input data evenly between all mapper functions based on the number of mappers. Map functions generate intermediate output <Key,Value> same as Google MapReduce. Hence, for each key there will be some value element and one key can be repeated multiple times. This intermediate output is sorted ascending based on keys.

After that intermediate output from each map function is distributed among reducers in even manner. For example, there are 6 mappers and 3 reducers available, then output of mapper 1 will be distributed among 3 reducers evenly. As the intermediate output of mapper function is sorted, the values with same key has higher chances of being executed

on the same reducer. Similarly, output of mapper 2 will be distributed among 3 reducers and so on.

Once, all the reducers perform their tasks, a final reducer is called which combines the output of all the reducers. As each reducer has equal amount of load to process data, all reduce functions can complete it's processing simultaneously. Final reduce function is performed to aggregate data and generate final output.

We have implemented URL count program to compare the performance of traditional MapReduce and our approach. As of now, our implementation is for standalone application.

For example, we have taken 100 URLs for which count operation needs to be performed and have also taken 6 Mappers and 3 Reducers. So, in the first step, the 100 URLs are distributed among 6 Mappers for processing and each mapper generates <URL, 1> intermediate value. Now, the output of each Mapper is sorted in ascending manner. Hence, we can say that the URLs with same key are grouped together. Now, the output of individual mappers are distributed among reducers as explained above. So, if a website like Google or Facebook is hit more frequently, it would not be running on same reducer giving it too much of load. But, it would be grouped together, so until a maximum number (which is Total URLs/values Processed by Mapper / Total Reducers), it can be processed on the same reducer. At last, final reducer combines all the values.

Algorithm provides following advantages over traditional MapReduce framework:

It provides effective utilization of reducers as map function output is evenly distributed among reducers and all reducers can finish their task almost simultaneously. No reduce function have loads of values to process while other reduce functions finish their task and sit idle. In Master-Slave architecture, if Master node already assigned all tasks to slaves then no reducer will wait idle for a long time for a busy reducer to finish its task. In MapReduce, until all the reducers finish their tasks, a new MapReduce cycle cannot be started. So, in traditional MapReduce, if a reduce function has loads of keys to process then it makes other reducers to wait until all the reducers are finished with their tasks and a new cycle can be started. In our approach, this problem does not happen as reducers finish their tasks almost simultaneously and no reducer has to wait for a long time to finish the cycle.

6. Visual Applications

- GUI design

For GUI, we used Java Programming Language to design algorithm. Below is directory structure of source files.

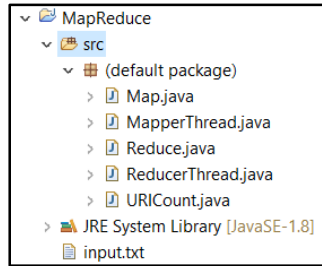


Fig 2: Source directory structure

To visualize processing from mappers and reducers, we dumped data in form of txt files. Below is example of **6 Mappers and 3 Reducers** with fifty records. Configure number of mappers and reducers in attached source code using `numMappers` and `numReducers` variable, if required.

MapOutput0.txt	5/4/2017 2:24 PM	Text Document	1 KB
MapOutput1.txt	5/4/2017 2:24 PM	Text Document	1 KB
MapOutput2.txt	5/4/2017 2:24 PM	Text Document	1 KB
MapOutput3.txt	5/4/2017 2:24 PM	Text Document	1 KB
MapOutput4.txt	5/4/2017 2:24 PM	Text Document	1 KB
MapOutput5.txt	5/4/2017 2:24 PM	Text Document	1 KB
MapperOutputFiles.txt	5/4/2017 2:24 PM	Text Document	1 KB
ReduceOutput0.txt	5/4/2017 2:24 PM	Text Document	1 KB
ReduceOutput1.txt	5/4/2017 2:24 PM	Text Document	1 KB
ReduceOutput2.txt	5/4/2017 2:24 PM	Text Document	1 KB
ReducerFinalOutput.txt	5/4/2017 2:24 PM	Text Document	1 KB
ReducerFinalOutputFile.txt	5/4/2017 2:24 PM	Text Document	0 KB
ReducerOutputFiles.txt	5/4/2017 2:24 PM	Text Document	1 KB

Fig 3: Data dump from Mappers and Reducers

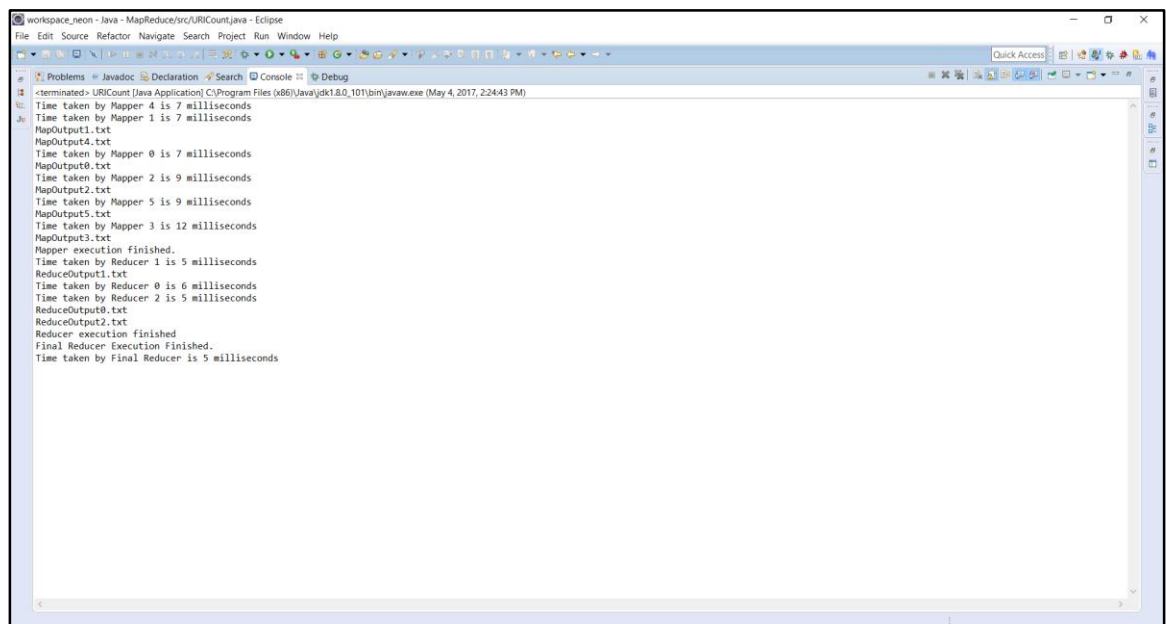


Fig 4: URL Count Program output based on even data distribution algorithm

- Design modules

Google MapReduce algorithm distribute data to reducers based on key. Each reducer has only specific data related to key only. Below flowchart shows dataflow for Unrealistic (synthetic) and Realistic (Real-time) Data in Google MapReduce.

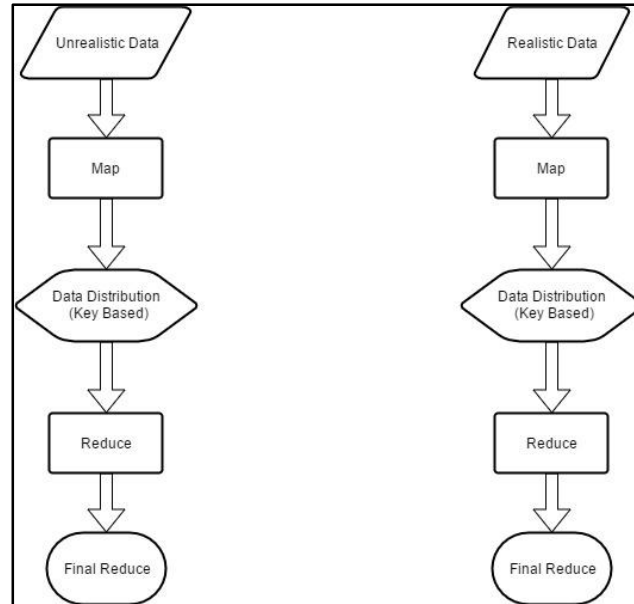


Fig 5: Flowchart for Google MapReduce algorithm

Design module is based on even data distribution between reduce functions so that each reducer can get fair amount of data to process and no reduce function will sit idle.

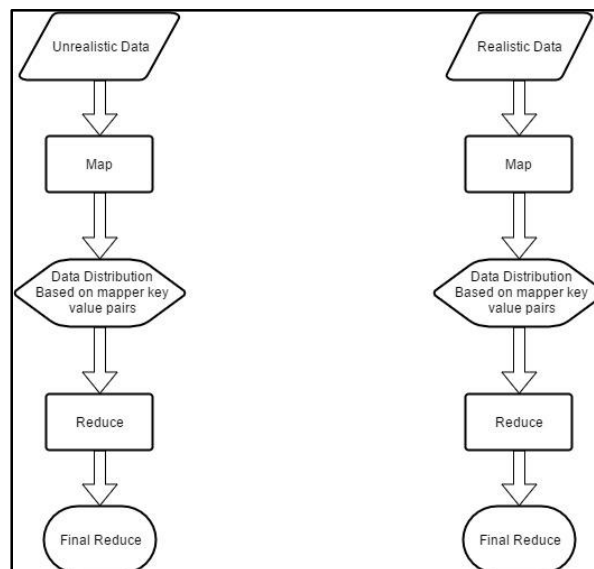


Fig 6: Flowchart for data distribution algorithm based on mapper key value pairs

7. Experimental Evaluation

- Experimental settings

For evaluation of algorithm, we used two types of datasets: Unrealistic (synthetic) and Realistic (real time) filtered datasets. (Source: <http://ita.ee.lbl.gov/html/traces.html>) First we measured performance of this datasets on Hadoop. Hadoop performance for realistic dataset is quite poor compare to unrealistic dataset because in real time some URLs are visited more frequently compare to others like search engines, social network sites, etc. Due to key based distribution in Google MapReduce algorithm, this key-value pair has more data compare to others which leads to more processing time for reducer. Below figure describes Hadoop experiment.

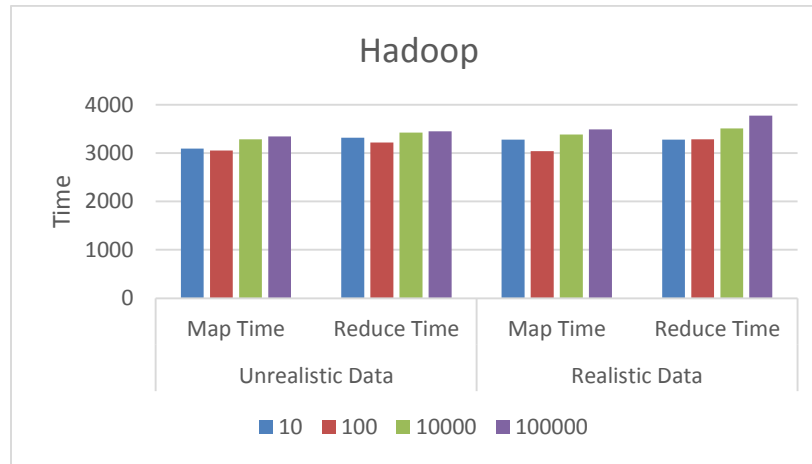


Fig 7: Hadoop experiment for unrealistic and realistic datasets (Time milliseconds)

As shown in chart, Map function performance is almost same for realistic and unrealistic datasets. For 0.1 million records, reduce function with realistic data takes more time for processing.

- The performance report

To measure performance, we compared even data distribution algorithm with Google MapReduce algorithm. First, we measure performance for unrealistic datasets.

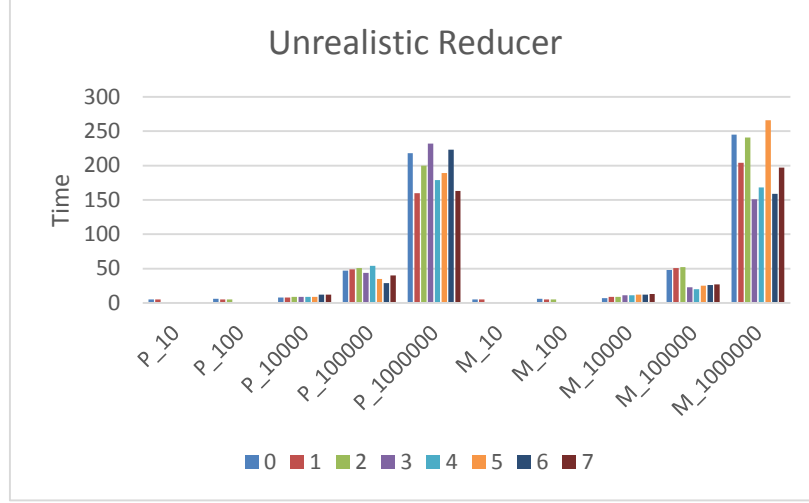


Fig 8: Reduce function comparison for unrealistic dataset

We measured performance in eight thread based reduce functions. As shown in figure, 'P_' notation is for even distribution algorithm and 'M_' notation is for Google MapReduce algorithm. For 0.1 and 1 million records, traditional MapReduce failed to distribute data in balanced manner.

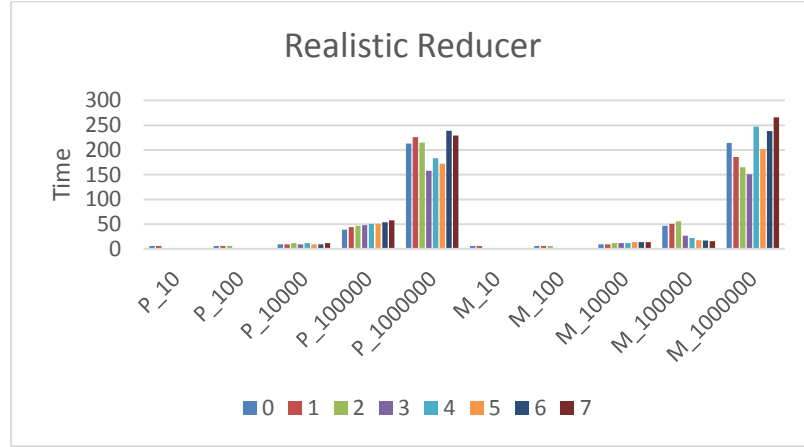


Fig 9: Reduce function comparison for realistic dataset

For realistic dataset, even distribution algorithm able to share load between other reduce functions. Even distribution of data provides effective load balancing between reduce functions which leads to overall performance improvement.

- Screen captures
For Realistic data of 100 URLs, where URL count operations is to be performed by MapReduce our programming language, the input dataset will be like:

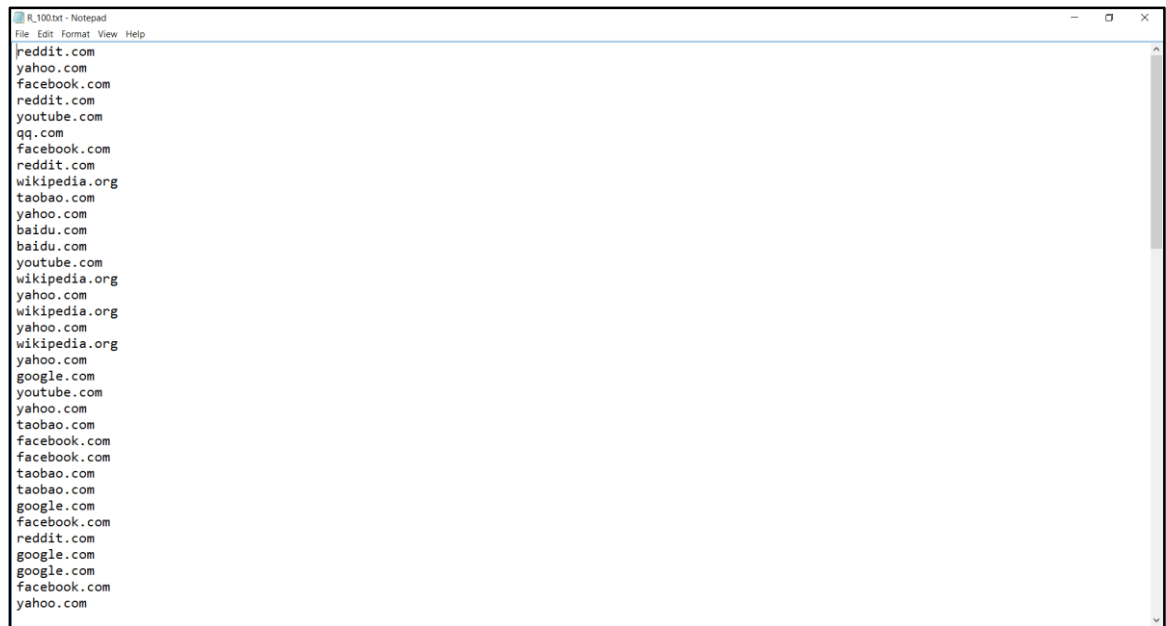


Fig 10: Input File

The input URLs are distributed among number of mappers. Here, we have taken 6 mappers. In standalone application, a thread is created for each mapper and so there are total 6 threads where every thread creates one output mapper file.

```
}
PrintWriter file = new PrintWriter("MapOutput" + count + ".txt");
StringTokenizer itr = new StringTokenizer(keys + " ");
while (itr.hasMoreTokens()) {
    String key = itr.nextToken().toString();
    file.write(key + ":" + 1);
    file.println();
}
file.close();
```

Fig 11: Mapper Intermediate file creation

As highlighted above, count is the Thread Id which will be 0, 1, 2, 3, 4 and 5 for total of 6 threads.

Like mappers, reducers also generate output files. In this case, we have set number of reducers parameter to 3 which will create 3 threads and 3 output files.

```
PrintWriter file = new PrintWriter("ReduceOutput" + count + ".txt");
```

Fig 12: Reducer Output file creation

At last, the directory structure will be like:

MapOutput0.txt	4/27/2017 10:25 A...	Text Document	1 KB
MapOutput1.txt	4/27/2017 10:25 A...	Text Document	1 KB
MapOutput2.txt	4/27/2017 10:25 A...	Text Document	1 KB
MapOutput3.txt	4/27/2017 10:25 A...	Text Document	1 KB
MapOutput4.txt	4/27/2017 10:25 A...	Text Document	1 KB
MapOutput5.txt	4/27/2017 10:25 A...	Text Document	1 KB
MapperOutputFiles.txt	4/27/2017 10:25 A...	Text Document	1 KB
R_100.txt	4/27/2017 12:45 A...	Text Document	2 KB
ReduceOutput0.txt	4/27/2017 10:25 A...	Text Document	1 KB
ReduceOutput1.txt	4/27/2017 10:25 A...	Text Document	1 KB
ReduceOutput2.txt	4/27/2017 10:25 A...	Text Document	1 KB
ReducerFinalOutput.txt	4/27/2017 10:25 A...	Text Document	1 KB
ReducerFinalOutputFile.txt	4/27/2017 10:25 A...	Text Document	0 KB
ReducerOutputFiles.txt	4/27/2017 10:25 A...	Text Document	1 KB

Fig 13: Generated files directory structure

There is also a MapperOutputFile.txt is generated which keeps track of all the mapper output file names. The content of MapperOutputFile.txt will be as shown below:

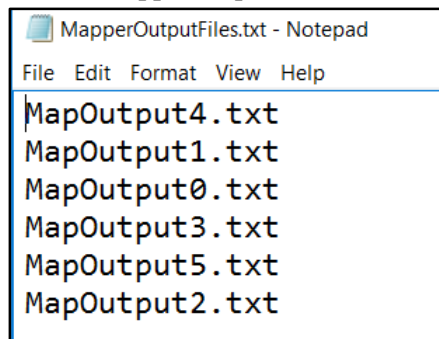
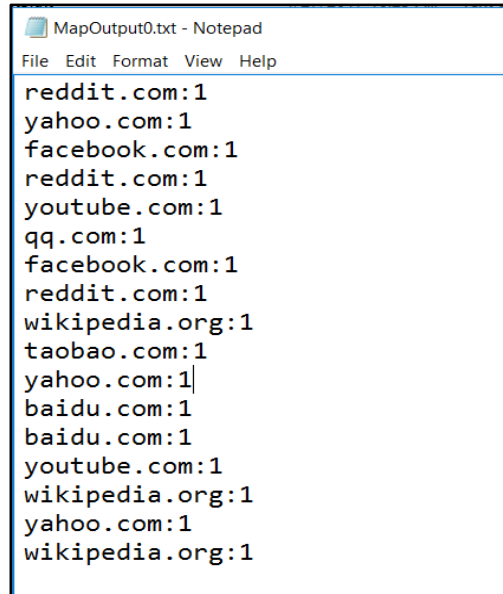


Fig 14: Mapper output file names

Now, MapperOutput0.txt file will be generated as shown below. There are 100 URLs in R_100.txt which is the input file and there are 6 Mappers which generates key-value pairs. So, each will get around 15-16 URLs to process. In this example, data separation is done based on new line.



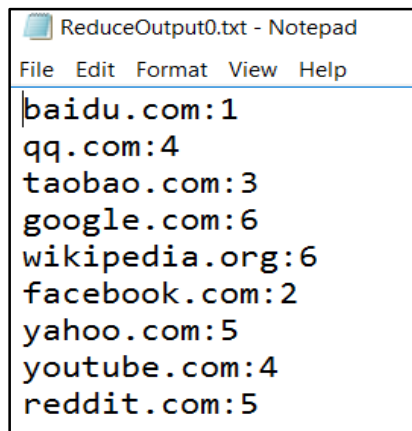
```
MapOutput0.txt - Notepad
File Edit Format View Help
reddit.com:1
yahoo.com:1
facebook.com:1
reddit.com:1
youtube.com:1
qq.com:1
facebook.com:1
reddit.com:1
wikipedia.org:1
taobao.com:1
yahoo.com:1
baidu.com:1
baidu.com:1
youtube.com:1
wikipedia.org:1
yahoo.com:1
wikipedia.org:1
```

Fig 15: Map function output

So, the content of MapperOutput0.txt is sorted and distributed among all the reducers. There are 17 URLs processed by Mapper 0. So, instead of taking a key and assigning all the key-value pairs related to one key to one reducer, the sorted output is divided into equal number of reducers and assigned the appropriate URLs.

Similarly, the content of MapperOutput1.txt, MapperOutput2.txt, MapperOutput3.txt, MapperOutput4.txt and MapperOutput5.txt are also divided equally among reducers.

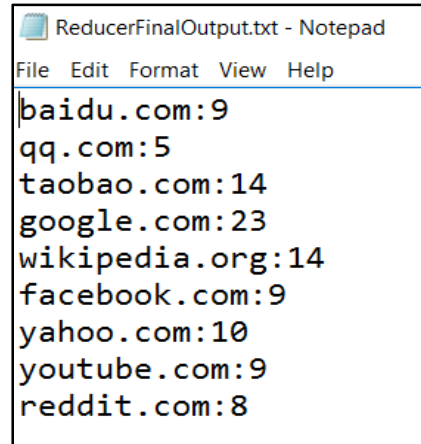
So, the ReducerOutput0.txt will be like:



```
ReduceOutput0.txt - Notepad
File Edit Format View Help
baidu.com:1
qq.com:4
taobao.com:3
google.com:6
wikipedia.org:6
facebook.com:2
yahoo.com:5
youtube.com:4
reddit.com:5
```

Fig 16: Reduce function output

And, there is one ReducerFinalOutput.txt generated which combines the count of all the reducer outputs and generates the final output.



```
ReducerFinalOutput.txt - Notepad
File Edit Format View Help
baidu.com:9
qq.com:5
taobao.com:14
google.com:23
wikipedia.org:14
facebook.com:9
yahoo.com:10
youtube.com:9
reddit.com:8
```

Fig 17: Final Reduce function output

8. Future Work

Future work for this project is to make an implementation in node-based architecture as the implementation shown here is a standalone application. In node-based architecture, the implementation should be done and then performance should be compared with traditional MapReduce.

Other work is to consider heterogeneous environment. Here, as a standalone application, threads perform the operations which are being executed on same processor. But, in node-based architecture, some nodes will be faster than others. So, an algorithm should be designed in such a way that apart from even distribution, it should also consider the processing capabilities of slower nodes.

9. References

- [1] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters.
- [2] Madhavi Vaidya. Parallel Processing of cluster by Map Reduce. In International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.1, January 2012.
- [3] Jun Liu, Tianshu Wu, Ming Wei Lin and Shuyu Chen. An Efficient Job Scheduling for MapReduce Clusters. International Journal of Future Generation Communication and Networking Vol. 8, No. 2 (2015), pp. 391-398.
- [4] Weina Wang and Lei Ying. Data Locality in MapReduce: A Network Perspective.
- [5] Xiaohong Zhang, Zhiyong Zhong, Shengzhong Feng, Bibo Tu and Jianping Fan. Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments. Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications.

- [6] Andrew Konwinski. Improving MapReduce Performance in Heterogeneous Environments. Technical Report No. UCB/EECS-2009-183.
- [7] Salma Khalil, Sameh A.Salem, Salwa Nassar and Elsayed M.Saad. Mapreduce Performance in Heterogeneous Environments: A Review. International Journal of Scientific & Engineering Research, Volume 4, Issue 4, April -2013 ISSN 2229-5518.
- [8] YongChul Kwon, Magdalena Balazinska, Bill Howe and Jerome Rolia. A Study of Skew in MapReduce Applications
- [9] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, Russell Sears. MapReduce Online. Technical Report No. UCB/EECS-2009-136
- [10] The Hadoop Distributed File System: Architecture and Design, <http://hadoop.apache.org/>
- [11] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, Ion Stoica. Improving MapReduce Performance in Heterogeneous Environments. University of California, Berkeley.
- [12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, The Google File System, Google.