4. Proposed Algorithm

4.1 Algorithm 1

Input: user files
Output: merged result
Input: Task
Output: Reduced disk access
1) Start
2) Split input file into multiple splits
3) Assign each split to each map task
4) Generate MOF
5) Build priority queue from MOF headers
6) Start shuffling when two or more MOF available.
7) Merge files by parallelism
8) Start reducing and generate final reduced tasks
9) Stop

4.2 Algorithm 2

Map Task: // one for each split
Input: Si //Split i, line = transaction
Output :< key, 1>pairs, where key is an element of candidate item set
1) for each transaction t in Si
2) Map (line offset, t) //Map Function
3) For each item set I in t // I = all possible subsets of t
4) out (I, 1)
5) End foreach
6) End map
7) End foreach
8) End
Reduce Task:
Input: <key2, value2>pairs Minimum support count where key2 is an element of candidate, Item set and value2 is its occurrence in each split
Output: <key3, value3>pairs, key3 is an element of frequent item set and value3 are its occurrences in the whole dataset
1) Reduce (key2, value2) // Reduce function
2) Sum=0
3) While (value, hasNext ())
4) Sum=value2.getNext ();
5) End while
6) If (sum>=min sup count)
7) Out (key2, sum);
8) End

5. Mathematical Model

Let Ti be a task
1) Input data is spilt into multiple splits
2) Let S be a set of split i
   S = s1, s2, s3.....si
3) Ti 2 S
   ti => Si
4) We have, Pair = key, value
   Value = occurrence in each split
   Solution criteria => minimum support count
5) Select sum such that
   T >= minimum support count
   Output = (key, T)
6) In hadoop instead of processing MOF per reduce
   Process MOF per core
7) C = No. of cores
8) R= No. of Reducers
   Number of shuffles will be
9) M * R
   M= no. of mappers
   To improve performance M * R should be less
10) Performance is inversely proportional to M*R
   No. of disk access = 1 / M*R

6. Experimental Setup

For conducting the experiment we have installed Ubuntu 12.04 on machine. And second machine may have any OS with. Ubuntu machines having openjdk1.7 installed in it and SSH enabled. Hadoop 1.2.1 have been configured on machine. Hadoop plugins are used to configure eclipse.

The Name Node is center piece of Hadoop in light of the fact that it controls the entire Data Nodes exhibit in bunch. The Data Nodes contain all the information in group on which we will work our MapReduce projects and perspective the movement information from different points of view. Job Tracker controls every one of the assignments which are running on Task Trackers demonstrated in taking after.

A HDFS cluster comprises of a solitary Name Node, a master server that deals with the file system framework namespace and manages access to records by customers. Moreover, there are various Data Nodes, generally one for every node in the bunch, which oversee capacity connected to the nodes that they keep running on. HDFS uncovered file system framework namespace and permits client information to be put away in records. Inside, a file system is split into one or more pieces and these blocks are put away in a situated of Data Nodes.

Remote Direct Memory Access (RDMA permits computers in a network to exchange information in primary memory without including the processor, reserve, or working arrangement of either PC. Like by regional standards based Direct Memory Access (DMA), RDMA enhances throughput and execution on the grounds that it authorizes assets. We have developed word count MapReduce programs which take the input as a text file and calculate the word counts.

6.1 Module 1 [Locally Submitted Job]:

In this module a job is submitted to the hadoop system. Before submitting a job i.e. text file is splits into three splits of same size. These splits are now processed by map reduce program parallelly.

6.2 Module 2 [Remote Submission of Job]:

In this module a job is submitted to the hadoop system by remote machine using RDMA protocol. Hadoop is pure java based framework. So to provide multiplatform connectivity RDMA protocol is used which overcomes interconnectivity

rule of TCP/IP. Submitted job is processed by hadoop system and the results are provided to remote client.

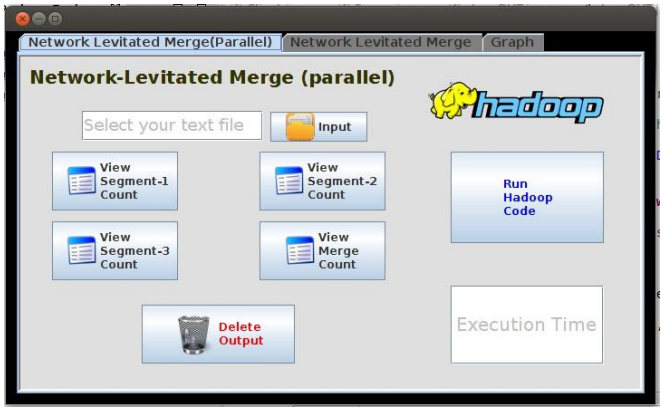

Figure 3: Module 1

Figure 4: Module 2

7. Results

Our experiment is conducted on machines containing hadoop 1.2.1 configuration with core 2 duo and i3 processers. Proposed scheme provides improved results than existing system. To find the results our experiment is conducted on word count program. The data table shows inputs used.

Table 1: Input Dataset

Sr. No.	Data Size	Execution time (in ms)	
		Proposed	Existing
1	1 MB	0.010	0.081
2	5 MB	0.015	0.100
3	10 MB	0.020	0.057

Proposed results may vary according to processer and memory availability. As data size grows the performance will improve.

The graph shows difference between code 1 and code2. Code 1 is existing system and code 2 shows execution time of proposed system.

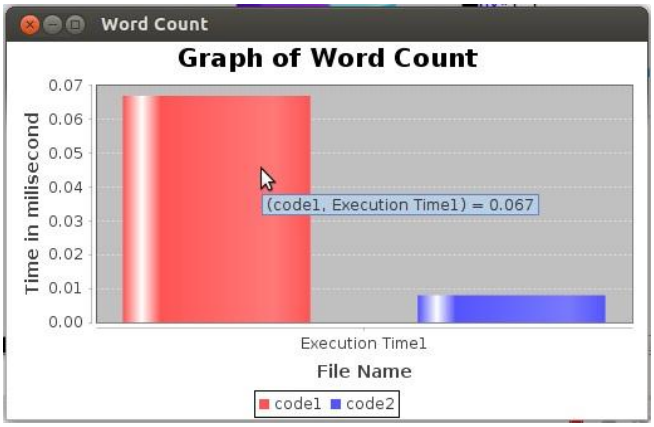

Figure 5: Results

8. Conclusion

We have conducted the experiment on different machines. Particularly, our analysis has focused on data processing inside Reduce-Tasks. We reveal that there are several critical issues faced by the existing Hadoop implementation, including its merge algorithm, its pipeline of shuffle, merge, and reduce phases, as well as its lack of portability for multiple interconnects. A parallel shuffling in map and reduce phase increases the speed of execution by reducing the number of disk accesses. In future security constraint requires to be covered. Security becomes another important issue. If we use strong encryption technique to encrypt data from client and server, then system will become complete.

References

[1] Weikuan Yu, Member, IEEE, YandongWang, and Xinyu Que, Design and
[2] Evaluation of Network-Levitated Merge for Hadoop Acceleration IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS
[3] Dawei Jiang Beng Chin Ooi Lei Shi Sai Wu, The Performance of MapReduce: An Indepth Study Proceedings of the VLDB Endowment, Vol. 3, No. 1
[4] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein MapReduce:Online Yahoo! Research
[5] Apache Hadoop Project, http://hadoop.apache.org
[6] Hadoop- The Definitive Guide, 3rd Edition by Tom White.
[7] R. Recio, P. Culley, D. Garcia, and J. Hilland, An rdma protocol specification (version 1.0), October 2002.
[8] J. Dean and S. Ghemawat, Mapreduce: Simplified data processing on large clusters, Sixth Symp. on Operating System Design and Implementation (OSDI), pp. 137150, Dec. 2004.
[9] Y. Mao, R. Morris, and F. Kaashoek, Optimizing mapreduce for multicore architectures, MIT, Tech. Rep. MIT-CSAIL-TR2010-020, May 2010.
[10] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, Improving mapreduce performance in heterogeneous environments, in 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings. USENIX Association, 2008, pp. 2942.

[11] X. Que, Y. Wang, C. Xu, and W. Yu, Hierarchical merge for scalable mapreduce, in Proceedings of the 2012 workshop on Management of big data systems, ser. MBDS 12. New York, NY, USA: ACM, 2012, pp. 16.
[12] R. Chen, H. Chen, and B. Zang, Tiled-mapreduce: optimizing resource usages of data-parallel applications on multicore with tiling, in Proceedings of the 19th international conference on Parallel architectures and compilation techniques, ser. PACT 10. New York, NY, USA: ACM, 2010, pp. 523534.
[13] Kishorkumar K. Shinde, Prof. Venkatesan N., Review on Data merging and Data movement to accelerate Hadoop performance, International Journal Of Engineering And Computer Science (IJESC), Volume 3 Issue 12 December 2014
[14] Network Levitated Merge for Hadoop Acceleration with RDMA Accelerated with Parallel Data Shuffling, C-PGCON Fourth post graduate conference at MITBKC, Nashik.