International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 04 Issue: 06 | June -2017 www.irjet.net p-ISSN: 2395-0072

Enhancing Performance and Fault Tolerance of Hadoop cluster

Chetan Singh[1], Rajvir Singh[2]

[1]M. Tech scholar, Computer Science Department, D.C.R.U.S.T., Murthal
[2]Assistant Professor, Computer Science Department, D.C.R.U.S.T., Murthal

---

Abstract - *New approach to make a system more fault tolerant is to expect failures rather than trying to avoid it. Here, by fault tolerance, we do not mean that there will be no or less failure in the system, instead, it means how the system deals with the failure when it occurs. In hadoop clusters, faults are handled by applying various measures like many copies of data blocks are maintained over several HDFS nodes, re-execution of map and reduce tasks is scheduled if it fails during execution. This re-processing of jobs can, however, decrease the efficiency of job execution. To this end, we are proposing a method to identify faulty nodes of the cluster and remove them to increase the job execution efficiency of the cluster. Our experiment shows that overall efficiency of cluster is improved by our method.*

Key Words: **Hadoop, HDFS, MapReduce, Fault tolerance, Blacklist node.**

1.INTRODUCTION

With the rise of continuous advancement in technologies such as big data and cloud computing, the architecture of high performance computing and distributed systems have become even more complicated. Fault-tolerant computing involves intricate algorithms which make it extremely hard. It is simply not possible to construct certainly foolproof, 100% reliable fault tolerant machines or software. Thus the task to which we should focus on is to reduce the occurrence of failure to an "acceptable" level.

Distributed systems have capability of large scale processing and MapReduce[1] provides a simple way to achieve it. Hadoop[2] has already been successfully applied as an open source implementation of MapReduce. Hadoop has two major components: MapReduce (execution engine) and HDFS (hadoop distributed file system). Both of these components provide fault tolerance[3] to some extent.

First, HDFS[4] provides fault tolerance through replication by splitting files into equal sized data blocks and replicating it over several HDFS nodes, so that, if any one node shows sign of failure, data can still be recovered from other replicated nodes. Second, MapReduce handles the task failures by re-assigning them to other nodes and also handles the node failures by re-scheduling all tasks to other nodes for re-execution. In other words, we can say, HDFS provide fault tolerance to the storage part of the distributed system and MapReduce provide job level fault tolerance.

One of the reasons of the degradation in efficiency of a hadoop cluster is the repetitive failure of some faulty nodes, which prevent smooth execution of jobs because tasks have to be re-scheduled on every failure which acts as an overhead for the overall cluster.

For this purpose, in this paper we have proposed a mechanism to detect these faulty nodes of the cluster and reset the cluster by removing such nodes to increase the overall performance of the cluster. We proposed a blacklist based faulty node detection method in which performance of a node is monitored and according to the number of task failures, a node is categorized as an active node or a blacklisted node. By monitoring the status of a node i.e. how often a node has been blacklisted, we can consider a poorly performing node to be a faulty node. In the end, our empirical experiment shows the increase in performance due to our proposed method.

The remaining paper contain the following sections: Section 2 contains some background of hadoop. Previous work done is reviewed in Section 3. Our proposed method is explained in Section 4. Experiments conducted are discussed in section 5 and then result and conclusions are discussed in the last section.

2. GROUNDWORK

This section contains some background information about hadoop. Hadoop is an open source project hosted by Apache Software Foundation.[5]

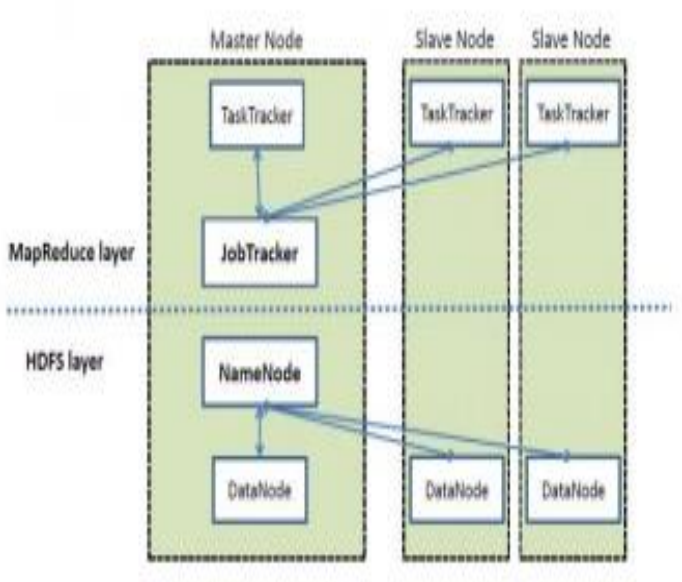

Fig -1: Hadoop Architecture

Hadoop has two major components:

- A File System (HDFS)
- Programming Paradigm or execution engine (Map Reduce)

1) Hadoop Distributed File System: HDFS deals with two types of nodes – Namenode and Datanode. HDFS follows master – slave architecture where namenode acts as a master and datanode acts as a slave. Namenode contains the information of all the datanodes involved in the cluster and helps in coordination, managing file system namespace and overall node management. Datanode store and retrieves block as commanded by the client or the namenode. HDFS data blocks are much larger in size (64 MB by default) than that of the normal file system[6]. The size of data blocks is kept this large in order to reduce the number of disk seeks.

Multiple copies of data blocks are replicated over several nodes so that data can be recovered if some block goes missing due to some task failure or node failure[7]. In order to accomplish this smoothly, replicated copies must be consistent

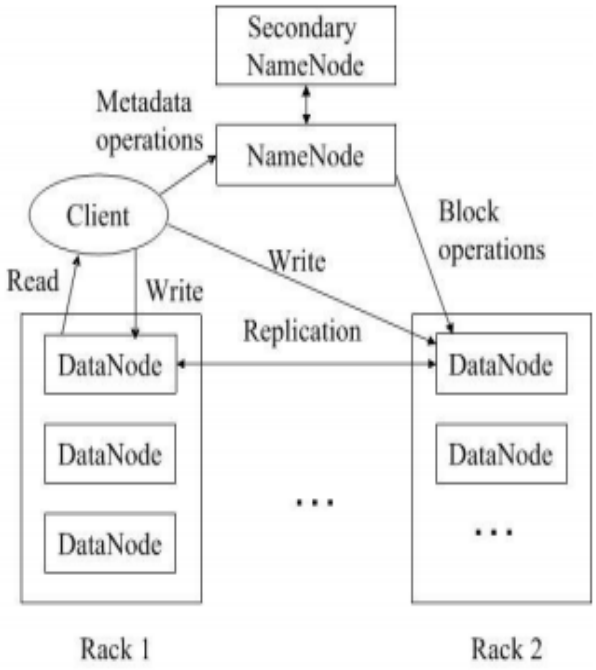

Fig -2: HDFS Architecture

with the original data block. Any write operation on the data block must be reflected in all its replicas to maintain the overall consistency of the cluster data.

2) MapReduce: MapReduce acts as the programming model for hadoop. Fig 3 shows the work flow of MapReduce which is explained as follow:

First, input is broken down into smaller divisions of favourable size. These partitions are then supplied to various map tasks which perform processing on them according to the design of the map functions. Map tasks produce the intermediate result as sequence of key-value pairs which is defined by the code written for map function. These intermediate results are then passed to some reduce nodes by some partition functions. The process of sorting and shuffling takes place between the mapping and reducing phase. Sorting takes place to assure that same key value ends with the same reduce tasks. The code written for reduce tasks will then defines that how the combination process will take place. Then, by working one key at a time, reduce tasks will combine all values associated with it.

The master node of hadoop runs jobTtracker which handles the management and scheduling of several tasks. The slave nodes run taskTracker where actual mapping and reducing takes place.

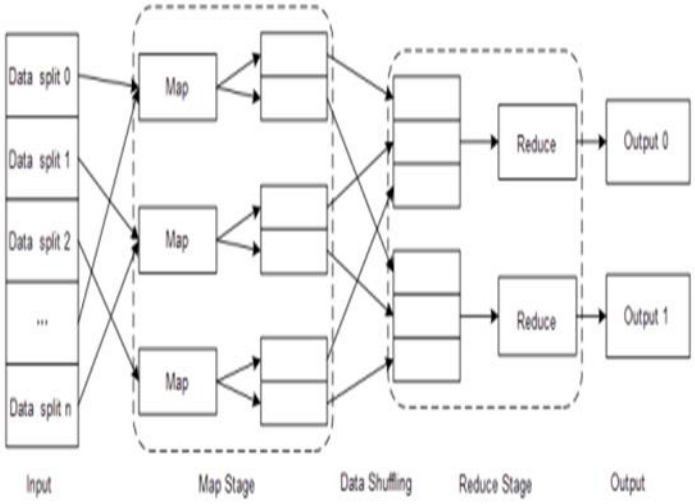

Fig -3: Working process of MapReduce

Master node detects the failure of a node by periodically pinging all its slave nodes. If a node does not reply for specific interval of time, then the master node consider it as failure of the node. Now, all the map tasks which were assigned to that node, have to be re-executed even if it had completed because the results of that computation would be available on that node only for the reduce tasks. These map tasks are then marked as idle by master and they get re-scheduled on a worker when a worker becomes available. The master must also update the information to each reduce task regarding the change of the location of its input from that map task.

3. PREVIOUS WORK

There also has been some other work done in the field of fault tolerance in hadoop's MapReduce paradigm. Peng Hu et al.[8], proposed an alternative method for failure detection of nodes rather than completely depending upon the timeout mechanism of native hadoop. The authors proposed a trust based failure detection algorithm to detect failures earlier as compared to native hadoop. After detection of failure, a

checkpoint based recovery algorithm has also been proposed by the authors.

Matei Zaharia et al.[9], proposed a method to improve the total execution time of the cluster. Authors proposed a scheduling mechanism based on the longest approximate time taken to end a job and uses longest remaining time as a measure for scheduling various tasks.

Borthakur et al.[10], proposed a method to handle Single Point Of Failure (SPOF) i.e. failure at master node (namenode), which contains all the metadata of all datanodes. Author introduced a concept of avatar node which takes place of a master node in case of master node failure.

Quan chen et al.[11], proposed a self – adaptive MapReduce scheduling algorithm which helps in scheduling map and reduce tasks by adjusting time weight of each stage of map and reduce according to the historical information collected earlier which was stored on every node and updated after every execution. This scheduling reduces the overall execution time of the job and hence increase the performance of the cluster.

In our paper, we have proposed a mechanism to improve the total execution time of a job by identifying and removing those particular nodes (faulty nodes) which are bringing the overall cluster down with them by lagging behind in completion of the job.

4. PROPOSED WORK

In hadoop, execution engine i.e. MapReduce perform in three stages. First, Map tasks are performed during first stage and their intermediate results are saved to the local storage. Note that we have to re-execute all the map tasks in case of failure because their results are stored on the local disk(s) of the failed machine and hence, are inaccessible after failure of the machine[12]. Second, sorting and shuffling of intermediate result takes place. Sorting takes place to assure that same key value ends with the same reduce tasks. Local results are transferred to reduce tasks during the shuffling stage. Third, the results are saved to the global file system (HDFS) after the completion of reduce tasks[13].

In this section, we proposed a blacklist based faulty node detection method to detect the nodes which is considerably degrading the overall performance of the cluster. These nodes are then removed from the cluster so that future jobs are not assigned to them and master does not need to apply extra overhead in continuously sending heartbeat messages to those nodes to check their "liveness".

ALGORITHM – Blacklist Based Faulty Node Detection:

1. Setup a hadoop ultimode cluster by adding necessary metadata information to the 'masters' and 'slaves' files of each node.
2. Set a threshold value 'θ_1' for each node associated with cluster and let 'N_f' be the number of failure of tasks for a job.
3. Assign a job to the cluster and check:
   i. if $N_f < \theta_1$, continue job execution until completion of the job.
   ii. Else, add the node to the list of blacklisted nodes and stop further scheduling of tasks to that node.
4. After the completion of the current job, remove the nodes from the list of blacklisted nodes.
5. Maintain a record of the number of times a node has been blacklisted, let it be N_b.
6. Set a threshold value 'θ_2' for every node such that, if N_b reaches θ_2, than that node is considered as the faulty node.
7. Remove this node from the cluster to prevent any scheduling of future jobs on it, as it has been identified as the faulty node.

Once the faulty node has been detected using above mentioned algorithm, then that node will be removed from the cluster. The threshold values θ_1 and θ_2 must be chosen carefully depending upon the size of the cluster and the type and size of the job to be assigned to the cluster.

5. EXPERIMENT AND RESULT

To perform our experiment, we first need to setup a multimode cluster. We have installed four Ubuntu machines on a single PC using Vmware. Each machine is assigned 2GB RAM and 100GB hard disk. We let one of these nodes to be a master node and others will act as slave nodes.

Master node will run namenode, secondary namenode and nodemanager on its machine and datanode and resourcemanager will run on each of the slave node. Note that, nodemanager is the jobtracker and resourcemanager is the tasktracker and they have to be run on master and slave nodes respectively.

We have chosen the job of calculating the value of 'pi' using hadoop mapreduce. Using the command: pi 64 100000000, we set 64 map tasks for the job and 100000000 samples will be generated per map task. In our experiment, we will determine execution time of this job before and after applying our mechanism

As shown in fig 4, total execution time of the same job is reduced after the removal of faulty node. This happened because native hadoop takes time to consider a node as a

failed node even if it is suffering from too many faults[14]. In our experiment, we have a faulty node which stops working for a while but it starts working again before it can be considered as failure. This leads to re-assigning of tasks to that node which is failing occasionally and which further leads to failing and re-execution of tasks. As a result, total execution time for the job is increased.

However, if the node had failed completely, it would have taken much more time to complete the job because native hadoop would have taken large amount of time to consider it as a failure and then only its tasks would have been assigned to another node. This delay would have added extra time to the execution time of the job.

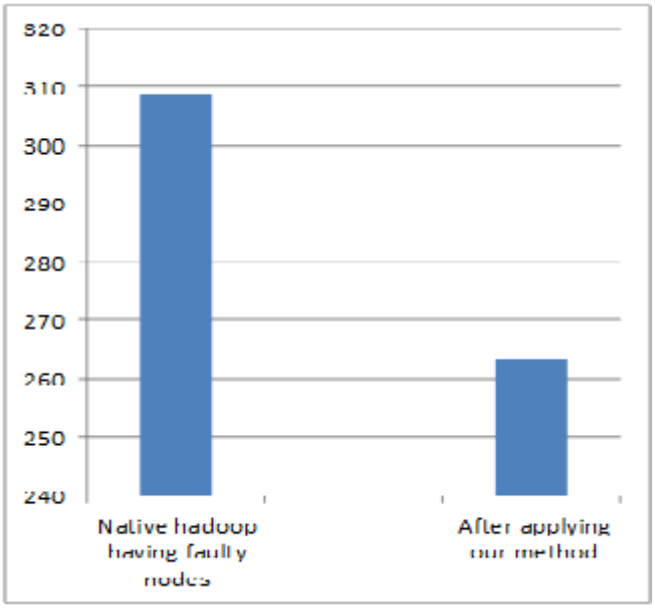

Chart -1: Comparison of execution time

Note that, not every fault reaches the stage of failure but it still degrade the performance to some extent. Here, these faults are detected and handled before they become any major failure and have any serious impact on the job completion efficiency of the cluster. And hence, the execution time for the job is reduced.

5. CONCLUSION

In this paper, we proposed a mechanism to identify those faulty nodes which are majorly responsible for the degradation of the overall efficiency of the cluster. Some nodes are referred as stragglers which increase the total execution time of the job by lagging behind during the final phase of job completion. If these nodes fall under the specifications of our proposed mechanisms, then they will also be detected as faulty nodes and will be removed from the cluster to increase the overall performance.

Some faulty nodes show errors for repeated but short intervals. These intervals are shorter than the timeout interval of detecting failures. These faults needed to be detected and handled because it is not practical to wait for them to become any major failure which we seriously need to be concerned with at later stage.

REFERENCES

[1] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[2] T. White, "Hadoop: the definitive guide", O'Reilly, (2012).

[3] Sivaraman, E., & Manickachezian, R. (2014, March). High performance and fault tolerant distributed file system for big data storage and processing using hadoop. In Intelligent Computing Applications (ICICA), 2014 International Conference on (pp. 32-36). IEEE.

[4] Shvachko, K., et al. 2010. The Hadoop Distributed File System. IEEE. http://storageconference.org/2010/Papers/MSST/Shvachko.pdf.

[5] http://hadoop.apache.org

[6] Li, B., & Jain, R. (2013). Survey of Recent Research Progress and Issues in Big Data. Washington University in St. Louis, USA.

[7] Kwon, O., Lee, N., & Shin, B. (2014). Data quality management, data usage experience and acquisition intention of big data analytics. International Journal of Information Management, 34(3), 387-394.

[8] Hu, P., & Dai, W. (2014). Enhancing fault tolerance based on Hadoop cluster. International Journal of Database Theory and Application, 7(1), 37-48.

[9] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., & Stoica, I. (2008, December). Improving MapReduce performance in heterogeneous environments. In Osdi (Vol. 8, No. 4, p. 7).

[10] Borthakur, D., Gray, J., Sarma, J. S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., ... & Schmidt, R. (2011, June). Apache Hadoop goes realtime at Facebook. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 1071-1080). ACM.

[11] Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010, June). Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on (pp. 2736-2743). IEEE.

[12] Egwutuoha, I. P., Levy, D., Selic, B., & Chen, S. (2013). A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. The Journal of Supercomputing, 65(3), 1302-1326..

[13] Goranson, C., Huang, X., Bevington, W., & Kang, J. (2014). Data Visualization for Big Data.

[14] Katal, A., Wazid, M., & Goudar, R. H. (2013, August). Big data: issues, challenges, tools and good practices. In Contemporary Computing (IC3), 2013 Sixth International Conference on (pp. 404-409). IEEE.