Improving Hadoop MapReduce job execution by Load Balancing using Machine Learning Techniques

Akash S H[1], Annappa Swamy D R[2]

[1]PG Student, Dept. of Computer Science and Engineering, MITE, Mangalore, Karnataka, India
[2]Associate Professor, Dept. of Computer Science and Engineering, MITE, Mangalore, Karnataka, India

---***---

Abstract -*For academics and scientists, Hadoop has become a crucial tool for storing and analyzing large amounts of data. The Hadoop Distributed File System is used to store this massive amount of data (HDFS). In HDFS, a block placement policy is used to break a very large file into blocks and distribute them throughout the cluster in an extremely dispersed way. Hadoop and HDFS, in general, are meant to function quickly on a consistent cluster. However during this era of networking, we cannot think of having merely a cluster of consistent nodes. As a result, owing of the diverse cluster, a storage strategy that will work quickly on each consistent is required. Thus, the necessity of programs which will be running in a time-efficiently way and providing consistent yet since the heterogeneous environment will be sufficient data1 in Hadoop.*

MapReduce is a programming framework for creating Map-Reduce programs that may operate in parallel on a distributed environment. MapReduce makes it possible for Hadoop applications to execute. Hadoop places data blocks on nodes using the HDFS block placement strategy. Every now and then, the Hadoop cluster becomes unbalanced due to the overutilization of a few nodes versus the less utilized nodes, or newly generated new nodes with no blocks held on them. Hadoop has a built-in utility called HDFS Balancer that can help you overcome this problem

***Key Words*: Hadoop, Load Balancing, Data Blocks, HDFS Storage.**

1.INTRODUCTION

Hadoop [1] has become popular due to its ability to make use of common machines. The Hadoop Distributed File System (HDFS), MapReduce, and Yet Another Resource Negotiator (YARN) are the three most important components [3]. HDFS allows nodes in the distributed cluster to store data. HDFS is a well-known adherent file system that can be installed on any machine.

The data blocks are stored by Hadoop on the cluster's multiple nodes. The strategy of storing blocks in Hadoop aids in the uniform distribution of data blocks among cluster nodes. This policy allows the data blocks to be moved around in a manner that is compatible with the following approach:

- Splitting the huge number of files into blocks and replicating the blocks in order to address the replication issue described in the hdfs-site.xml file.
- That particular data node. Otherwise, you can put it on any of the cluster's data nodes.
- If available, the next precise copy of the block will be placed on other racks of the nodes, or even placed on a similar rack of the initial replica.
- The third copy must be installed on any rack node where the second copy is already present.

2. HDFS Block Placement Policy

Blocks are mapped to processes inside the same node by Data Locality when using HDFS's block placement policy, however when dealing with large data, it's common to need to map data blocks to processes across several nodes. Hadoop includes a feature that copies the data block wherever mappers are operating to affect this. Because of I/O latency or network congestion, this causes numerous performance degradations, particularly on heterogeneous clusters. We utilize an efficient technique to balance the data block on particular nodes, i.e. custom block placement, by separating the total number of nodes into two classes: Nodes that are homogeneous vs. heterogeneous, or that are high performing vs. low performing. This policy ensures that load is distributed evenly among the nodes and that data blocks are placed exactly where we want them for processing.

3.HDFS Balancer

When the load is not evenly distributed between the nodes, load imbalance might develop inside the cluster. The flexibility of Hadoop is another factor that contributes to cluster imbalance. At any moment, we can add data nodes to an existing cluster. Because freshly added data nodes won't have any data blocks, this might create cluster imbalance. In order to achieve efficient outcomes, it is critical to have a balanced cluster. The HDFS Balancer tool is used to distribute load across a Hadoop cluster.

The intended threshold limit that is established on the cluster is used to move the number of blocks. When the HDFS Balancer tool is set to operate with the default threshold limit of 10%, it tries to maintain a 10% difference in disc utilization from overall cluster use. That means that if cluster utilization is 60 percent of the cluster limit, each node's disc use will be anywhere between 50 and 70 percent [4].

There are two issues with HDFS Balancer: the first is that it moves information blocks without investigating what the threshold limit of nodes where information blocks are being moved is, and the second is that there is no method for controlling the translation behaviour of information blocks of determined documents/datasets as it were. It's also important to remember that transferring data from one node to the next requires a lot of data transmission bandwidth so that cluster balancing can happen quickly.

4.Techniques Used to Solve a Load Balancing Issue in Hadoop

There are two popular approaches to resolving load balancing concerns in Hadoop: The first option is to rebalance the cluster by moving the data, while the second option is to migrate the Task itself. Researchers are focusing on Task Migration rather than Data Migration since data transfer across hubs would increase the temporal complexity.

4.1 Hadoop Schedulers

Can accomplish load balancing to decrease job or task information migration to a degree [8]. The latency time, completion time, and data locality of several Hadoop programming methods are all assessed. Six Big Data applications are enforced in this experiment using three alternative scheduling queue configurations: single queue, multi-queue, and mixed multi-queue. The majority of the tests were carried out by fine-tuning the scheduling rules for the Hadoop environment. The following conclusions are reached:

- In a single queue, Fair-DRF surpasses the other three in terms of execution time and effective resource utilization capabilities. The only flaw is that C.P.U. use time may be a little longer than with Fair-Fair scheduling.

- When considering workload waiting time, completion time, turnaround time, and C.P.U. use in a multi-queue system, Fair-FIFO is the best option. Only if resource utilization is critical is it preferable to be fair.

- In a mixed multi-queue system, When it comes to resource usage and task execution performance, Fair-DRF is the most appropriate option.

4.2Apart from Built-In Scheduler which is provide by Hadoop, Researchers have developed Improved Scheduling Algorithm to contribute it for Load Balancing[6,12]

It is insufficient to distinguish modest tasks. What matters is identifying the task that will have the greatest impact on response time and doing it as quickly as feasible.

Distinguishing an activity as a slow poke when it continues to run for more than two standard deviations outside the mean isn't helpful for reducing reaction time: at this point, the activity might have just ran 3x longer than it should have! As a result, LATE is based on the estimated time remaining and can identify the moderate job at an appropriate moment. A number of factors, such as a high grade on theoretical tasks, ensure responsible behaviour. By inferring heterogeneity from appropriated applications. There are four exercises:

1. Make judgments as soon as feasible rather than relying on base decisions for mean and variance measures.

2. Rather than utilizing progress time, utilize completion time to prioritize activities for speculation.

3. Not all of the nodes are the same. Refrain from assigning theoretical tasks to sluggish nodes.

4. Resources are priceless. Tops should be used to prevent the structure from being overburdened.

This research is also linked to multiprocessor task scheduling with processor heterogeneity [8] and errand duplication when using dependency diagrams [11]. Multiprocessor errand booking is concerned with situations in which processor speeds, while diverse, are known ahead of time and tasks are intimately linked owing to entomb task correlation. This means that, in a multiprocessor environment, it is both possible and necessary to plan task allocations ahead of time, but with MapReduce, the scheduler must adapt gradually to changing conditions on the ground.

Speculative execution in MapReduce has several similarities to "speculative execution" in DFS, configuration management, and data collecting. Strong scheduling algorithm: whereas LATE focuses on approximation that running a job can be overtaken to reduce the response time of a distributed computation, LATE focuses on approximation that running a task can be overtaken to reduce the response time of a distributed computation.

- LATE, which uses projected finish timings to speculatively execute the activities that have the greatest impact on response time.

- On Amazon's Elastic Computing Cloud, LATE outperforms Hadoop's de-fault speculative execution algorithms in actual workloads.

4.3 Tree Based Load Balancing

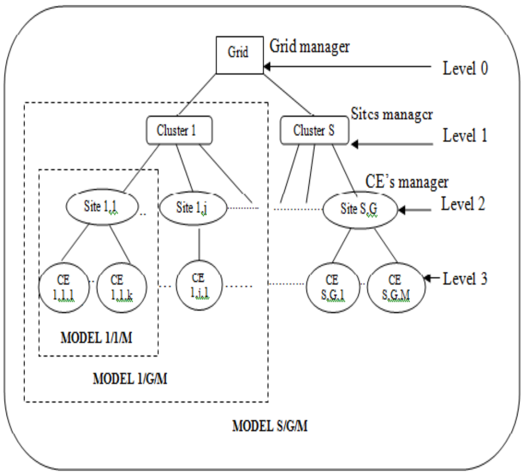


Figure-4.3: Load Balancing Generic Model

To the matrix, the network and offer adaptability to internal failure. The leaf of a tree is regarded as a site's figure component. A lattice is made up of various groups, and bunches are made up of various locations, or we might say that various destinations are gathered to form a group, and various figure components are gathered to form a site. The status of the registration components is saved in a place. The registering components administrator is a program that runs in the background. Each bundle has information on the heap of its locales buried beneath it. The group's programming is referred to as the destinations chief since it manages all of the group's programming, the heap of the destinations that are under the group, and in the event that any site under it fails, it circulates its heap to a less crowded site, preventing the entire group from being disappointed as a result of the site's failure, and thus providing adaptation to non-critical failure to the group. S/G/M refers to the number of bunches that make up a network, G to the number of locations in each bunch, and M to the number of figuring components in each locale. Depending on the upsides of S and G, this model may be transformed into three explicit models: S/G/M, 1/G/M, and 1/1/M.

4.4 Load Balancing Based on Disk Latency and DiskSpace Utilization

When the disc latencies of the data nodes are similar, balancing the blocks uniformly may be the best option. Furthermore, disc latencies will grow with time due to mechanical difficulties and hazardous sectors. Furthermore, crashed and non-functional discs are replaced with newer discs, which may be of a newer generation and have a higher rate. This causes disc non-uniformity inside the cluster, which is otherwise unvarying, therefore balancing evenly in line with disc utilization may not provide the best task execution time.

To address this issue, a disc latency aware balancer was developed, which balances the cluster while considering both space consumption and disc delay. This disc balancing method ensures that a low latency disc receives a greater diversity of blocks than a high latency disc.

The authors evaluated their unique block placement technique on a heterogeneous cluster and found that it improved the job's running time by up to 20%.

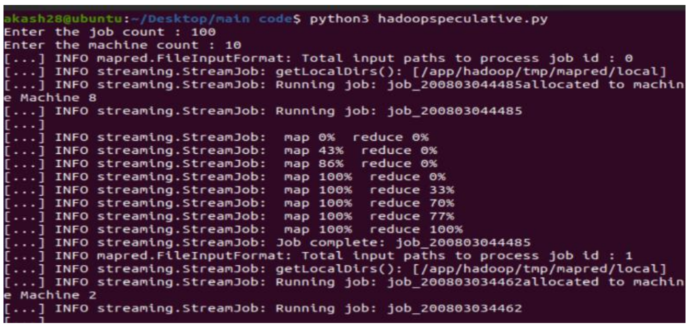
4.5 Hadoop tool for load balancing:

Using a block placement policy that is unique to you Hadoop [12] is a tool for load balancing that takes into account the node's hardware generation. The TSNSGAII multiobject selection method is coupled with a PMKELM prediction model and an adaptive strategy. The PMKELM can help with task execution time prediction, while the TSNSGAII was created to help with picking an appropriate number of reducers. Both models perform well in the experiment, according to the data. During tests, about 47–55 seconds were saved. In terms of storage efficiency, just 1.254 percent of changes in hard disc occupation were created among all planned reducers, which reaches 26.6 percent improvement than the original scheme.

4.6 Load Balancing through Block Rearrangement policy for Hadoop Heterogenous clusters

During block placement, the custom block placement policy limits the CPU's processing capabilities. In MapReduce, this method will assist to reduce inter-node and inter-rack transmission. This demonstrates that only data blocks from a single file are assigned to specified nodes. As a consequence, because the remainder of the files are unaffected, this technique will have no effect on the cluster's overall load balancing. The results of the experiments show that this technique may be applied to both homogeneous and heterogeneous clusters.

5. RESULTS



```
akash28@ubuntu:~/Desktop/main code$ python3 hadoopspeculative.py
Enter the job count : 100
Enter the machine count : 10
[...] INFO mapred.FileInputFormat: Total input paths to process job id : 0
[...] INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
[...] INFO streaming.StreamJob: Running job: job_200803044485allocated to machin
e Machine 8
[...] INFO streaming.StreamJob: Running job: job_200803044485
[...]
[...] INFO streaming.StreamJob:  map 0%  reduce 0%
[...] INFO streaming.StreamJob:  map 43%  reduce 0%
[...] INFO streaming.StreamJob:  map 86%  reduce 0%
[...] INFO streaming.StreamJob:  map 100%  reduce 0%
[...] INFO streaming.StreamJob:  map 100%  reduce 33%
[...] INFO streaming.StreamJob:  map 100%  reduce 70%
[...] INFO streaming.StreamJob:  map 100%  reduce 77%
[...] INFO streaming.StreamJob:  map 100%  reduce 100%
[...] INFO streaming.StreamJob: Job complete: job_200803044485
[...] INFO mapred.FileInputFormat: Total input paths to process job id : 1
[...] INFO streaming.StreamJob: getLocalDirs(): [/app/hadoop/tmp/mapred/local]
[...] INFO streaming.StreamJob: Running job: job_200803034462allocated to machin
e Machine 2
[...] INFO streaming.StreamJob: Running job: job_200803034462
```

Fig-5.1: The above image describes the UI for user to enter the job count and machine count.

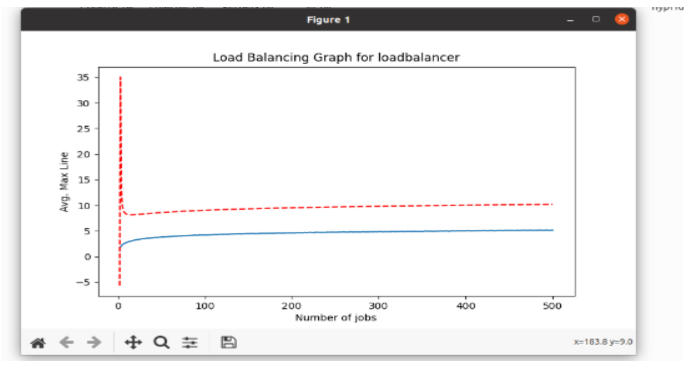
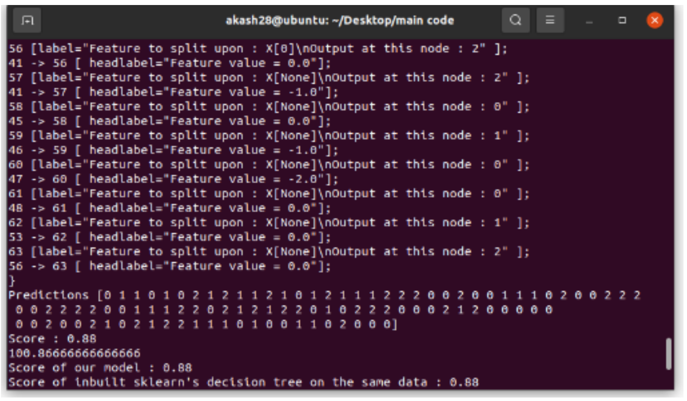


Fig-5.2: This image shows the graph of load balancing for load balancer. The graph is plotted for Number of jobs v/s Avg. Maxline of each jobs.



```
56 [label="Feature to split upon : X[0]\nOutput at this node : 2" ];
41 -> 56 [ headlabel="Feature value = 0.0"];
57 [label="Feature to split upon : X[None]\nOutput at this node : 2" ];
41 -> 57 [ headlabel="Feature value = -1.0"];
58 [label="Feature to split upon : X[None]\nOutput at this node : 0" ];
45 -> 58 [ headlabel="Feature value = 0.0"];
59 [label="Feature to split upon : X[None]\nOutput at this node : 1" ];
46 -> 59 [ headlabel="Feature value = -1.0"];
60 [label="Feature to split upon : X[None]\nOutput at this node : 0" ];
47 -> 60 [ headlabel="Feature value = -2.0"];
61 [label="Feature to split upon : X[None]\nOutput at this node : 0" ];
48 -> 61 [ headlabel="Feature value = 0.0"];
62 [label="Feature to split upon : X[None]\nOutput at this node : 1" ];
53 -> 62 [ headlabel="Feature value = 0.0"];
63 [label="Feature to split upon : X[None]\nOutput at this node : 2" ];
56 -> 63 [ headlabel="Feature value = 0.0"];
}
Predictions [0 1 1 0 1 0 2 1 2 1 1 2 1 0 1 2 1 1 1 2 2 2 0 0 2 0 0 1 1 1 0 2 0 0 2 2 2
 0 0 2 2 2 2 0 0 1 1 1 2 2 0 2 1 2 1 2 2 0 1 0 2 2 2 0 0 0 2 1 2 0 0 0 0 0
 0 0 2 0 0 2 1 0 2 1 2 2 1 1 1 0 1 0 0 1 1 0 2 0 0 0]
Score : 0.88
100.86666666666666
Score of our model : 0.88
Score of inbuilt sklearn's decision tree on the same data : 0.88
```

Fig-5.3: The above image describes the prediction score of load balancing

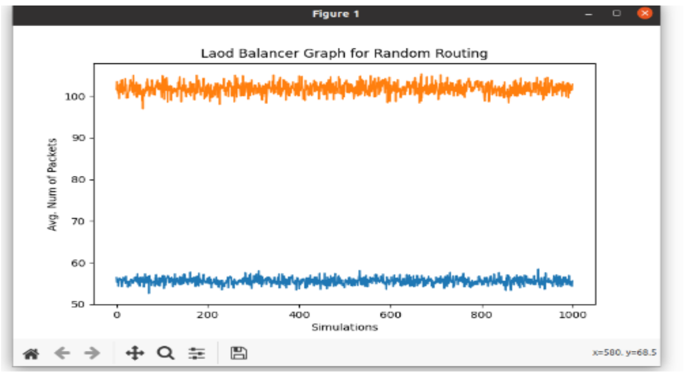


Fig-5.4: This image shows the graph of load balancing for Random routing. The graph is plotted for Number of jobs v/s Avg. Maxline of each jobs.

6. CONCLUSIONS

In Hadoop, there are a variety of load equalization techniques. The conclusions made here are frequently application dependent, and future researchers will examine constants with a variety of application types. The majority of the load equalization analysis work focuses primarily on load equalization during planning [8, 9] or load equalization during MapReduce [11,12]. These methods produce better outcomes, especially in the case of heterogeneous clusters.

REFERENCES

[1] Dharanipragada, Padala.S, Kammili.B and Kumar.V (2017 IEEE) "Tula: A disk latency aware balancing and block placement strategy for Hadoop".International Conference on Big Data (BigData).

[2] Vavilapalli, Seth.S, Saha.B, Curino.C, O'Malley.O, Radia.S, Reed.B, Baldeschwieler.E, Murthy.A, Douglas.C, Agarwal.S, Konar.M, Evans.R, Graves.T, Lowe.J and Shah.H (2013) "Apache Hadoop YARN", Proceedings of the 4th annual Symposium on CloudComputing SOCC '13.

[3] "HDFS Balancers | 5.7.x | Cloudera Documentation", Cloudera.com, 2018. [Online] Available https://www.cloudera.com/documentation/en terprise/57x/topics/admin_hdfs_balancer.html. [Accessed: 07- Jul- 2018]

[4] "Hadoop Fair Scheduler". https://hadoop.apache.org/docs/r2.7.2/hadoopyarn /had oop-yarn-site/FairScheduler.html. Accessed 20 May2017.

[5] Pike.R, Dorward.S, Griesemer.R and Quinlan.S (Oct. 2005) "Interpreting the Data: Parallel Analysis with Sawzall", ScientificProgramming Journal, 13 (4): 227298.

[6] Olston.C, Reed.B, Srivastava.U, Kumar.R and Tomkins.A(June 2008) "Pig Latin: A Not-So-Foreign Language for Data Processing". ACM SIGMOD 2008.

[7] Ucar.B, Aykanat.C,Kaya.K, and Ikinci.M (Jan 2006) "Task assignment in heterogeneous computing systems". J. of Parallel and Distributed Computing, 66 (1): 32-46.

[8] Shanjiang Tang, Bu-Sung Lee, and Bingsheng He, "DynamicMR: A Dynamic Slot Allocation Optimization Framework for MapReduce Clusters," in Ieee Transactions On Cloud Computing, Vol. 2, No. 3, July-September 2014.

[9] Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoic, Y. Lu, B. Saha, and E.Harris, " Reining in the outliers in MapReduce clusters using mantri,"In USENIX OSDI, Vancouver, Canada, October 2010.

[10] Chen, M. Kodialam, and T. Lakshman,"Joint scheduling of processing and shu_e phases in MapReduce systems," In Proceedings of IEEE Infocom, March 2012.