International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958 (Online), Volume-9 Issue-4, April, 2020

Improved Hadoop Cluster Performance by Dynamic Load and Resource Aware Speculative Execution and Straggler Node Detection

Juby Mathew, Terry Jacob Mathew, Thomas Scaria

Abstract**: The big data is one of the fastest growing technologies, which can to handle huge amounts of data from various sources, such as social media, web logs, banking and business sectors etc. In order to pace with the changes in the data patterns and to accommodate the requirements of big data analytics, the platform for storage and processing such as Hadoop, also requires great advancements. Hadoop, an open source project executes the big data processing job in map and reduce phases and follows master-slave architecture. A Hadoop MapReduce job can be delayed if one of its many tasks is being assigned to an unreliable or congested machine. To solve this straggler problem, a novel algorithm design of speculative execution schemes for parallel processing clusters, from an optimization perspective, under different loading conditions is proposed. For the lightly loaded case, a task cloning scheme, namely, the combined file task cloning algorithm, which is based on maximizing the overall system utility, a straggler detection algorithm is proposed based on a workload threshold. The detection and cloning of tasks assigned with the stragglers only will not be enough to enhance the performance unless cloning of tasks is allocated in a resource aware method. So, a method is proposed which identifies and optimizes the resource allocation by considering all possible aspects of cluster performance balancing. One main issue arises due to the pre configuration of distinct map and reduce slots based on the number of files in the input folder. This can cause severe under-utilization of slot as map slots might not be fully utilized with respect to the input splits. To solve this issue, an alternative technique of Hadoop Slot Allocation is introduced in this paper by keeping the efficient management of slots model. The combine file task cloning algorithm combines the files which are less than the size of a single data block and executes them in the highly performing data node. On implementing these efficient cloning and combining techniques on a heavily loaded cluster after detecting the straggler, machine is found to reduce the elapsed time of execution to an average of 40%. The detection algorithm improves the overall performance of the heavily loaded cluster by 20% of the total elapsed time in comparison with the native Hadoop algorithm.

Revised Manuscript Received on April 25, 2020.
* Correspondence Author
Dr.Juby Mathew*, Department of Computer Applications, Amal Jyothi College of Engineering, Kanjirapally, Kottayam, Kerala, India
Dr.Terry Jacob Mathew, Department of Computer Applications, MACFAST Thiruvalla, Kottayam, Kerala, India
Lt.Dr.Thomas Scaria, Department of Computer Science, St.Pius X College, Kerala, India.

Keywords: Big data, Clustering, Hadoop, Node detection

I. INTRODUCTION

Today, the world is guided by data. People as well as machines are generating huge amounts of data every moment by sending messages, uploading videos and photos, generating sensor data from different type of sensing mechanisms etc. The handling of the phenomenal data explosion posed a challenge to technolgical firms such as Google, Yahoo, Amazon, and Microsoft. The companies had to sift and sieve through massive amounts of data to find the customer orientations and preferences related to books, adverts and trending websites. Traditional tools for data handling also failed in this regard. Hence, Google introduced the revolutionary MapReduce system that can handle big data processing. Subsequently, Doug Cutting initiated an open source version of this MapReduce system namely Hadoop.
Apart from the traditional distributed systems, Hadoop differs in the core execution strategy of Data Locality. This indicates that the mode of existence and execution of Hadoop differs from the existing data warehouses and relational databases used for data analytics in the past.

A. MapReduce and speculative execution

In short Hadoop allows the distributed execution of various analytics works in large amount data in a simple and yet powerful manner. The storage and processing is handled by 2 different engines known as HDFS and MapReduce. Hadoop have the data locality feature where the data will reside in the storage platform itself and the program will go down to the data location and executes within. Thus the importance of Hadoop like platform in the rapid growing world is priceless. As the name suggests, MapReduce is implemented as an independent map and reduce phase. MapReduce envisages a model for executing big volumes of data simultaneously by dividing the tasks into standalone groups. The normal speculative execution strategy doesn't have the concept of resource aware scheduling and dynamic and fast detection of stragglers. Thus, in order to mitigate the lagging of job due to straggler node problem and also incorporate the concepts and requirements of the distributed system an effective parallel processing architecture should be developed as part of open source project Hadoop. So, the development of a configuration patch that could rectify these limitations of default speculative execution is relevant.

B. Objective

The objective of this research work is to develop a novel algorithm which is expected to provide enhancements in performance of heavy loaded cluster.
The optimized speculative execution strategy can make great changes in the performance rates of multinode cluster. The detection and mitigation of stragglers in the system has to effectively equipped for obtaining higher throughput.

C. Problem definition

This work focuses on the development of a configuration patch that can provide more performance and throughput for the job running on the Hadoop multinode cluster irrespective of the load of input data and file formats in a resource aware manner. The load balancing is done dynamically by identifying whether the system is lightly or heavily loaded and execute the combine file task cloning algorithm for the lightly loaded condition and for heavily loaded condition detection of straggler node algorithm is performed. These two algorithms are evaluated in the heterogeneous and homogeneous multinode cluster and combine file task cloning is evaluated in the single node cluster for performance evaluation.

D. Scope of the Work

The main challenges in the system development is the possible overhead that can cause due to the number of execution stages. The system has also threats on dependencies of the Hadoop framework as it is tightly coupled system. These dependencies and overheads are to be handled efficiently to achieve a better performance increase on comparing with existing systems. The successful implementation of this system in the heterogeneous cluster which handles instantaneously varying load in the fields like business environments like banking and other machine to human interaction platforms can perform better. High efficiency in the CPU time and execution time can be achieved.

E. Expected Outcome

The core expectation of this work is thorough study of the fundamental concepts and apply them to develop the proposed system. An analysis of the proposed system will be done and duly tabulated, thus allowing us to compare the proposed system with existing techniques. The novel algorithm for the speculative execution is developed and which is expected to show improvements in the performance than the older existing versions.

II. LITERATURE SURVEY

In [1], the main focus is on speculative execution, which handles the straggling problem. Unlike the existing heuristics-related work, this paper presents a hypothetical structure for optimization of solo job queues. The simulation results show the ESE algorithm can reduce the job flow time by 50% while consume fewer resources comparing to the strategy without backup. The results on ESE algorithm are compared with the traditional method without backup and they show that the resources and time for job execution can be reduced by 50%.

[2] This paper proposes a new dynamic method of implementation known as Maximum Cost Performance (MCP). In this novel strategy the total computing expense are divided between tasks, resulting in reduced task completion time and elevated cluster throughput. This synergism in MCP leads to better performance. This method focuses on selectively adding straggler tasks with precision and performs proper follow up on the worker nodes. The tasks are assigned in first come first serve preference.

[3] Combination Re-Execution Scheduling Technology (CREST) is the name of a new strategy to conclude on the best re calculation methodology in a typical MapReduce job. The motive is to reduce the response time which is usually derived as the sum of the longest duration of execution for all map and reduce tasks in a generic MapReduce job.

[4] This paper is a result of a diversified dynamic supposition-oriented job scheduler namely, Hopper. The job is generally triggered with the launch of speculative spawns of jobs in a generic way being a common approach for reducing the impact of stragglers. Due to this the job schedulers are often twixed between choosing dynamic jobs versus original job tasks.

[5] Mantri is the name of a new model for mitigating the outliers in a typical MapReduce networks. This work introduces the first approach to study a large production Map-Reduce in a cluster form. The core of Mantri's benefit is the amalgamation of stable definite knowledge of job structure with the dynamically available job progress cards. This mechanism is sure to pick the outliers at an early stage and exerts cause-specific alleviation of jobs based on the cost benefit analysis.

A new method of scheduling dynamically generated job clusters for better job approximation is tested in [6]. The authors have put forward a simple and analytical implementation, specifically derived from the dynamic algorithm known as GRASS. GRASS explores the total opportunity cost in deciding the time of speculation of a job. The decision revolves around the early time to determine the execution of the job and moves to more aggressive dynamic methods of speculation as the job enters the final phase of its approximation bound. The proposal was tested in Hadoop and Spark implementations, deployed on a large cluster node and resulted in approximately 47% improvement in finishing the job deadlines. The total time to complete the error some jobs showed around 38% improvements in data provided from Facebook and Bing.

Literature Summary

From the study of literature review the existing systems failed to enhance the performance of tasks in a speculative strategy by a greater performance which can be achieved by calculating the job service matrices and additional parameters like job flow time and computational cost. The dynamic resource allocation capabilities of MapReduce structure are also not established to the level where task cloning and its effective allocation is maintained simultaneously. In contrast, the obtained results do not support the general rules, when the job servicing time is in accordance with the heavy-tailed distributions such as the Pareto Distribution. Thus, they are out of scope of this work. Thus, the foundation for the proposed system lies in these aspects of considering these crucial parameters. The optimization of speculative execution procedures combined with dynamic slot allocation refines the speculative strategy of Hadoop in all aspects. So it is evident that the proposal for such a system is relevant in the big data era.

III. SYSTEM MODEL

In our implementation, we take a colossal data processing job cluster with M servers (machines). The set of jobs J ={J1, J2,… JN } approach the job cluster at a rate of n jobs per unit time and the time at which job Ji arrives is denoted by ai. As the job arrives, the job Ji is added to a public queue managed by the speculative scheduler, ready to carry out the job execution. The set of Jobs, Ji is a deterministic number composition of mi tasks. We assume that δji depicts the jth task of job Ji. We also assume that each server can only execute only one job task at any given time. A random variable, Xji, denotes the service time (i.e., duration) of task δji without any dynamic projection of job completion time. For all jε{1,2,…m_i},X^j_i follows the same distribution, which is characterized by the cumulative distribution function (CDF),F_i (x), i.e., Pr($X^j_i < x$) = $F_i(x)$.

A. Job service process under speculative execution

For the sake of assumption, we take the time is divided and job task preemption is not asserted in order to reduce the system overhead. The job of the scheduler is to make optimal speculative decisions, so that the unused time is approximately nil on job task execution machines. The number of copies on idle machines is also decided by the scheduler in a similar manner at the start of the time slot. Let c^j_i denote the total number of copies made for task δ^j_i where the kth copy is launched at time $w_i^{j,k}$. Thus, we have the following constraint for $w_i^{j,k}$

All the scheduling variables are summarized in Table 1.

Table 1: The notations for the job service parameters

Notation	Corresponding meaning
λ	Job arrival rate
a_i	Arrival time of job J_i
m_i	Number of tasks consisted in job J_i
C_i	Time when job J_i completes its work
X_i^j	Service time of task δ^j_i without speculative execution.
$w_i^{j,k}$	Time when the kth copy of task δ^j_i is scheduled.
$F_i(x)$	The CDF function of X_i^j

B. Problem formulation

In this area of problem formulation, the stress is put on basically two performance measures --the job flow time, i, and the computation cost, both of which are computed by the total time spent on the job servers.

As a general situation, the two performance metrics are often rigid to be rearranged at the same time (except for the detection approach). Hence to solve this crisis, a utility function is defined for each job task as a trade-off between

$$min_z \Sigma_{i=1}^{N} E[\Gamma_i] + \gamma . \sum_{i=1}^{N} \sum_{j=1}^{m_i} \sum_{k=1}^{c_i^j} (E[C_i^j] - w_i^{j,k})$$

these two metrics. This function does the job of maximizing the total utility scale of all jobs in the data cluster by means of finding z. The resulting optimization problem can be represented as:

C. Deriving the cut-off threshold for different Operating regimes

In determining the threshold, a generic approach is to find the approximation solutions, which is from finding the dynamic speculative execution methodology along with the scheduler. This is also in view of the strongly NP-hard nature of the problem. The possible two classes of dynamic strategies for execution are applied here namely, the Cloning and Detection approach. The Cloning strategy stimulates all job tasks in parallel without which the priority among job tasks are lost leading to futile usage of resources. This is the only applicable scheme for a lightly loaded cluster. In contrast, the Straggler-detection methodology produces new spawns of jobs in an intelligent manner so as to handle any load balancing situations.

For further exploring the the details of the proposed methodology, it is necessary to define the cutoff workload threshold, λU , whose job is to segregate the remaining analytical stage into an easy handled job; in comparison with the heavily loaded server data clusters.

1. A First upper bound for λ^U

To keep the system not overloaded, the job arrival rate must be bounded by the job processing rate, which then yields the first upper bound, λ_1, for λ^U, i.e.,

$$\lambda_1 = \frac{NM}{\sum_{i=1}^{N} \sum_{j=1}^{m_i} c_i^j E[T_i^j]}$$

2.A second upper bound for λ^U

The efficiency of cloning is not guaranteed by the single upper bound. An efficient cloning strategy should have a smaller task delay than a strategy which does not make speculative execution. The second upper bound and it can be shown as:

$$\lambda_2 = \frac{\lambda_i^* M}{m^{‘}}$$

D. Optimal cloning in lightly loaded Regime

In a lightly loaded cluster, i.e., $\lambda < \lambda^U$, we maximize the overall system utility in P1 by coordinating job scheduling with task cloning. The lightly loaded conditions always suites with a smart cloning of outlier tasks rather than detection based approach.

So we introduced a combine file task cloning algorithm for the cloning the tasks in straggler machine and reallocates to other machines.

1. The design of the Smart Cloning Algorithm (SCA) in a lightly-loaded cluster

After successfully executing the cloning algorithm, the next focus is on the tracking of the job progress as well as job completion cost. This is done with the help of an algorithm, which calculates the solid integer part of the task progress rate, such that the task job counts are also equally integer parts. But, we detect a case in analysis, where the provision of job cloning is limited by space for some specific time slots. i.e, $\sum_i^k mli > N(l)$. This rare situation demands a careful study in a lightly loaded data cluster. At this point it is not wise to solve P2. Instead, the new proposal suggests a design alternative dynamic scheduling scheme to allocate job clusters, based on a smallest remaining workload scheme, which is extended from SRPT scheduler.

2. Design of a straggler-detection-based Algorithm for the heavily loaded regime

For a heavily loaded cluster, i.e. ,$\lambda \geq \lambda^U$, the cloning-strategy is not viable as there is no scope to make a copy for all tasks. To avoid this drawback, we devise a detection methodology for obtaining approximate results. The primary dynamic execution strategy is laden with numerous flaws in principle. The first drawback is that, it creates extra copies for the tasks that are in conservative mode, characterized by lower amounts of resource consumption. The second drawback is the lower precision in the estimation, which falls heavily on the job completion duration and scale.

IV. IMPLEMENTATION

A. Implementation Tools

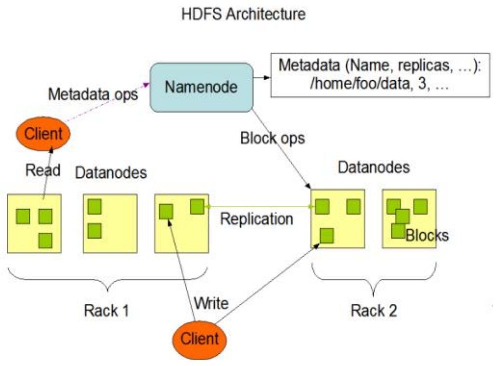
The usage of software and hardware tools are most important elements of setting up of a heterogeneous multimode cluster. The system is aimed to implement with the Ubuntu OS support. The software and hardware requirements can be pointed out as:

B. Software Requirements

1. Hadoop framework - Hadoop 2.7.2:

Apache Hadoop is a popular open-source framework used extensively for distributive storage and processing of massive data. It is a mix of server clusters where the motivation behind its construction lies in the fact that the hardware failures are quite common and should be handled dynamically by the server mass in the framework. The Fig 1 depicts a typical Hadoop framework..

Figure 1: Hadoop Architecture



The multinode cluster is formed in a master-slave architecture where each machine in the cluster is installed with Hadoop properly and set one among them as Master, thus Name node and provides a Resource manager. The slave nodes act as data nodes and they start a Node manager.

2. Java - 1.8.0.91:

JDK is downloaded from the official site and the java coding is done and compiled in Eclipse 4.4. Java is the core language of Hadoop framework. java is used for the whole implementation of the algorithms in this work.

3. Eclipse:

The popular platform - Eclipse provides IDEs and platforms for any amicable framework, irrespective of the language and scheme. The Java IDE, C/C++, JavaScript and PHP IDE's are built on these platforms for creating typical desktop, Web and cloud IDEs.

4. Cloudera:

Cloudera's is another popular open-source Apache Hadoop distribution. Also known as Cloudera Distribution Including Apache Hadoop (CDH), this framework is meant for corporate deployments of applications in a massive scale. According to Cloudera, the major share of its engineering output is designated upstream to various Apache-licensed open source projects that work on the common Hadoop platform. Cloudera quick start virtual machine is used for the compilation purpose, which comes with inbuilt packages of Hadoop and its daemons. Thus it is used for the survey of elapsed time of execution of various matrices considered in the project.

5. Oracle VM VirtualBox:

Oracle VM VirtualBox (formerly owned and managed by Sun VirtualBox, Sun xVM VirtualBox and Innotek VirtualBox) is a free and open-source hypervisor for x86 computers currently under the flagship of Oracle Corporation. It was developed initially by Innotek GmbH and was later on acquired by Sun Microsystems in 2008 which in taken over by Oracle in 2010.

C. Hardware Requirements

The hardware requirement is a heterogeneous multimode cluster where the big data analytics and processing is being done. For simulation and testing, a multimode Hadoop Cluster consisting of 3 machines are used. The configurations of the machines are:

- Processors: Any Intel or AMD x86 processor.
- RAM: 3GB.
- System Type: 64-bit OS, x64-based processor.
- Disk Space: 20GB in C drive for reserved for cluster job execution
- Virtual Machine Specifications
- Quick Start VM 5.5 : Red Hat (64 Bit), 6 GB RAM, 64 GB virtual hard disk space.
- Hadoop Cluster nodes with CentOS Minimal version.

2.Module Description

The first stage of project development the multinode cluster of Hadoop with 3 nodes with 91.5Gb of shared HDFS in each machine, then uploads the 155Mb data as a sample for simulation and it can be extended to great ranges in gigabytes, which will be evenly split and replicated automatically.

Then executedWordCount program which includes map and reduce functions and submits the job. Once the job completes, it will be notified of the results. The log details are analysed and sorted for the node failure and decommissioned reports. The data transfer details are analysed to find the network accessibility between the machines, within the cluster.

3. Implement and Evaluate performance of WordCount with optimized SCA & ESE algorithm

Formulate the code for Smart Cloning Algorithm and Enhanced Speculative Execution Algorithm. Generate the patch of optimized speculative execution. Generate its patch file and add with the Hadoop Framework. Check its performance with word count running in 3 node cluster. The algorithm pseudo code of SCA and ESE can be explained as:

Algorithm 1: Smart Cloning Algorithm

....

```
Input: The jobs in the cluster associated with their running status at time slot l
Output: Scheduling decisions for time slot l.
1. schedule the unassigned tasks of the running jobs in the cluster with the fewest remaining first;
2. update N(l) and $\chi$ (l);
3. if N(l) == 0 then
4. return;
5. if $\sum_i^k mli > N(l)$ then
6. for Job $J_{li}$ in $\chi$ (l) do
7. assign only one copy for each task of $J_{li}$ and update N(l);
8. if N(l) == 0 then
9. return;
```

Algorithm 2: Enhanced Speculative Execution Algorithm

....
..

```
Input: The jobs in the cluster associated with their running status at time slot l;
Output: Scheduling decisions for time slot l.
1. Count N(l), the number of idle machines at time slot l and update D(l), R(l), $\chi$ (l).
2. for task $\delta_i^j$ in D(l) do
3. Assign a duplicate of $\delta_i^j$ on a random idle machine;
4. N(l) == 1;
5. if N(l) == 0 then
6. return;
7. for job $J_i$ in R(l) do
8. Assign the unscheduled tasks of $J_i$ on idle machines;
9. update N(l);
10. if N(l) == 0 then
11. return;
12. for job $J_{li}$ in $\chi$(l) do
13. Assign one copy for each task in $J_{li}$ ;
14. update N(l);
15. if N(l) == 0 then
16. return;
17. return;
```

4. Detailed Analysis in loading conditions, different programs and performance optimization

Evaluate and analyse the heavily and lightly loaded conditions of the cluster. Overall performance Tuning and employing manual network contention in data nodes. Detailed performance analysis using several classical MapReduce Program along with word count. Performance analysis by using Spark by parallelism tuning. Performance Evaluation is to be sketched in detail to analyse the enhancements. Overall validation test should be performed with the system. The performance enhancement can be done after the validation testing apart from the module wise testing.

V. TESTING AND EVALUATIONS

A.Key Tuning Parameters

1 Mappers

mapreduce.input.fileinputformat.split.minsize : The minimum size chunk that map input should be split into. By increasing this value beyond dfs.blocksize, it can reduce the number of mappers in the job. This is because if the value of mapreduce.input.fileinputformat.split.minsize to 4x dfs.blocksize, then 4 times the size of blocks will be sent to a single mapper, thus, reducing the number of mappers needed to process the input. The value for this property is the number of bytes for input split. Thus to set the value to 256MB, should specify 268435456 as the value for this property.

mapreduce.input._leinputformat.split.maxsize : The maximum size chunk that map input should be split into when using CombineFileInputFormat or MultiFileInput- Format. By decreasing this value below dfs.blocksize, and increase the number of mappers in the job. This is because if the value of mapreduce.input fileinputformat.split.maxsize to 1/4 dfs.blocksize, then 1/4 the size of a block will be sent to a single mapper, thus, increasing the number of mappers needed to process the input. The value for this property is the number of bytes for input split. Thus to set the value to 256MB, specify 268435456 as the value for this property. If it is set with a max split size when using CombineFileInputFormat, the job will only use 1 mapper.

2 Reducers

mapreduce.job.reduces : One of the biggest killers for workflow performance is the total number of reducers in use. Use too few reducers and the task time is longer than 15 minutes. But too many also causes problems! Determining the number of reducers of individual jobs is a bit of art. But here are some guidelines to think about when picking the number: More reducers = more files on the name node. Too many small files bogs down the namenode and may ultimately make it crash. So in order to reduce output is small (less than 512MB), it needs only fewer reducers

More reducers = less time spent processing data If there is too few reducers, the reduce tasks may take significantly longer than they should. The faster the jobs, reducers run more jobs we can push through the grid.

Shuffling is expensive for large tasks for the FileSystem Counters of the job, it can be observed how much data may potentially need to be pushed around from node to node. Let's take a job with 20 reducers. Here are the FileSystem Counters:

mapreduce.job.reduce.slowstart.completedmaps : This setting controls what percentage of maps should be complete before a reducer is started. By default, we set this to .80 (or 80%). For some jobs, it may be better to set this higher or lower. The two factors to consider are: If the map output is significant,

it is generally recommended that reducers start earlier so that they have a head start processing. If the maps tasks do not produce a lot of data, then it is generally recommended that reducers start later. A good rough number is to look at the shuffle time for the first reduce to fire off after all the maps are finished. That will represent the time that the reducer takes to get map output. So ideally, reducers will fire off (last map) - (shuffle time).

3.Compression

mapreduce.map.output.compress : Setting this to true (default) will shrink map output by compressing it. This will reduce internode transfers, however care must be taken that the time to compress and uncompress is faster than the time to transfer. For large or highly compress-able intermediate/map output, it is usually beneficial to turn on compression.
This can reduce the shuffle time and make disk spills faster. For small intermediate/map output datasets, turning intermediate output compression off will save the CPU time needed to do the (ultimately useless for this data) compression. Note that this is different than apreduce.output.fileoutputformat.compress; that setting controls whether the final job output should be compressed when writing it back to HDFS

4.Memory

mapreduce.(map-reduce).memory.mb : One of the features in newer releases of Hadoop is memory limits. This allows for the system to better manage resources on a busy system. By default, the systems are configured to expect that Java tasks will use 1GB of heap and anywhere from 0.5-1GB of non-heap memory space. Therefore, the default size of mapreduce.(map-reduce).memory.mb is set to 2GB. In some situations, this is not enough memory. Setting just Xmx will result in more than 2GB and the tasks will get killed. Therefore, in order to request more memory for the task slot it needs to adjust both the Xmx value and the mapreduce.(map-reduce).memory.mb value.

5. Advanced

Controlling the number of spills / io.sort.record.percent : io.sort.record.percent controls how much of the circular buffer is used for record vs. record metadata. In general,it and a family of tunables are ones to look at when spills are out of control. Changing this results in maps running faster and fewer disk spills because io.sort.mb is used more efficiently; we do not hit the 80% mark in the metadata buffer as quickly. The end result of changing io.sort.record.percent was that many maps did not spill to disk at all and of those that did, many dropped spilled to 55% fewer files. End result: system thrash was reduced-we saved 30% of the CPU and dropped 30 minutes off the runtime.

mapreduce.(map-reduce).speculative : Set these properties to false in order to prevent parallel execution of multiple instances of the same map or reduce task. The data skew of the mappers or reducers will take significantly longer. In this case, should disable speculative execution to prevent spawning lots of unnecessary map and reduce instances.

6. Running of Algorithm

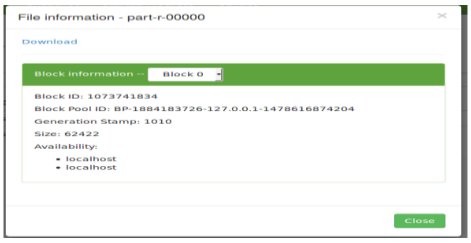
As a detailed survey of different performance factors in Hadoop Yarn. A test input for wordcount problem is selected as a 3.6GB CSV file.
It is analysed in various conditions

- Normal wordcount without Combiners
- Wordcount with Combiners
- Multiple Reducers
- Input Splits
- Speculative execution property disabled

VI. EXPERIMENTAL RESULTS

As part of the straggler machine detection and task cloning strategy the setting up of multinode cluster is the preliminary stage of the project. The evaluation is based on the heterogeneous cluster performance. The project implementation is planned such that the first module of the project is setting up of homogeneous and heterogeneous multimode cluster and evaluating a MapReduce program to check the performance variation due to the system resource utilization and availability constraints.
The experimental results are achieved from the execution of the classical MapReduce program WordCount in the 3 - node cluster and an input of 155Mb of text data. The step-by-step evaluation can be described as :

- ✓ Multinode cluster with 3 nodes are setup in the lab with one server and 2 slave machines.
- ✓ All the 3 nodes where properly installed with Hadoop and relevant set up procedures where followed to establish the master - slave architecture with 3 machines in the lab with LAN and ssh connectivity.
- ✓ Secret ssh keys where generated and shared with all the 3 systems for the communication.
- ✓ Master node in the cluster where set up with a shared HDFS memory capacity of 95 GB in the drive and slaves with 65 GB of space for the distributed access.
- ✓ After the setting up of the 3 node cluster the namenode is formatted and started the utilities and datanode.
- ✓ The slave machines are checked for the datanode working and found it working by the 'jps' command which showed the activated components as Datanode and Node-manager at slave machine and all the other components of Hadoop like Namenode,Secondary Namenode, Resource Manager, Job Tracker are all active the master ma-chine or node.
- ✓ Created a folder in HDFS and loaded input file of size 155Mb.
- ✓ Run the classical problem WordCount in the master which internally utilized the other 2 datanodes and the output folder (figure 2) is generated at the HDFS and the folder contained the text file with counts of all the words in the text input.
- ✓ The report is analyzed from the web. The failed and decommissioned datanodes are checked.


File information - part-r-00000

Block information -- Block 0

Block ID: 1073741834
Block Pool ID: BP-1884183726-127.0.0.1-1478616874204
Generation Stamp: 1010
Size: 62422
Availability:
- localhost
- localhost

Figure 2: Details of the output folder

Thus obtained a fair result of the first module and the works of next module is going on but it is facing some unexpected errors. It is being tried out to solve them and expecting a good results. For a simulated development environment, further proceeding is done by Oracle Virtual Box and created a cluster with a namenode and 3 datanodes and a client machine, all with static IP address. Developed programs for obtaining the different cluster performance impacts of key tuning parameters. Observed the difference in the time of completion of jobs in cluster for each parameter. The detailed tabular results can be reviewed from Table 2 and 3.

The survey of elapsed time is done in the classical Word Count (WC) program and the input file is given as the 3.65 Gb CSV file.

TABLE 2: CLUSTER REPORT

Node ID	Memory	Core Processor	OS Ram Size
Namenode- nn	20 Gb	i3	CentOS(Minimal version) 2 Gb
Datanode1- dn1	20 Gb	i3	CentOS(Minimal version) 2 Gb
Datanode - dn2	20 Gb	i3	CentOS(Minimal version) 2 Gb
Datanode3- dn3	20 Gb	i3	CentOS(Minimal version) 2 Gb
Client -vclient	20 Gb	i3	CentOS(Minimal version) 2 Gb

TABLE 3: EXPERIMENTAL REPORT OF JOB COMPLETION TIME WITH VARIOUS FACTORS OF PERFORMANCE

Application Details	Platform	Time of Completion
WC without Combiners	Cloudera(4 Gb RAM)	156803 ms
	Multinode Cluster	162970 ms
WC with Combiners	Cloudera(4 Gb RAM)	155542 ms
	Multinode Cluster	158746 ms
WC with multireducers and Combiners	Cloudera(4 Gb RAM)	103368 ms
	Multinode Cluster	143659 ms
WC with Inputsplits - 1 Gb	Cloudera(4 Gb RAM)	135888 ms
	Multinode Cluster	150035 ms

VII. CONCLUSION

Hadoop, the open source framework for distributed data processing acquired much production importance as it provides data locality features and efficient processing platform for huge file processing than by using traditional distributed systems. Thus it should be much accurate and dynamic according to the applications so that it will be a tunable processing approach. An approach for the speculative execution procedure enhancement is proposed by the work where this approach is proved to enhance overall cluster performance and thus the overall execution time of the bulk of jobs. In the execution proces of the work is done in two phases.

In Phase-1 the parallel execution of MapReduce with example program Wordcount is observed in a multinode cluster of homogeneous as well as heterogeneous nodes in virtual machines and real machines with quad core processor. The dynamic slot allocation procedures are executed within the fair schedule module of the speculative execution and yarn common files.

The data locality is one of the main concerns the dynamic slot allocation is based on the data aware allocation and reallocation of map and reduce slots. The combining algorithm offers about 60% of average performance enhancement in the cluster for the word count program. The detection of straggler node module will offer a dynamic notification of outlier nodes in the cluster and decommission of them at the time of detection itself.

It is found to create some overload on cluster while executing detection algorithm but in the case of heavily loaded criteria it is negligible comparing to the estimated time of execution without eliminating the straggler nodes. The modified algorithms are expected to give an optimized result in the overall performance of the MapReduce system. Thus the speculative execution can be implemented in a very efficient manner. Then it is can be applied to get added along with the Hadoop package. Then with a single command speculative execution procedure can be enabled or disabled by the common users.

The speculative execution as well as scheduling strategies in Hadoop needs more efficiency in the big data platforms as a small degradation of resources may lead to heavy production lose. The commodity hardware is very prone to damages, bandwidth scarcity, bad machine faults in the overall cluster. The extensions to the project focusing on the energy impact-oriented enhancement in smart speculation can reduce overhead due to task cloning and thus can achieve much more reliable distributed platform.

REFERENCES

1 Huanle Xu, Wing Cheong Lau, “Optimization for Speculative Execution in Big Data Processing Clusters," in Transactions on Parallel and Distributed Systems.
2 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.
3 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.
4 Chen, M. Kodialam, and T. Lakshman,”Joint scheduling of processing and shu_e phases in MapReduce systems," In Proceedings of IEEE Infocom, March 2012.
5 Q. Chen, C. Liu, and Z. Xiao,”Improving MapReduce performance using smart speculative execution strategy," IEEE Transactions on Computers, 63(4), April 2014.
6 M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly,` Dryad: distributed data-parallel programs from sequential building blocks," In EuroSys, March 2007.
7 X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu,”Hopper: Decentralized speculation-aware cluster scheduling at scale," In Sigcomm, August 2015.
8 Lei LEI, Tianyu WO, Chunming HU,”CREST: Towards Fast Speculation of Straggler Tasks in MapReduce,2011 Eighth IEEE International Conference on e-Business Engineering"
9 Faraz Ahmad, Srimat Chakradhar, Anand Raghunathan, T. N. Vijaykumar, Tarazu: Optimizing MapReduce On Heterogeneous Clusters,"ASPLOS12 March 3-7, 2012, London, England, UK.
10 Qi Liu, Weidong Cai, Jian Shen, Zhangjie Fu, Nigel Linge,”A Smart Speculative Execution Strategy based on Node Classi_cation for Heterogeneous Hadoop Systems,"Jan. 31 , Feb. 3, 2016 ICACT2016.
11 Huanle XU, Wing Cheong LAU,”Task-Cloning Algorithms in a MapReduce luster with Competitive Performance Bounds,"Ieee Transactions On Computers, Vol. 63, No. 4, April 2014.

12 Juby Mathew, R Vijaya kumar,Multilinear Principal Component Analysis with SVM for Disease Diagnosis on Big Data,IETE Journal of Research,1-15,Taylor & Francis[2019]

13 Ananthanarayanan, M. C.-C. Hung, X. Ren, and I. Stoica, Grass: Trimming stragglers in approximation analytics, In NSDI, April 2014.

14 Ganesh Ananthanarayanan, Ali Ghodsil, Scott Shenker, Ion Stoica, E_ective Straggler Mitigation: Attack of the Clones, In 10th USENIX Symposium on Networked Systems Design and Implementation, 2013.

15 Tien-Dat Phan, Shadi Ibrahim, Gabriel Antoniu, Luc Bouge, On Understanding the Energy Impact of Speculative Execution in Hadoop,, IEEE International Conference on Data Science and Data Intensive Systems, 2015.

16 Ganesh Ananthanarayanan, Ali Ghodsi1, Scott Shenker, Ion Stoica. E_ective Straggler Mitigation: Attack of the Clones, In 10th USENIX Symposium on Networked Systems Design and Implementation, 2013.

17 Masatoshi Kawarasaki, Hyuma Watanabe, System Status Aware Hadoop Scheduling Methods for Job Performance Improvement, In 10th USENIX Symposium on Networked Systems Design and Implementation, 2013.

18 S. Khalil, S. A. Salem, S. Nassar and E. M. Saad, Mapreduce Performance in Heterogeneous Environments: A Review, International Journal of Computer Applications, December 2016.

AUTHORS PROFILE

Dr Juby Mathew is a Dynamic, Resourceful Teaching Professional. He received his PhD and PostDoc in Computer Science from Mahatma Gandhi University, Kottayam. He pursued his MCA from Periyar University, Salem, and M.Tech from MS University, Tirunelveli. So far he has published his articles in 12 international Journals and presented papers in more than twenty National and International Conferences. At present, he is working as an Associate Professor in the Department of Computer Applications at Amal Jyothi College of Engineering, Kanjirapally and Research guide at Kerala Technological University,Trivandrum.He won Best Faculty award as a result of his proven ability to enhance students performance, promising to shape a better world for the students and empower them with knowledge. He has reviewed many paper publications and journals and PhD thesis within an incredibly short period.

Dr. Terry Jacob Mathew is an Associate Professor at MACFAST, affiliated to Mahatma Gandhi University, Kottayam, India. He received his Ph.D. degree from school of computer sciences, Mahatma Gandhi University, Kottayam and has worked with the industry and academia since 2005. His research interests include decision making systems, data mining, soft sets, fuzzy topology and predictive medical analytics.

Lt. Dr Thomas Scaria working as assistant professor and Head in the department of Computer Science at St.Pius X College, Rajapuram and at the same time he is the chairman of Computer science board of studies of Kannur University. He published so many papers in Scopus and SCI indexed Journals.