IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-8, Aug 2019 E-ISSN: 2347-2693

Straggler Problem –Tail Latancy in Distributed network

Md. Nesar Rahman[1], Ayesha Siddika[2], Muhammad Shafiqul Islam[3], Md. Shahajada[4]

[1]Department of ICT, Bangladesh University of Professionals (BUP), Bangladesh
[2]Department of CSE, World University of Bangladesh (WUB), Bangladesh
[3]Department of ICT, Bangladesh University of Professionals (BUP), Bangladesh
[4]Senior Database Engineer, eGeneration Limited, Bangladesh

DOI: https://doi.org/10.26438/ijcse/v7i8.168178 | Available online at: www.ijcseonline.org

Accepted: 12/Aug/2019, Published: 31/Aug/2019

***Abstract*-** Distributed processing frameworks split a data intensive computation job into multiple smaller tasks, which are then executed in parallel on commodity clusters to achieve faster job completion.
A natural consequence of such a parallel execution model is that slow running tasks, commonly called stragglers potentially delay overall job completion. Stragglers in general take more time to complete tasks than their peers. This could happen due to many reasons such as load imbalance, I/O blocks, garbage collections, hardware configuration etc. Straggler tasks continue to be a major hurdle in achieving faster completion of data intensive applications running on modern data-processing frameworks. The trouble with stragglers is that when parallel computations are followed by synchronizations such as reductions, this would cause all the parallel tasks to wait for others meaning that the parallel runtime is dominated by the slowest performing straggler.
In a large-scale distributed system comprising a group of worker nodes, the stragglers' delay performance bottleneck, is caused by the unpredictable latency in waiting for slowest nodes (or stragglers) to finish their tasks.
Such stragglers increase the average job duration by 52% in data clusters of Facebook and Bing even after these companies using state of the art straggler mitigation techniques[1]. This is because current mitigation techniques all involve an element of waiting and speculation. Existing straggler mitigation techniques are inefficient due to their reactive and replicative nature – they rely on a wait speculate- execute mechanism, thus leading to delayed straggler detection and inefficient resource utilization. Hence, full cloning of small jobs, avoiding waiting and speculation altogether is proposed in a system called as Dolly. Dolly utilizes extra resources due to replication.

Keywords **–** Distributed network, latency, straggler detection, data clusters, slowest performing straggler

I. INTRODUCTION

One of the main causes of performance problems in distributed data processing systems (from the original MapReduce to modern Spark and Flink) is “stragglers”. Stragglers are parts of the input that take an unexpectedly long time to process, delaying the completion of the whole job, and wasting resources that stay idle. Stragglers can happen due to imbalance of data distribution or processing complexity, hardware/networking anomalies, and a variety of other factors.

Google Cloud Dataflow is the first system to address the problem of stragglers in a fully general way. By dynamically redistributing parts of already launched work from straggler workers onto idle workers to maximize utilization, Google Cloud Dataflow is able to preserve data consistency and minimizing re-execution.

In all major distributed data processing engines — from Google's original MapReduce, to Hadoop, to modern systems such as Spark, Flink and Cloud Dataflow — one of the key operations is Map, which applies a function to all elements of an input in parallel called ParDo in the terminology of Apache Beam (incubating) programming model. All of these frameworks execute a Map step by splitting the specification of its input into parts (often called shards or partitions), of roughly equal size, and then reading/processing data in each part in parallel. For example, input specified by a glob file pattern might be split into tuples; input from a key-value storage system such as Big table, tuples. In all such frameworks, except Cloud Dataflow, the splitting is done upfront, before starting to execute any of the shards, and doesn't change during execution. The number of shards is usually either specified by the user or determined by the system heuristically, e.g., based on an estimate of the data size or just on the number of input files.

A very common performance problem in Map steps, faced by all frameworks, is stragglers — when a small number of shards take much longer to be processed than the rest. Stragglers can easily dominate the runtime of the map step, defeating much of the parallelization benefits. They can also

waste resources and increase costs as other workers have to stay idle for long periods of time before they can start working on the next stage. Stragglers are recognized, both in the industry and in academia, as one of the main causes of poor performance of data processing jobs. Stragglers can have a variety of causes: an abnormally slow worker; some parts of data being particularly computationally intensive to process; unbalanced splitting of the input where the amount of data in different parts turns out to be drastically different. This last case happens rarely when processing files, but is quite frequent with key-value stores and other complex inputs where not enough information is available about the data distribution to split the input evenly in advance.

Another class of issues is when the amount of data is balanced, but the processing complexity is not. For example, we can imagine a pipeline which takes as input a file containing *filenames* of videos to be transcended. It's not enough to give the same *number* of videos to each worker to ensure a balanced workflow, as the videos can be of very uneven length. As one of the essential factors of system and network performance, latency indicates how fast a user can get a response after the user sent out a request. Low latency, which means systems response quickly to actions, can make users feel more natural and fluid than long response time[3].

In the financial market, as more and more business trades and banking operations are executed online, lower latency now means more revenues, especially for companies which adopt high frequency trading to earn huge profit. High frequency trading means to rapidly trade large volumes of securities by using automated financial tools. A millisecond decrease in a trade delay may boost a high-speed firm's earnings by about 100 million per year, and also helps a firm to gain great competition advantage. Traditionally, financial organizations can achieve low latency via adopting high performance computers, which provide great processing capability, especially the capability of floating-point processing. When the processing capability is not enough, high performance computers can also be scaled via two methods, which are scale up (adding more CPUs or memory to a single computer) and scale out (adding more computing nodes, and connecting them with high performance interconnects). However, as the size of data needed to be analyzed is growing dramatically in the last few years, the primary bottleneck has shifted to the performance of storage system, and the frequent data movement in traditional high performance computing can significantly impact the latency when the volume of processing data is huge. Therefore, the system architecture for financial computing needs to be improved in such a situation. Such data explosion problem can also be called as the big data problem, which has been a hot trend in recent years.

Objectives

a. To identify the cause of Straggler problem in the Map Reduce frame work.
b. To analyze how changes in configuration of the system especially memory, CPU, Storage added to the tail latency problem in distributed environment.
c. Analyze how different configuration of nodes in a distributed system like Hadoop performs while performing big data processing and analyzing.
d. To study the straggler problem effect in distributed framework.

II. LITERATURE REVIEW

Straggler is very known issue in parallel computing, and many techniques have been developed to mitigate them. As the data scale increases, the communities of architecture, systems and data management pay more attention on developing new big data systems to satisfy requirements from different areas. The interactive nature of modern web applications necessitates low and predictable latencies because people naturally prefer fluid response times[6]. The growing data volume makes applications more complex and diverse. The increasing adoption of commercial clouds to deliver applications further exacerbates the response time unpredictability. In these environments, almost each and every application almost unavoidably experience performance interference due to contention for shared resources (like CPU, memory, and I/O)[7].

Most distributed applications of iterative convergent algorithms follow the Bulk Synchronous Parallel (BSP) computational model which uses an input-data-parallel approach. The input data is divided into worker node that execute in parallel, each node perform the work associated with their input data, and execute synchronizations barrier at the end of each iteration. The model parameters are stored in a shared data structure in distributed nodes and all nodes update during each iteration. BSP guarantees that all workers see all updates from the previous iteration, but not the updates from the current iteration, which enable the leaf nodes to use cached copies for efficiency. The assignment of leaf nodes remain the same from one iteration to the next to avoid the overheads of input data movement.

For BSP, stragglers are major performance issue. Because in BSP in each iteration, all leaf nodes must wait for the slowest node to complete its task. If one at least one leaf node will run unusually slowly in a given iteration which is a common scenario the straggler problem grows in parallelism. Even when it is a different straggler in each iterations, due to uncorrelated transitory effects, the entire application can be slowed significantly.

Stragglers can occur for a number of reasons[8][9] including hardware heterogeneity[10][11], hardware failures[9],

unbalanced data distribution among tasks, garbage collection in high-level languages, and various OS effects[12][13]. Even temporary latency spikes from individual nodes may ultimately dominate end-to-end latencies. In shared cloud infrastructures resource contention is another common cause of straggler problem. Besides, expensive stopping criteria computations can lead to straggler effects, when performed on a different leaf node.

The HPC community runs applications using the BSP model frequently. It gives significant effort to identifying and remove sources of performance jitter based on the hardware and OSs of their supercomputers[13][14]. Naiad used the same approaches. While this approach can be effective at reducing performance "jitter" in specialized and dedicated machines[2]. It does not solve the more general straggler problem. For instance, it does not work for today's multi-tenant computing infrastructures, it is not applicable to programs written in garbage-collected languages, does not handle algorithms that inherently cause stragglers during some iterations[15][16][17]

Blacklisting is another approach to mitigate certain straggler performance issue by ceasing to assign work to workers that are falling behind. However, this approach is fragile. Stragglers caused by temporary slowdowns, the reason could be due to resource contention with a background activity which often occur on non-blacklisted machines[5]. It can even blacklist a good performing worker node which worsens the performance further.

Dean and Barroso described techniques employed at Google to tolerate latency problem[5]. They have developed short-term adaptations in the form of request reissues, along with additional logic to support prevention of duplicate requests to reduce unacceptable additional load. In DSPTF[18], a request is forwarded to a server. If the server has the data in its cache, it will respond to the query. Otherwise, the server forwards the request to all replicas, which then make use of cross-server cancellations to reduce load[5].

Pisces is a multi-tenant key-value store architecture that provides fairness guarantees between tenants[19]. It is concerned with fair-sharing the data-store and presenting proportional performances to different tenants.
Priority Meister focuses on providing tail latency QoS for bursty workloads in shared networked storage by combining priorities and rate limiters[20].

Speculative execution is used to mitigate stragglers in data processing systems like MapReduce, Hadoop, and Spark[8][9]. Jobs in these systems consist of stateless, idempotent tasks like "map" and "reduce", and speculative execution runs slow tasks redundantly on multiple machines. While this consumes extra resources, it can significantly reduce job completion delays caused by stragglers, because the output from the first instance of any given task can be used without waiting for slower ones.
Work stealing and work shedding are mirror approaches for adaptively rebalancing work queues among workers. The concept is to move work from a busy worker to an idle worker. FlexRR carefully avoids data movement by limiting and pre-determining reassignment patterns to avoid expensive on-demand loading of input data and parameter state. It is designed explicitly to work in conjunction with flexible consistency bounds.

Hadoop Tools

The below list are the tools that are available in Hadoop DataNode

- Hadoop Distributed File System (HDFS)
- MapReduce
- Hbase
- Spark
- Hive
- Impala
- Sqoop
- Pig
- ZooKeeper
- NOSQL
- Mahout
- Solr
- Avro
- Oozie
- Flume
- Clouds
- Ambari
- MongoDB

III. TESTING TOOLS

A Details overview of the technology that we are using in our experiment to find tail latency is given below. The Three most used tools of Cloud era Hadoop are:

- HIVE
- IMPALA
- HUE

IV. METHODOLOGY

Our goal is to evaluate the performance of MapReduce job using Hadoop tools like HDFS, Hive, Impala and Hue and try to find how different node with varied configuration of RAM, Memory, Processor and Storage perform a MapReduce job. We also run parallel processing to find out how performance varied due to multiple job execution in

parallel. To achieve this goal, we have prepared 4 machines where one is Master Node and three Slave nodes. In the Master node also called namenode in Hadoop, we have installed Clouder Manager 5.16 and other necessary tools like, HDFS, HIVE, IMPALA, HUE, HBASE, SPARK and YARN MANAGER. In the 3 slave node, also called datanode in hadoop, we have installed HDFS, Hive and Impala only. We have also made necessary network configuration to communicate with all the 4 nodes smoothly using password less SSH Private and Public Key.

We have uploaded 350 GB of data in HDFS file system and loaded these records in HIVE using HIVE tools so that we can run HIVE query language in the records. After loading these data in HIVE, we have performed different experimental query and map/reduce job in the machines and evaluate the output to find tail latency while executing the job.

V. SETUP OF THE STUDY

First of all, we have setup a 4 node hadoop cluster in 4 machines with variant hardware configuration. To install Hadoop, we have to perform the below task:

At first, we install Ubuntu 16.04 Linux operating system in all four machines. Then we have made up necessary configurations like network in all four machines, firewall configuration, creating user with appropriate permission, Password less SSH, and necessary software and updates etc. Then we have installed the Cloud era Manager 5.16 of Hadoop in one of my good machine that have good configuration. We have made one Master and three Slave Machine. The Master machine's is called Name Node and Slave machine's is called Data Node Hadoop. Then we have loaded 350GB data in HDFS file system to perform my experiment.

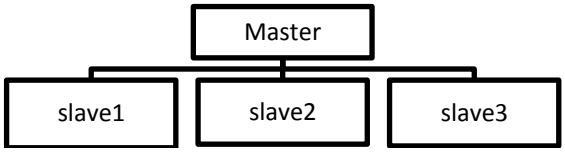

Figure 1: Master-Slave Architecture in Hadoop

Sample data tables

Table 1: Sample Data for performing Map Reduce Job

Country	*Item Type*	*Units Sold*	*Unit Cost*	*Total Revenue*	*Total Cost*	*Total Profit*
South Africa	Fruits	1593	6.92	14862.69	11023.56	3839.13
Morocco	Clothes	4611	35.84	503890.1	165258.2	338631.84
Papua New Guinea	Meat	360	364.69	151880.4	131288.4	20592
Djibouti	Clothes	562	35.84	61415.36	20142.08	41273.28
Slovakia	Beverages	3973	31.79	188518.9	126301.7	62217.18
Sri Lanka	Fruits	1379	6.92	12866.07	9542.68	3323.39
Seychelles	Beverages	597	31.79	28327.65	18978.63	9349.02
Tanzania	Beverages	1476	31.79	70036.2	46922.04	23114.16
Ghana	Office Supplies	896	524.96	583484.2	470364.2	113120

In order to find tail latency, we have performed map reduce job by changing the configuration of the Hardware of the three Data node.

Experiment: 1

Table 2: Hardware configuration of Nodes for experiment: 1

Node Name	*RAM*	*No of Processor*	*Storage in used*	*Parallel Processing*
Datanode1	8 GB	2	350 GB	1
Datanode2	8 GB	2	350 GB	1
Datanode3	8GB	1	350 GB	1

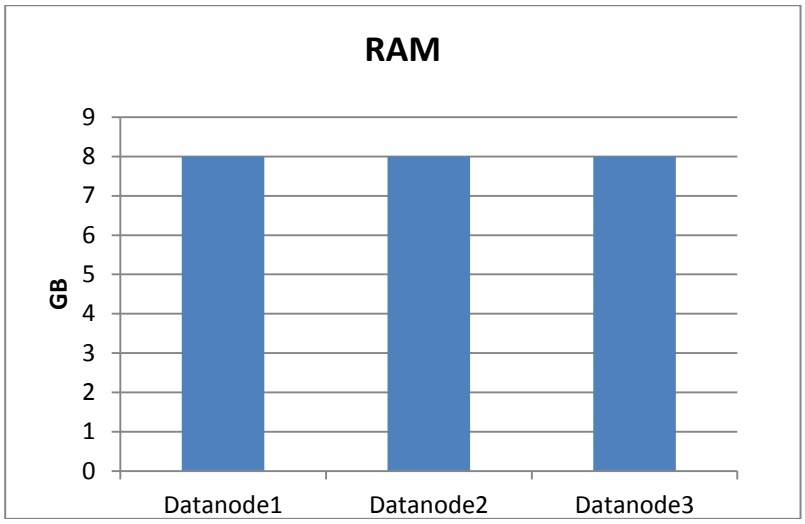

Figure 1: RAM configuration of Nodes for experiment 1

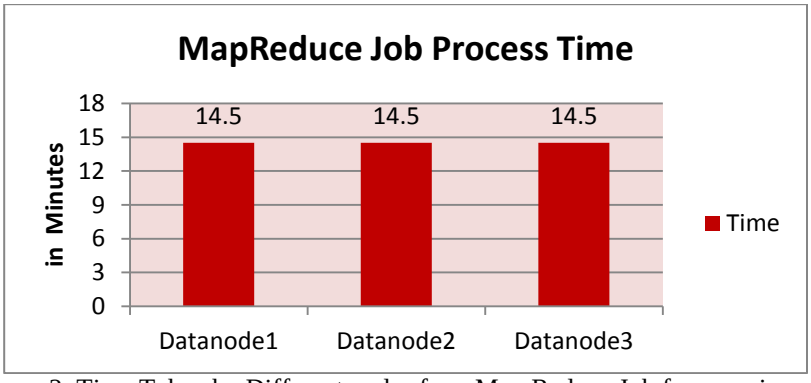

Figure 2: Time Taken by Different nodes for a Map Reduce Job for experiment 1

In this experiment, we have kept same configuration Datanode1, Datanode2and Datanode3 has only 1 processor. We have found that all node perform the task in same time despite datanode3 has different configuration, so no latency found in this experiment.

Experiment 2

Table 3: Hardware configuration of Nodes for experiment 2

Node Name	*RAM*	*No of Processor*	*Storage in used*	*Parallel Processing*
Datanode1	8 GB	2	350 GB	1
Datanode2	8 GB	1	350 GB	1
Datanode3	6 GB	2	350 GB	1

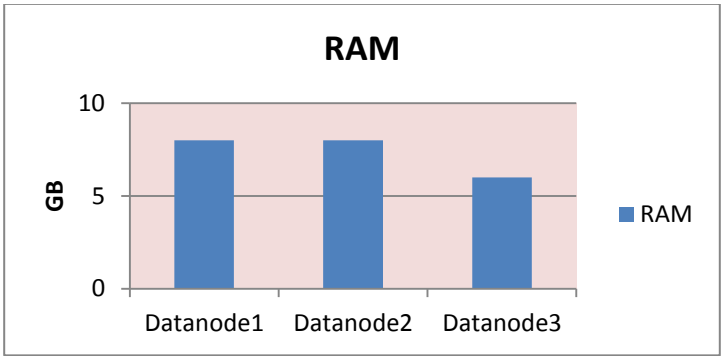

Figure 3: RAM configuration of Nodes for experiment 2

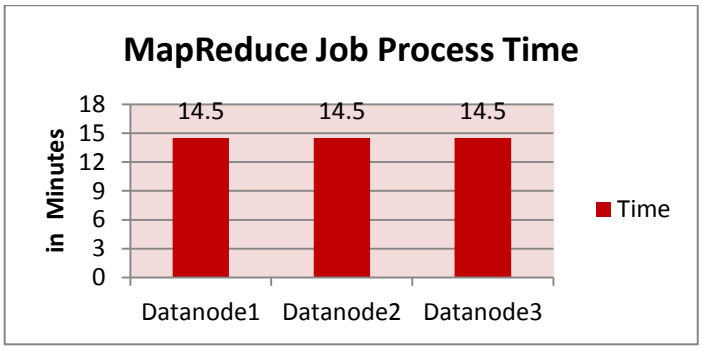

Figure 4: Time Taken by Different nodes for a Map Reduce Job for experiment 2

In this experiment, we have only lower the RAM size of Data node 3 from8 GB to 6 GB and number of processor of Datanode2 changed to 1. We have found that the node 3 also perform the task in same time like other 2 data node. So, we can say only simple change does not degrade the performance of the Map Reduce Job and no tail latency is found in this experiment.

Experiment 3

To find tail latency, we have performed Map Reduce job by changing the configuration of the three Data node machine.

Table 4: Hardware configuration of Nodes for experiment 3

Node Name	*RAM*	*No of Processor*	*Storage in used*	*Parallel Processing*
Datanode1	8 GB	2	350 GB	1
Datanode2	4 GB	2	350 GB	1
Datanode3	4 GB	2	350 GB	1

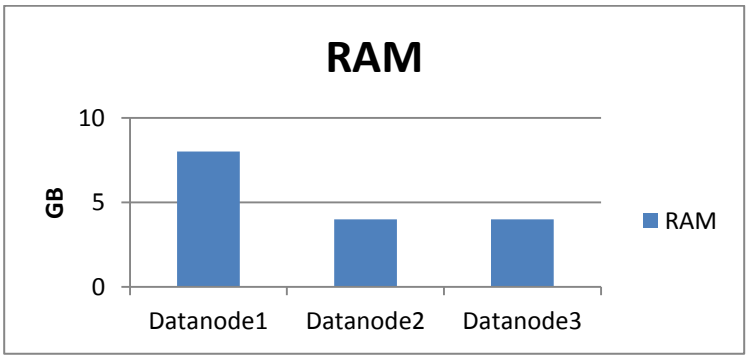

Figure 5: RAM configuration of Nodes for experiment 3

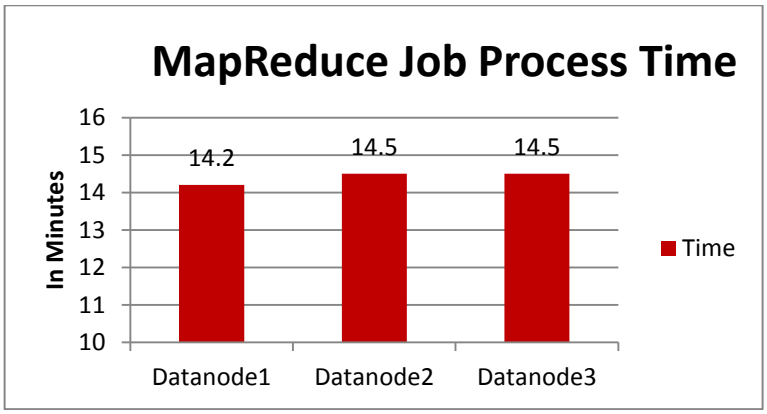

Figure 7: Time Taken by Different nodes for a Map Reduce Job for experiment 3

Experiment 4

We have made one of machine's RAM but we could not fond and difference in completing the Map Reduce job.

‘

Table 5: Hardware configuration of Nodes for experiment 4

Node Name	*RAM*	*No of Processor*	*Storage in used*	*Parallel Processing*
Datanode1	8 GB	2	350 GB	1
Datanode2	2 GB	1	350 GB	1
Datanode3	2 GB	1	350 GB	1

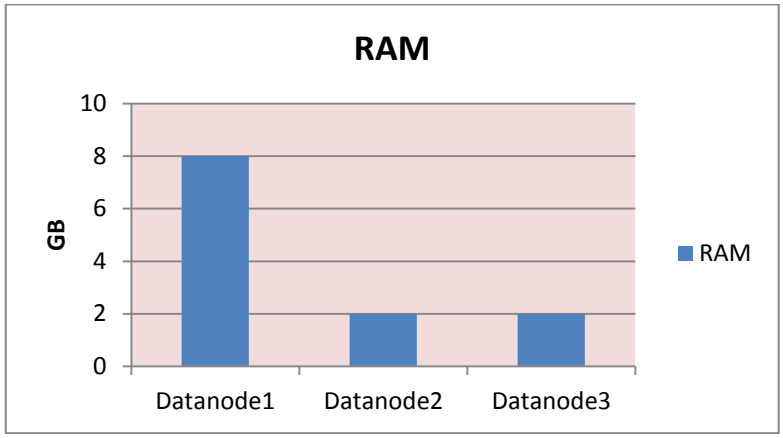

Figure 6: RAM configuration of Nodes for experiment 4

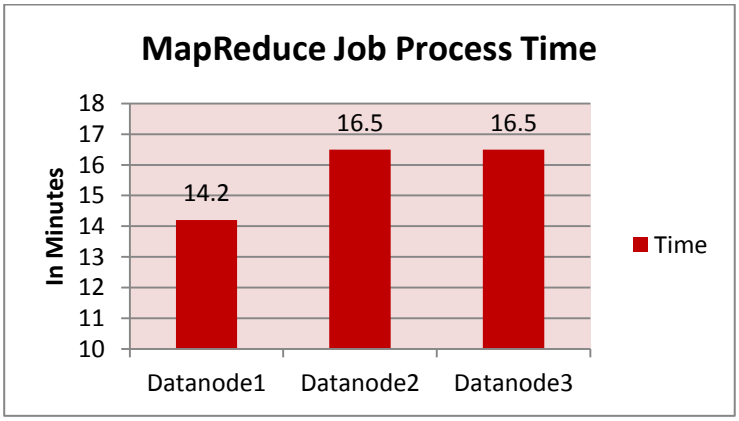

Figure 7: Time Taken by Different nodes for a Map Reduce Job for experiment 4

In this experiment, we have lowered the RAM size of Datanode2 and Data node 3 from 8 GB to2 GB and also number of processor from 2 to 1. After running the Map Reduce job, we found that these two nodes finish the job 2 minutes 30 seconds later than datanode1. So, we can say only these changes degrade the performance of the Map Reduce Job by 2 minutes 30 seconds which is a very significant change in a 4 node cluster system. From this experiment, we can say if there is a 1000 node scenario this issue can cause and significant tail latency issue because then the performance of the job will be very slow.

Experiment 5

Table 6: Hardware configuration of Nodes for experiment 5

Node Name	*RAM*	*No of Processor*	*Storage in used*	*Parallel Processing*
Datanode1	8 GB	2	400 GB	1
Datanode2	8 GB	2	350 GB	1
Datanode3	8 GB	2	250 GB	1

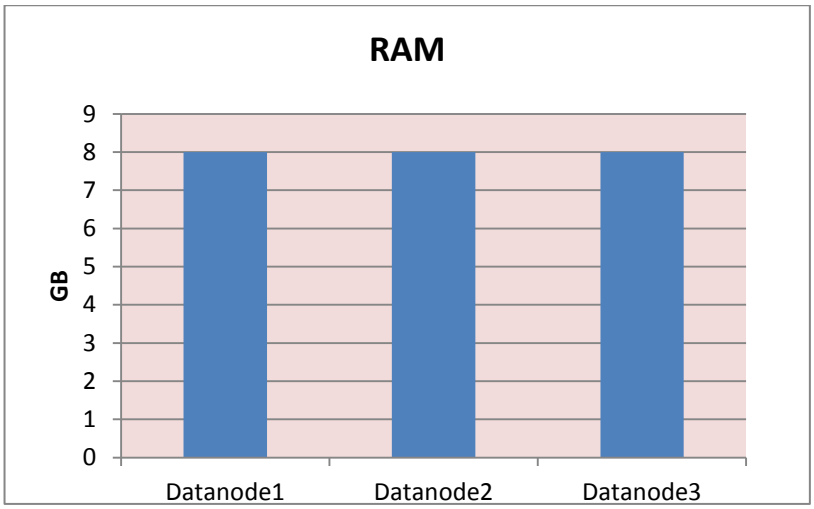

Figure 8: Hardware configuration of Nodes for experiment 5

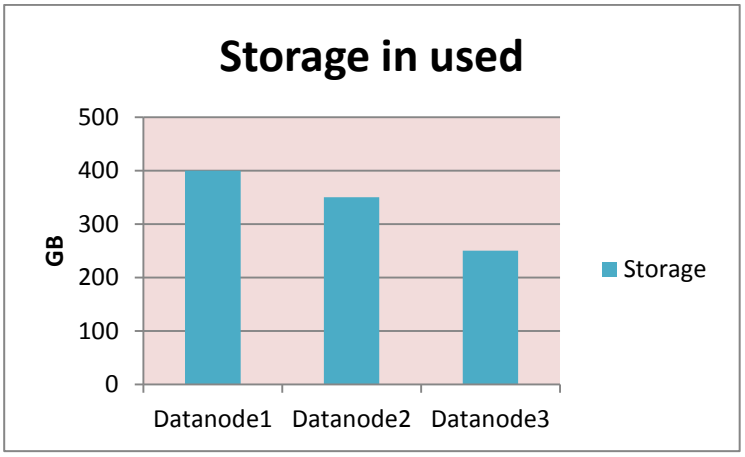

Figure 9: Storage used in Different node for experiment 5

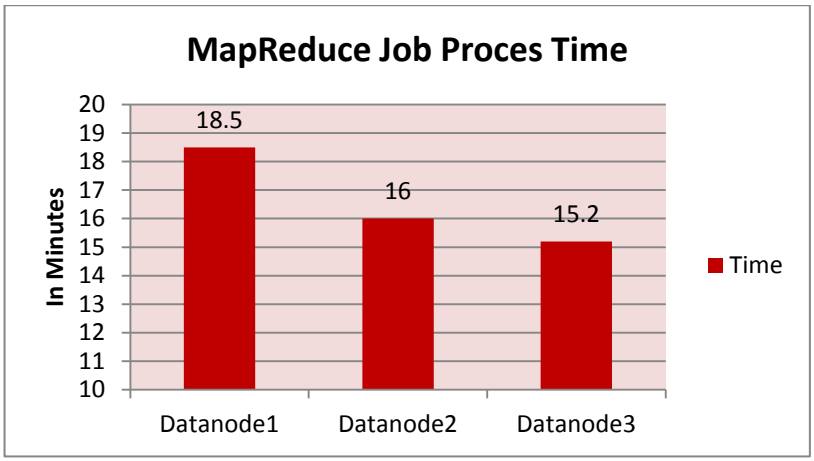

Figure 10: Time Taken by Different nodes for a Map Reduce Job for experiment 5

In this experiment, we have changed the storage used of the two data node and kept same configuration of the other parameters of all the 3 Data nodes and found that all node perform the task in due time, the node which got less record run faster. So, no latency is found.

Experiment 6

We have made one of machine's RAM but we could not fond and difference in completing the Map Reduce job.

Table 7: Hardware configuration of Nodes for experiment 6

Node Name	*RAM*	*No of Processor*	*Storage in used*	*Parallel Processing*
Datanode1	8 GB	2	350 GB	4
Datanode2	8 GB	2	350 GB	6
Datanode3	8 GB	2	350 GB	6

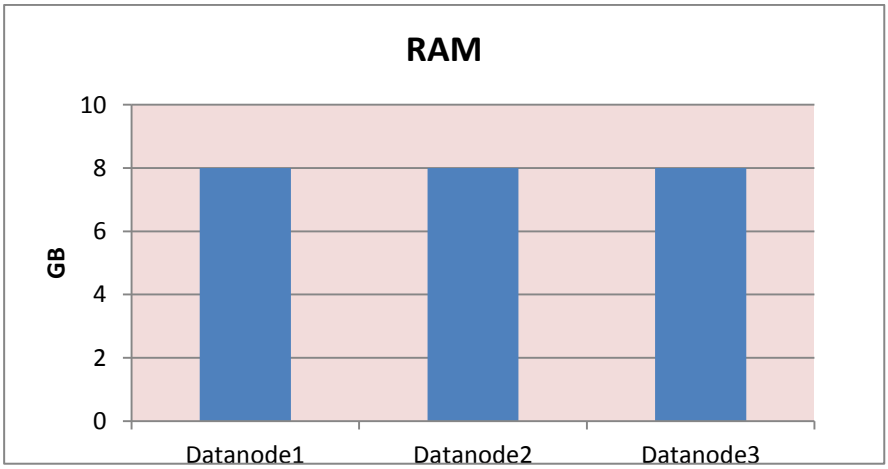

Figure 11: Hardware configuration of Nodes for experiment 6

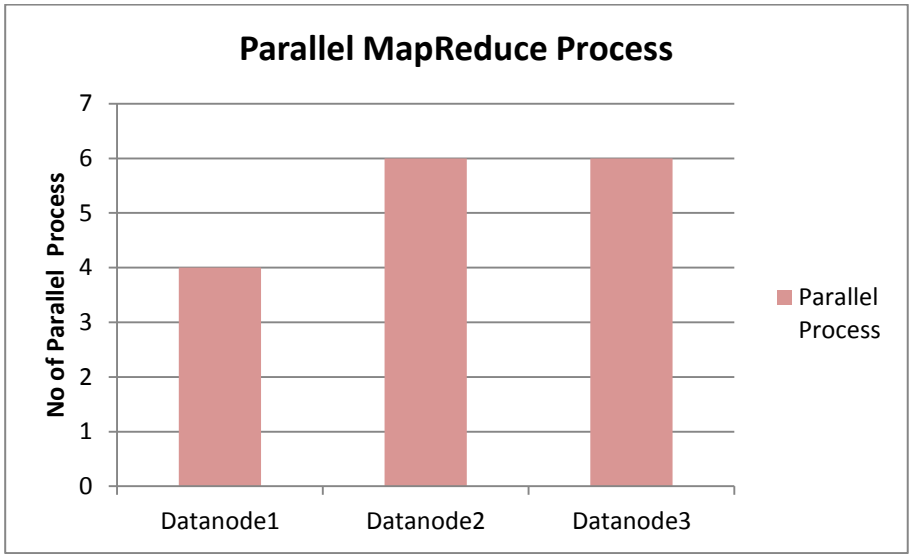

Figure 12: No of Parallel Map Reduce job in Different node for experiment 6

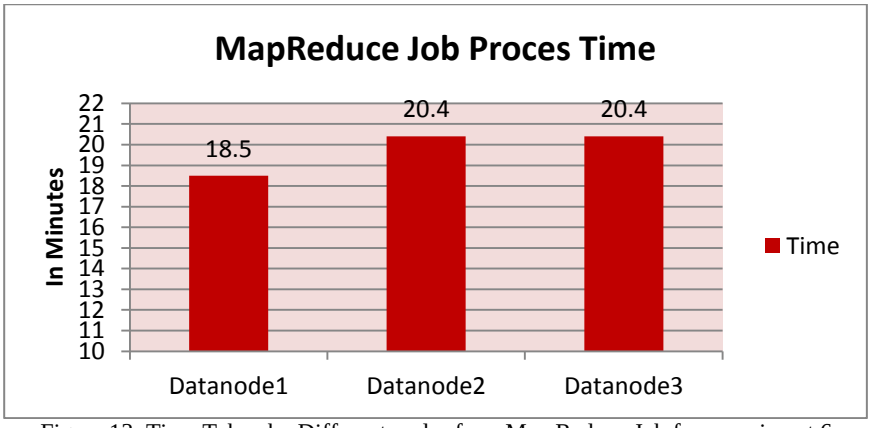

Figure 13: Time Taken by Different nodes for a Map Reduce Job for experiment 6

In this experiment, we have fixed the RAM size of all the Data nodes and process multiple jobs in parallel. Datanode1 has 4 processes running in parallel while datanode2 and datanode3 has 6 processing running. Here, we have found that overall performance of the job is degraded due to multiple parallel processing. We have also found that datanode2 and datanode3 finishes their job 1 minute 50 seconds later than datanode1. So, we can say that parallel processing degrades the performance of the Map Reduce Job and there are tail latency found in this experiment by1 minutes 50 seconds which is significant in a 4 node cluster.

In this experiment, we have lower the RAM size of Datanode2 and Data node 3 from 8 GB to 4 GB and found that these two nodes finish the job 3seconds later than datanode1. So, we can say that only these changes degrade the performance of the Map Reduce Job by 3seconds and also there are tail latency found in this experiment as 3 seconds delay in Data node is still significant in a 4 node cluster.

VI. ANALYSIS OF THE EXPERIMENTS

After completing all the experiment, we have found that if we make minor changes in the parameter of the hardware configuration then the Map Reduce job is not delaying and no tail latency is found. The tail latency is only found only when we have made significant changes in the parameter of the hardware or run multiple jobs in parallel. That means if the Data Node's configuration is too bad or perform poorly the map reduce job is taking longer time despite other node perform the job faster. We have also found the capacity of storage does not add to the performance of the Map Reduce job. Another observation is that if we perform multiple processes in parallel in the data nodes, then we found performance of the Map Reduce changes a lot which also cause tail latency. Here we have observed that one slow service can drastically slow down the entire combined response of the job. Identifying the slow node is a big step forward the solve the issue. If we can identify the slow node, we can take that job away from the slow node to faster node and also can prevent to send job in the slower node to avoid tail latency.

We cannot perform the geographically distributed record processing and also have limited node due to scarcity of resource. Due to these reasons, we couldn't perform the above mention test to find out latency.

VII. CONCLUSION

In this study paper, we have tried to give our focus on finding the tail latency of Apache Hadoop distributed system. We basically used Hadoop Map Reduce technology using Apache Hive to find out the Straggler problem. We have created a 4 node hadoop system and perform several experiments on it using different hardware configuration. After running multiple jobs, we have found tail latency in some our experiment. Especially when a particular node performs very poorly, it degrades the overall performance of a job very significantly. In one case, we have found that overall performance degraded for 2.5 minutes which is15% of the overall time. We have also found that if multiple jobs run in parallel then performance of the jobs degraded a lot and in this case there are a lot of chance that some of the slowly performing node does take longer time than usual which increases the overall processing time and decrease the performance of the job. Tail latency is of great importance in user-facing real-time big data processing. However, maintaining low tail latency is challenging.

This Straggler problem can cause huge impact when processing huge amount of data (pet byte data), so it should be taken very seriously while processing huge data. Many techniques have been evolving recently to mitigate the tail latency issue. Google has developed short-term adaptations of tail latency issue but the issue still persists.

Outcome of the Study

The outcome of the study is if a particular node has very poor configuration i.e. (RAM, CPU) then it performs very poorly and degrade the overall performance of a job which add to tail latency problem. Secondly, if multiple jobs run in parallel, then the node that running more multiple jobs simultaneously then it takes more time to perform a task which increases the overall processing time and decrease the performance of the job.

Drawback of the Study

We have some limitations while performing the experiments which are given below:

- Does not have geographically Distributed record set
- Fewer number of node
- Fewer number of processor in a node
- Latency due to distance Network issue cannot be tested

Future Planning

There is lots of solution on tail latency for various type of problem. We want to analyze those papers and try to find out a better solution which is lot better than existing solutions. So, our future goal is to invent a new solution which will be much more effective to solve the Straggler problem which will be better than the existing solutions and try to solve the tail latency issue.

REFERENCES

[1] S. Venkataraman, A. Panda, M. J. Franklin, and I. Stoica, "The Power of Choice in Data-Aware Cluster Scheduling This paper is included in the Proceedings of the Operating Systems Design and Implementation .," 2014.

[2] D. Ford *et al.*, "Availability in Globally Distributed Storage Systems," *9th USENIX Symp. Oper. Syst. Des. Implement.*, pp. 61–74, 2010.

[3] X. Tian, R. Han, L. Wang, J. Zhan, and G. Lu, "Latency critical big data computing in finance," *J. Financ. Data Sci.*, vol. 1, no. 1, pp. 33–41, 2015.

[4] J. Dean and S. Ghemawat, "Summary of Installed Capacity , Dependable Capacity , Power Generation and Consumption (2003-2016)," pp. 137–149, 2016.

[5] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, p. 74, 2013.

[6] W. D. Gray and D. A. Boehm-Davis, "Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior," *J. Exp. Psychol. Appl.*, vol. 6, no. 4, pp. 322–335, 2000.

[7] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim, "Measuring interference between live datacenter applications," *Int. Conf. High Perform. Comput. Networking, Storage Anal. SC*, no. 3, 2012.

[8] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones," *Nsdi*, p. 185, 2013.

[9] G. Ananthanarayanan *et al.*, "Reining in the Outliers in Map-Reduce Clusters using Mantri," *Time*, pp. 265–278, 2010.

[10] E. Krevat, J. Tucek, and G. R. Ganger, "Disks are like snowflakes: no two are alike," *Proc. 13th USENIX Conf. Hot Top. Oper. Syst.*, p. 5, 2011.

[11] A. Tumanov, R. H. Katz, M. A. Kozuch, C. Reiss, and G. R. Ganger, "Heterogeneity and dynamicity of clouds at scale," pp. 1–13, 2012.

[12] P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan, "The influence of operating systems on the performance of collective operations at extreme scale," *Proc. - IEEE Int. Conf. Clust. Comput. ICCC*, 2006.

[13] F. Petrini, D. J. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance," vol. 836, p. 55, 2011.

[14] K. B. Ferreira, P. G. Bridges, R. Brightwell, and K. T. Pedretti, "The impact of system design parameters on application noise sensitivity," *Cluster Comput.*, vol. 16, no. 1, pp. 117–129, 2013.

[15] C. Curino, D. E. Difallah, C. Douglas, and S. Krishnan, "Socc14-Paper15."

[16] A. D. Ferguson, P. Bodik, E. Boutin, and R. Fonseca, "Jockey : Guaranteed Job Latency in Data Parallel Clusters," *Proc. 8th ACM Eur. Conf. Comput. Syst. - EuroSys '12*, pp. 99–112, 2012.

[17] B. Hindman *et al.*, "2011_Benjamin Hindman_Benjamin Hindman_Mesos A Platform for Fine-Grained Resource Sharing in the Data Center."

[18] C. R. Lumb and R. Golding, "D-SPTF: Decentralized Request Distribution in Brick-based Storage Systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 38, p. 37, 2004.

[19] D. Shue, M. Freedman, and A. Shaikh, "Performance Isolation and Fairness for Multi-Tenant Cloud Storage Setting : Shared Storage in the Cloud."

[20] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "PriorityMeister: Tail Latency QoS for Shared Networked Storage," *Symp. Cloud Comput.*, pp. 1–14, 2014.

[21] M. Capitão, "Mediator Framework for Inserting Data into Hadoop Micael José Pedrosa Capitão Plataforma de Mediação para a Inserção de Dados em Hadoop Mediator Framework for Inserting Data into Hadoop," no. January, 2015.

[22] "Apache Hadoop." [Online]. Available: http://hadoop.apache.org/. [Accessed: 08-Mar-2019].

[23] "NameNode and DataNode – Hadoop In Real World." [Online]. Available: http://www.hadoopinrealworld.com/namenode-and-datanode/. [Accessed: 08-Mar-2019].

[24] "What is Hadoop Distributed File System (HDFS)? - Definition from WhatIs.com." [Online]. Available: https://searchdatamanagement.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS. [Accessed: 10-Jan-2019].

[25] "20 Essential Hadoop Tools for Crunching Big Data – Data Science IO – Medium." [Online]. Available: https://medium.com/data-science-io/20-essential-hadoop-tools-for-crunching-big-data-efbc8b5c77ce. [Accessed: 15-Jan-2019].

[26] A. Manzanares *et al.*, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," *Ned. Tijdschr. Psychol.*, vol. 4, no. 4, pp. 1–9, 2010.

[27] "Hadoop Soup: 01/10/14." [Online]. Available: http://dailyhadoopsoup.blogspot.com/2014_01_10_archive.html. [Accessed: 10-Sep-2018].

[28] "20 essential Hadoop tools for crunching Big Data." [Online]. Available: https://bigdata-madesimple.com/20-essential-hadoop-tools-for-crunching-big-data/. [Accessed: 08-Sep-2018].

[29] "Apache Spark Introduction." [Online]. Available: https://www.tutorialspoint.com/apache_spark/apache_spark_introduction.htm. [Accessed: 08-Aug-2018].

[30] "Home - Apache Hive - Apache Software Foundation." [Online]. Available: https://cwiki.apache.org/confluence/display/HIVE. [Accessed: 08-Aug-2018].

[31] "What is Hive? Architecture & Modes." [Online]. Available: https://www.guru99.com/introduction-hive.html. [Accessed: 08-Jul-2018].

[32] "Impala Hadoop Tutorial." [Online]. Available: https://www.dezyre.com/hadoop-tutorial/hadoop-impala-tutorial. [Accessed: 20-Nov-2018].

[33] "Cloudera Impala Overview | 5.3.x | Cloudera Documentation." [Online]. Available: https://www.cloudera.com/documentation/enterprise/5-3-x/topics/impala_intro.html. [Accessed: 04-Oct-2018].

[34] "Big Data: How to manage Hadoop." [Online]. Available: https://www.cleverism.com/how-to-manage-hadoop-big-data/. [Accessed: 20-Dec-2018].

[35] "Introduction to batch processing - MapReduce - Data, what now?" [Online]. Available: https://datawhatnow.com/batch-processing-mapreduce/. [Accessed: 05-Jan-2019].

[36] A. Gupta and G. N. Campus, "HIVE- Processing Structured Data in HADOOP," no. August, 2018.

[37] "Why is Impala faster than Hive? - Quora." [Online]. Available: https://www.quora.com/Why-is-Impala-faster-than-Hive. [Accessed: 30-Sep-2018].

[38] "What is the advantages of Hadoop and Big data? - Quora." [Online]. Available: https://www.quora.com/What-is-the-advantages-of-Hadoop-and-Big-data. [Accessed: 12-Jan-2019].

[39] "Advantages of Hadoop MapReduce Programming." [Online]. Available: https://www.tutorialspoint.com/articles/advantages-of-hadoop-mapreduce-programming. [Accessed: 10-Dec-2018].

