Implementation of Blacklist Strategy for Faulty Node Detection to improve the Efficiency of A Hadoop Cluster using Big Data Analysis

Deepak Kumar[1], Saurabh Charaya[2]
[1]*M.Tech Research Scholar, Department of Computer Science & Engineering, OM Institute of Technology and Management Hisar(Haryana)*
[2]*Assistant Professor, Head of Department of Computer Science & Engineering, OM Institute of Technology and Management Hisar(Haryana)*

Abstract- Analyzing the data and extracting the information has acquired a lot of importance now a days from the vast store house of data. A strategy to improve the performance of such systems to make them more tolerant to failures is thus the need of hour. A lot of research work has been done so far in this field. We are here analyzing and purposing a fault tolerance mechanism to improve the efficiency. Fault tolerance here doesn't mean a system to be completely free of faults but how a system overcome and deal with the failures. Hadoop uses a several measures to minimize faults such as storing replica of a file at several nodes, executing map and reduce tasks repeatedly if failure occurs. However, this process results in decreased efficiency. One solution to this problem is finding the faulty nodes and removing them. This will result in increased efficiency. We are proposing the same technique and experimentally show the efficiency of the system.

Key Words- MapReduce, HDFS, Fault tolerance, Hadoop,

I. INTRODUCTION

With the increase of continuous advancement in technologies like huge knowledge and cloud computing, the design of high performance computing and distributed systems became even additional difficult. Fault-tolerant computing involves tangled algorithms that create it extraordinarily laborious. It's merely impracticable to construct actually foolproof, 100% reliable fault tolerant machines or code. Therefore the task to that we must always specialize in is to cut back the incidence of failure to AN "acceptable" level.

Distributed systems have capability of enormous scale process and MapReduce[1] provides a straightforward thanks to bring home the bacon it. Hadoop[2] has already been with success applied as an open supply implementation of MapReduce. Hadoop is primarily working with 2 major components: MapReduce (execution engine) and HDFS (hadoop distributed file system). Each of these parts give fault tolerance[3] to some extent.

Firstly, HDFS[4] replicates file copies over several nodes by splitting them into equal sized blocks. In this way, if, any node shows any type of failure in rendering the result, it can be recovered from other nodes. Thereafter, the failed tasks are re-assigned and re-scheduled to alternative nodes by MapReduce so that they are re-executed. We can say, in simple words , HDFS give storage level fault tolerance and MapReduce give job level fault tolerance.

One of the explanations of the degradation in potency of a hadoop cluster is that the repetitive failure of some faulty nodes, that stop smooth execution of jobs. These failed tasks must be re-executed which adds overheads to the cluster.

In this paper, we've proposed a mechanism to find these faulty nodes of the cluster and reset the cluster by removing such nodes to extend the general performance of the cluster. we have a tendency to plan a blacklist based most faulty node detection technique within which performance of a node is monitored and in keeping with the amount of task failures, a node is classified as a full of life node or a blacklisted node. By observance the standing of a node i.e. however usually a node has been blacklisted, we will think about a poorly performing art node to be a faulty node. In the end, our empirical experiment shows the rise in performance because of our planned technique.

The remaining paper contain the subsequent sections: Section 2 contains some background of hadoop. Section 3 review previous work done. Our planned technique is explained in Section 4.Section 5 discuss our conducted experiments, then result and conclusions at last.

II. GROUNDWORK

In this part, we will discuss the background of hadoop. It is an Apache software foundation's open source project.[5]

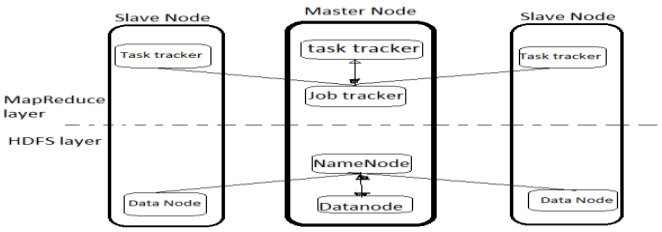

Fig.1: Hadoop Architecture

Hadoop has two major components:

a. A File System (HDFS)

b. Map Reduce

A. Hadoop Distributed File System:

A typical Hadoop consists of two types of nodes - Namenode and Datanode. HDFS follows master – slave design where namenode acts as a master and datanode acts as a slave. Namenode acts as manager of the datanodes and responsible for node management. Datanode accepts commands from the namenode and performs execution and retrieval of task blocks assigned. HDFS knowledge blocks are a lot of larger in size (64 MB by default) than that of the conventional file system[6]. The dimensions of information blocks is unbroken this huge so as to scale back the quantity of disk seeks.

In case of any task failure or node failure, copy of that block can be obtained from any other node as copies are already replicated to all nodes [7]. To make this execution smooth, replicated copies should be consistent with the original data block. Any write operation on the data block should be reflected in its replicas to keep up with the general consistency of the cluster data.

B. MapReduce:

MapReduce acts like the programming model for hadoop. Working of Mapreduce paradigm is shown in fig 3 , as explained below:

First, input is fragmented into smaller divisions of favorable size. These partitions are then equipped to numerous map tasks that perform process on them in keeping with the planning of the map functions. Map tasks turn out the intermediate result as sequence of key-value pairs that is outlined by the code written for map operation. These intermediate results are then passed to some scale back nodes by some partition functions. Sorting takes place to assure that very same key value ends with identical scale back tasks. The code written for scale back tasks can then defines that how combination method can manifest itself. Then, by operating one key at a time, mapreduce tasks can combine all values related to it.

The management and programming of the tasks is accomplished through a job tracker running on master node. Actual mapping and reducing task is done by task tracker which runs on slave nodes. Job tracker, which runs on master node, handles the management and programming of the tasks. The slave nodes run task Tracker where actual mapping and reducing takes place.

Master node uses ping method to detect any failure occurring at any of the slave nodes. If a node doesn't reply for specific interval of your time, then the master node take into account it as failure of the node. Any mapping task which was assigned to this node is now be re-executed. These map tasks then marked as idle by master and get re-scheduled on another working node once the node is available there. The master should additionally update the data to every reduce task relating to the modification of the placement of its input from that map task.

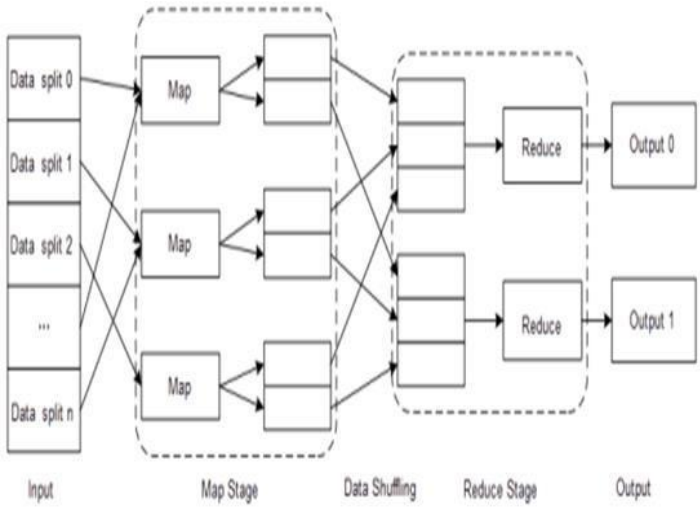

Fig.3: Working process of MapReduce

III. PREVIOUS WORK

Fault tolrence in MapReduce Paradigm of Hadoop was also a point of research previously. Peng Hu et al.[8], who projected on another methodology for failure detection of nodes instead of fully relying upon the timeout mechanism of native hadoop. The authors projected a trust primarily based failure finding formula to detect failures earlier as compared to native hadoop. When a failure been detected, a checkpoint primarily based recovery formula has also been

projected by the authors.

Matei Zaharia et al.[9], projected a technique to boost the overall execution time of the cluster. Authors projected a planning mechanism supported the longest approximate time taken to finish a task and uses longest remaining time as a means for planning varied tasks.

Borthakur et al.[10], projected a technique to handle Single point Of Failure (SPOF) i.e. failure at master node (namenode), that contains all the data of all datanodes. Author introduced an inspiration of avatar node that takes place of a master node just in case of master node failure.

Quan bird genus et al.[11], projected a self – adaptive MapReduce scheduling formula by adjusting time weight of every stage of map and reduce in line with the historical data collected earlier that was on each node and updated after every execution. This planning reduces the general execution time of the work and therefore increases the performance of the cluster.
In our paper, we've proposed a mechanism to boost the overall execution time of task by identifying and removing those specific nodes (faulty nodes) that are consistently decreasing the performance of cluster by not completing the task at time.

IV. PROPOSED WORK

In hadoop, execution engine i.e. MapReduce perform in three phases. Firstly, Map tasks are performed and their intermediate results saved to the native storage. Note that we've to re-execute all the map tasks just in case of failure as their results are on the local disk(s) of the failing machine and thence, are inaccessible during failure of the machine[12]. Second, sorting and shuffling of intermediate result takes place. Sorting takes place to assure that same key worth ends with an equivalent reduce tasks. Local results are transferred reduce tasks throughout the shuffling stage. Third, the results are saved to filing system (HDFS) when the completion reduce tasks[13].

In this section, we proposed a blacklist strategy for faulty node that is significantly degrading the performance of the cluster. These nodes are then aloof from the cluster so future jobs don't seem to be assigned to them and master doesn't got to apply further overhead in spontaneously sending heartbeat messages to those nodes to visualize their "liveness".

ALGORITHM –Faulty Node Detection based on Blacklisting:

1. To begin, we have to setup a Hadoop cluster having three nodes by adding metadata information to the 'masters' and 'slaves' for each node.
2. A threshold value 'θ_f ' must be set for each node in cluster which defines the maximum limit of failures for any node.
3. Let 'N_f' be the failure count variable for a node.
4. Now assign a job to the cluster and check if 'N_f'< 'θ_f '. If true, keep executing the job till completion. Else, add that particular node to the blacklisted nodes and make sure no job is assigned to it further.
5. Remove this node from blacklisted nodes after the job completion.
6. To keep a record for how many times a node has been blacklisted, lets define a variable N_b
7. Let 'θ_b' be threshold variable that states the maximum limit of being blacklisted for a particular node such that if $N_b = \theta_b$, the node is considered as faulty node.
8. Remove this node from the cluster and don't assign any job to it further.

Once the faulty node has been detected using above mentioned algorithm, then that node will be removed from the cluster. The threshold values θ_f and θ_b 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.

V. EXPERIMENT AND RESULT

For the purposed experiment, we have setup a multinode cluster. We've put in four Ubuntu machines on one computer in Vmware. Every machine is assigned 1GB RAM and 20GB disk space. We have a tendency to let one of these nodes to be a master node and other act as slave nodes.

Master node can run namenode, secondary namenode and nodemanager on its machine and datanode and resource manager can run on every of the slave node. Note that, node manager will track jobs and resource manager will track tasks accomplished and they would run on master and slave nodes in the same order.

For the value of 'pi' in the cluster, we have used a command: pi 64 and 100000000, we have a tendency to set 64 map tasks for the job and 100000000 samples are generated per map task. We would check the execution time of this job before and after applying our mechanism for this experiment.

Total execution time of job is reduced once the removal of faulty node. This happened because native hadoop takes time consider a node as a failed node even it is going through too many faults[14]. In our experiment, we've a faulty node that stops operating for a while however it starts operating once more before it is thought of as failure. This ends up in re-assigning of tasks to this node that is failing occasionally and that further ends up in failing and re-execution of tasks. As a result, total execution time for the job is increased.

However, if the node had failing fully, it'd have taken far more time to complete the job because native hadoop would have taken much more time to consider it as a failure and then only its tasks would be scheduled to a different node. This delay would have added additional time to the execution time of the job.

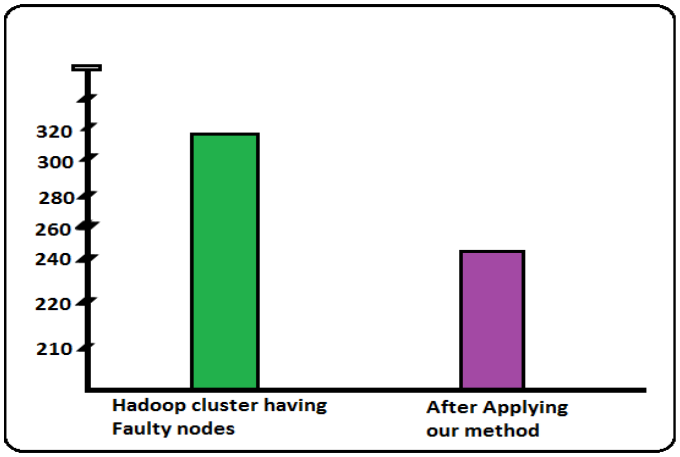

Chart -1: Comparison of execution time

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.

VI. CONCLUSION

The research work in this paper, is concerned with specifying 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.

VII. 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/Shva chko.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 BigData.

[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.

Deepak kumar is a M.Tech student in department of Computer Science & Engineering from OM Institute of Technology and Management, Hisar(Haryana)