International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2013): 4.438

Parallel Data Shuffling for Hadoop Acceleration with Network Levitated Merge and RDMA for Interconnectivity

Kishorkumar Shinde[1], Venkatesan N.[2]

[1,2] Savitribai Phule Pune University, Department of Computer Engineering, SKN Sinhgad Institute of Technology and Science, Lonavala, Pune, Maharashtra, India

[2] Professor, Savitribai Phule Pune University, Department of Computer Engineering, SKN Sinhgad Institute of Technology and Science, Lonavala, Pune, Maharashtra, India

Abstract: ***Performance is measure issue in today's hadoop framework. The execution time required for Map reduce model is depends on multiple factors. Shuffling and merging in map reduce requires much amount of time. Proper implementation of shuffling and merging improves the performance of overall system. With this Serialization, multiple interconnect issues are also covered in this paper. Serialization keeps reduce phase to wait; repetitive merges requires multiple disk access and lack of portability for different interconnections. Repetitive merges can be reduced by network levitated merge algorithm, Serialization issue is overcome by parallelization. RDMA is used to for multiple interconnects. A non Hadoop and non java machine can also use the hadoop features. If we use pipelining to avoid serialization some sort of serialization is there in shuffle and merge phase. In pipelining output file is shuffled and merged before providing it to reduce task. Instead of pipelined shuffling, parallel shuffling is proposed. This reduces the number of disk accesses resulting in improved performance.***

Keywords: Hadoop, Network levitated merge, MapReduce, Big- data, RDMA.

1. Introduction

In the present data age the prerequisite of information is expanding step by step. The information produced from distinctive sources is in terabytes every day, which is called right now Big Data. BIG DATA is huge in size, as well as data of diverse varieties, distinctive sizes and at distinctive pace. This enormous information is utilized for distinctive application or business related administrations like business insight and some more. To store and procedure this expansive measure of information we require an effective and tolerant framework. Google built up a file system to handle enormous information called as GFS. Hadoop is in light of the Googles document framework. Hadoop is free and open source programming structure for putting away and handling huge information productively. It is made in java programming language. HDFS (Hadoop Distributed File System) and MapReduce are the two parts of Hadoop. Map reduce is implementation for execution of Hadoop for distributed computing and cloud computing. Map reduce is a programming model to compose applications for processing big data. Hadoop is used by numerous associations like Yahoo, Google, Facebook and it is kept up by Apache Foundation.

Map-reduce are implemented with the help of two components: a job tracker and multiple task trackers. The job tracker is responsible to command the task trackers through two main functions i.e. map tasks and reduce tasks, the task trackers used to process data as per the commanded by job tracker. Job tracker is also in-charge of scheduling map task and reduces task to task trackers, it assigns job to the task trackers and also collects the intermediate results. Hadoop has name node and data node to manage and process data. Name node is node which stores file system metadata, and data node is actually store the data.

1.1. MapReduce Programming Model

A MapReduce library requires the programmer to cast the computation in the form of two functions, Map and Reduce. The library partitions the input into a number of splits, and calls Map on each split. Phoenix is a MapReduce library for multi-core and multi-processor systems [8]. The library typically has a pool of Map worker threads, one per core, that repeatedly take a split from a work list and call Map. Each Map calls output is a set of intermediate key and value pairs. When all the splits have been processed by Map, the library calls reduce once for each distinct key produced by the Map calls, passing Reduce the set of all values produced by the Maps for that key. Again, the library has a pool of Reduce worker threads, one per core. Each Reduce generates a set of output key and value pairs, and the libraries Merge phase sorts them by key to produce the final output. The programmer must ensure that the Map and Reduce functions have no side effects. The charm of MapReduce is that, for algorithms that can fit that form, the library hides all the concurrency from the programmer. For example, one can count the number of occurrences of each word in a body of text as follows. The Word Count Map function parses the input, generating an intermediate key/value pair for each word, whose key is the word and whose value is 1. The Reduce function (which the library calls once per distinct word) emits a key/value pair whose key is the word, and whose value is the number of values. The library gets parallel Speedup by running many Maps concurrently, each on a different part of the input file, and by running many Reduces concurrently on different words. The Merge phase combines

and sorts the outputs of all the Reduces [7]

1.2. Serialization

Hadoop has a serialization problem. Map task starts processing when input data is splits into multiple files. After processing of map task is done, these MOF (Map Output Files) are gained by reduce task. But reduce task starts processing when all the segments of MOF are available. This situation creates a serialization between shuffles and merges which is shown in figure

1.3. Interconnectivity

Hadoop is developed in java language. Hadoop supports only TCP/IP as a transport protocol. It does not support other transport protocols such as RDMA. Many popular systems using RDMA as transport protocol. RDMAP provides read and write services directly to applications and enables data to be transferred directly into ULP Buffers without intermediate data copies. It also enables a kernel bypass implementation.[6] Such a lack of portability in hadoop will prevent it from latest technologies. To overcome this limitation hadoop architecture is modified to hadoop acceleration frame work known as Hadoop-A [1].

In hadoop reduce task is start reducing after all the data is merged together. So this produces the serialization in map task and reduce task. Figure shows the serialization in hadoop. Many studies have been carried out to improve the performance of Hadoop. Yu [1] proposed new merging algorithm and a new framework of Hadoop. Jiang [2] identified the four factors that have effect on map reduce performance. Condie [3] proposed a direct channel between map tasks and reduce task.

2. Related Work

At this very moment is developing step by step the administration of information is turning into a basic issue. The data turns into a Big Data. Big Data is enormous in size as well as contains information of distinctive size, form, and from diverse sources. As an illustration New York stock trade produce information in Terabytes a day, prominent informal communication site Facebook transfers a huge number of pictures and features every day whose size in Terabytes. This information may require for future references or some business insight reason for some associations. Existing database administration framework is not equipped for dealing with this substantial measure of information. It has numerous reasons like information security, sensibility, transportability, recuperation from disappointment and so forth which prompts the need of viable and flaw tolerant framework. Googles new file system known as Google file system for his own purpose of big data management. On basis of Googles file system the Hadoop HDFS and Map Reduce is designed. Hadoop is kept up by Apache [4] Foundation and it is used by any associations like Yahoo, Facebook.

Existing Hadoop framework has numerous execution and security issues. Hadoop has two fundamental parts map and reduce. The information which is supplied by customer to Hadoop framework is isolated into various parts. Every split is relegated to each map task undertaking. Map task generates the key value pair of input data. The mappers job is map input key value pair to intermediate key value. Mapper transforms the input record to intermediate record. The number of maps are depends upon number of input blocks. This intermediate data is supplied to reduce task. Reduce task merges these key value pair into single value. During the map task the data is sorted according to user query and shuffled. We can improve performance with various factors like merging, data I/O etc. Many studies have been carried out to improve the performance over existing system. Numbers of different improvements are made.

Yu [1] suggested many changes to existing system. He invented a new algorithm network levitated merge. Reduce task fetches intermediate data i.e. output of map task and store it locally onto memory. This leads to multiple disk access and many I/O operations which require more execution time. New network levitated merge overcomes this problem. In this algorithm instead of fetching whole segment only small header is fetched.

In this paper they also proposed a pipelined structure of Hadoop shuffle, merge and reduce phase which overcomes the serialization barrier between reduce and merge phase as there is pipeline structure so without waiting reduce task directly fetches the header. Due to this intermediate data is fetched as map out file is generated, this decreases the execution time.

Existing Hadoop system doesn't have support for different network interconnects. It only supports TCP/IP transport protocol it has no support for RDMA i.e. Remote Direct Memory Access which good in high performance communication. In Hadoop-A two new plug-in components are added MOF-Supplier and NET-Merger to support RDMA capable interconnects. These two components have C implementation which allows choice of connections like RDMA. This framework is optional i.e. user can enable or disable this framework by setting a parameter.

Jiang [2] identified four important elements that notably effect on performance of Hadoop which are I/O mode, Indexing, Parsing, and Sorting. Map function doesn't take input directly from storage, a mediator called as reader is used. A reader takes data from storage and put it into map buffer. There are two methods for reading data i.e. direct I/O and streaming I/O. Streaming is mainly used for nodes which are placed locally or remotely. And direct I/O used for local nodes. Existing Hadoop uses only streaming I/O method no matter the node locally placed or remotely placed. And experiment shows that streaming I/O has 15% more performance than direct I/O method. Map- reduces uses range indexing scheme that uses same size data chunk to create data index. Next factor is parsing i.e. means when reader reads data from storage to buffer then there is need to convert this raw data into set of record having key/value pair. In this the fields from value part needs decoding into

appropriate type. This decoding has two types immutable and mutable. Immutable decoder is much slower than mutable decoder. Next important factor they considered is sorting and merging. Sorting and merging of data is important task in map reduce. A fingerprint mechanism is proposed to improve the sorting of keys, which reduce key comparisons to improve performance.
XinyuQue [10] proposed Hierarchical Merge [10] for reduction of the memory buffer usage for Hadoop-A and to enable scalable data processing.

Condie [3] designed an architecture which directly connects map and reduce tasks. The intermediate data is pipelined within different operators. They designed a new prototype known as Hadoop online prototype [3]. Zaharia [9] designed a new scheduling algorithm, Longest Approximate Time to End (LATE) [9], which is highly robust to heterogeneity. Tiled-Map Reduce (TMR) [11] divides a large MapReduce job into a number of small sub-jobs and repeatedly processes one sub job at a time with efficient use of resources and at last merges the results of all sub-jobs for output

3. Proposed Scheme

The basic idea is to improve the performance by adding parallelization in shuffling phase. The performance is depends on many factors. Shuffling and Merging are the two important factors. Repetitive merging requires multiple disk access which can degrade the performance. Here the two different modules are proposed. In first module the job submission is done locally at server side. Here processing of data and results are shown to user at user side. In second module the RDMA protocol is used to transfer data from one machine to another machine. Use of RDMA is to overcome platform interconnect limitation. The job is submitted by client from remote side. Submitted job will be processed by server i.e. machine containing hadoop framework. Here there is no need for client to configure hadoop on his machine. Client needs to provide servers IP address containing hadoop framework.

3.1 Shuffling

Shuffling is process in which system performs sorting and rearranging of files and transferring it to reduce task. In Hadoop shuffling is done by a special shuffle handler [5] or an additional service for handling shuffle process. In pipelined shuffling as map files are created, reducer can access it and start building priority queue. The process of shuffling and merging is pipelined. In this case if shuffling and merging is done in parallel fashion the number of disk access is again decreases resulting high performance.

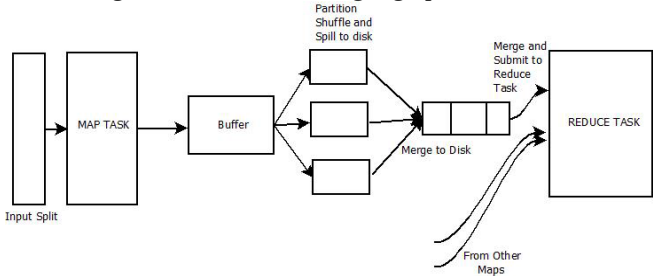

Figure 1: Shuffling Process

3.2 Merging

It is process where all the map files are merged and stored to disk. Map generates output files i.e. key, value pair. Each time a map generates an output file it is stored to disk, when all files are completed by map task the merging process is started. Every time map creates output file it incurs disk access, and for multiple files requires multiple disk access. In network levitated merge [1] instead of accessing the whole output files from disk directly, only a header file is accessed. The header files are lightweight so no need to store them to disk, which can reduce disk access.

Multiple merging can be avoided using network levitated merge algorithm. Map output files which are generated by map task are stored locally on buffer. The buffer size is 100Mb and it can be tuned. After buffer completely fills the files from buffer are transferred to disk. Each map generates different output files. After all output files are generated, these are shuffled and passed to reduce task. In pipelined method for every map task file a separate shuffling and merging is done which avoids serialization but number of disk accesses is still high. This separate shuffling will reduce the performance but in proposed system a shuffling is done in parallel fashion. After output files are generated by map task these are levitated to remote disk and all output files are shuffled in parallel fashion

In Algorithm 1 the input file is split into multiple splits and each split is provided each separate map task. The map task generates MOF which are stored on disk. When reduce task become free these MOF are provided to the reduce task. Here merging and shuffling is working in parallel.

3.3 Hadoop Acceleration Framework

Hadoop supports only TCP-IP as a transport protocol. To support transport protocols like RDMA a new framework is needed. Hadoop acceleration framework [1] supports multiple interconnects. Original hadoop is kept as it is. A new model is used which is developed in C language. It has two main components MOF supplier [1] and Net Merger [1]. These are remotely operates and provides their output to original hadoop system. So any technology using other than TCPIP as a transport protocol can able to process with hadoop.

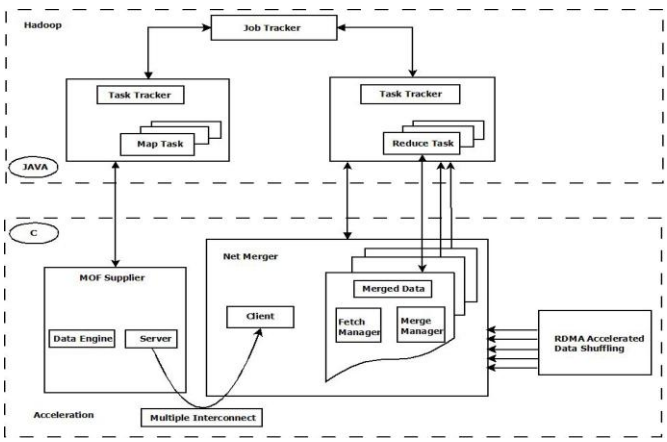

Figure 2: System Architecture

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.

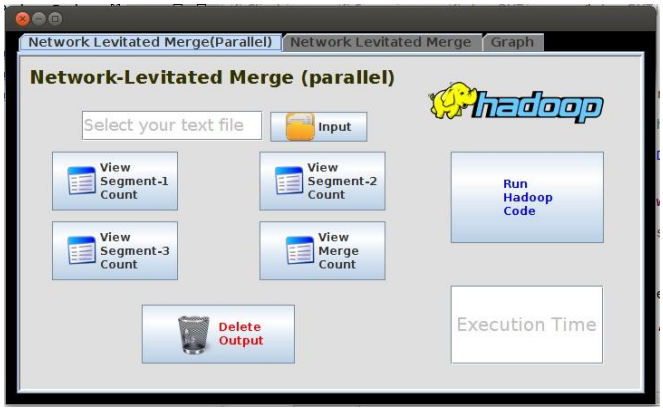

Figure 3: Module 1

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.

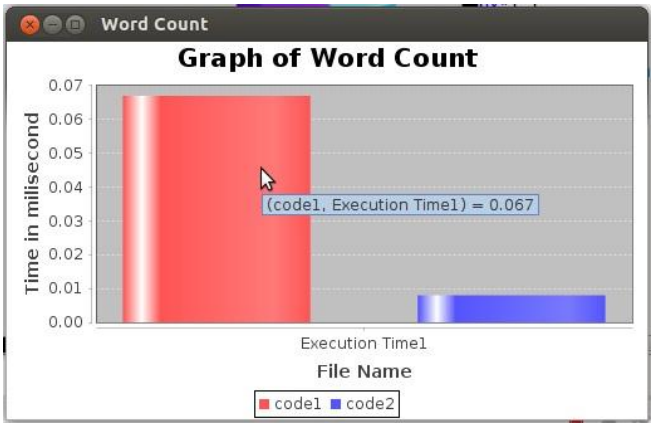

Figure 5: Results

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.