UiO : Department of Informatics
University of Oslo

In the Quest of Trade-off between Job Parallelism and Throughput

Adaptive Parameter Tuning of Hadoop

Ramesh Pokhrel

Master's Thesis Spring 2018

In the Quest of Trade-off between Job Parallelism and Throughput

Ramesh Pokhrel

22nd June 2018

Acknowledgement

First and foremost, I would like to express my gratitude to my thesis supervisors Ashish Rauniyar and Anis Yazidi for their expertise, guidance, feedback, and inspiration which were the key to successful completion of this thesis. Their encouragement helps me to drive the project down to the right path.

I would also like to thank Hårek Haugerud for his all the administrative support and suggestions.

Finally, I would like to extend my gratitude to my family for their all the support and understanding throughout the master program.

-Ramesh Pokhrel

Abstract

Big data is an emerging concept involving complex data sets which can give new insight and distill new knowledge. In the other hand, Hadoop MapReduce paradigm, a distributed computing software, has been adopted widely in the big data community for large-scale processing. It is known that the implementation of MapReduce with the default configuration results in less number of parallel running job and thus waste of resources in the cluster during MapReduce operation. In fact, poor resource utilization and overall low performance is usually induced by the default configuration during the run-time.

This thesis investigated how the cluster resources can be optimally and appropriately utilized during MapReduce operation in order to result in a better job parallelism and throughput. To achieve this, an optimal design is developed which can dynamically change the resources allocation by changing the system level parameters at run-time. The project results showed that the resources are optimally and appropriately utilized during job execution which resulted in better job parallelism and throughput.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The term *Big Data* stresses the enormous amount of complex and fast expanding data sets from several sources. Because of the tremendous increase in the number of datasets within the last few years, big data and its investigation have received a lot of research attention. Therefore, big data is being saved, processed and examined in order to get effective potential new insight. Additionally, big data implements advanced instruments and methods so as to capture, store, distribute, process and examine bigger datasets.

Over the last few years, Hadoop technology has undergone an unprecedented growth in terms of adoption to satisfy the challenges of accessing and processing these bigger datasets [22]. Hadoop is an open source application project under Apache, which modulates distributed computing environment for large datasets throughout the bunch of merchandise servers. Similarly, Hadoop represents a programming framework that breaks down a program into distinct equivalent pieces of sub-tasks [21]. Mappers in the worker nodes convert data to the collection of key-value pairs from the bunch. Further, in reduction stage the master nodes from the cluster quicken the output produced by the worker nodes in order to aggregate, mix, filter, and change functions to form a final output.

A general characteristic of Hadoop MapReduce framework is a resource management approach which relies on predefined resource units that have to be divided among many jobs. Usually, the size of the container and the total amount of resources unit available in the cluster are static in nature. Additionally, these amount of resources in the cluster can be determined before the cluster creation and is usually unchanged during the MapReduce job execution. This kind of static resources management in Hadoop cluster has limited ability to cope up with increasingly diverse applications and dynamic computing environments, resulting in poor resource utilization and low-performance [24].

Several challenges have been confronted by Hadoop application developers despite its prevalence. Especially, the researchers have discovered that Hadoop cluster configuration can influence the performance which can be delivered into the applications. Researchers have shown that a very small change in a configurable parameter can create the huge difference in the operation when running the same MapReduce job with the exact same size of input data. Therefore, Hadoop performance is mainly influenced by two types of parameters settings, i.e., system and job level parameters. Additionally, Hadoop has distributed system using its black-box like feature. So, it is remarkably hard to find appropriate mathematical model which can configure Hadoop cluster for the specific job. After all, it is injudicious to use the exactly same configuration to all sort of MapReduce tasks. Hadoop supplies over 200 tunable parameters, therefore manual proper tuning of the parameters is time-consuming and hard as most parameters in Hadoop data analytics framework have complicated inter-dependencies between them. Thus, setting the optimum value properly for all those tunable parameters in the cluster needs a substantial understanding and sufficient degree of expertise [37].

Especially, *YARN* (Yet Another Resource Negotiator) in Hadoop (version 2.0) comprises the amount of program level parameters that are responsible for controlling how MapReduce jobs are scheduled in the cluster which can put direct influence on the job performance. *Maximum Application Master Resource in Percent (MARP)* is one of the property that consists its value at floating point number (percent) which directly impacts the number of concurrently running MapReduce jobs versus corresponding throughputs [36]. Moreover, configuring improper MARP results in a decrease the number of jobs running in parallel which in turn results in increase in ideal resources unit in the cluster or reduce in the number of map-reduce tasks. Thus, this leads to an increase in overall job completion time. Furthermore, in order to achieve cluster performance, configuring appropriate MARP is far from trivial. On the other hand, one MARP value might work fine for one job while working worst for another MapReduce job due to diversity of jobs and workloads that results in overall performance degradation.

To address these limitations this thesis introduces an approach *Adaptive Parameter Tuning Of Hadoop (APTH)* which can dynamically change MARP value online in order to balance the job parallelism and associated throughput in the Hadoop cluster.

1.1 Problem Statement

This thesis reveals the way that leverage cluster resources during the runtime in order to maximize the number of parallel running MapReduce jobs and corresponding throughput. The main aim of this thesis is to find a way of reducing the number of ideal resources in the cluster during the runtime of MapReduce jobs. Therefore, this thesis addresses the following challenge:

How the number of parallel running MapReduce job and associated throughput can be increased online by utilizing the appropriate optimum resources unit in the Hadoop cluster?

Chapter 2

Background

2.1 Big data

Big data is a term designing massive data sets having large, more varied and complex structure with the difficulties of storing, analyzing and visualizing for further processes or results [28]. Use of the ever-increasing number of electronic devices in a broad range of applications areas are regularly being generated, collected and flowing data at unprecedented scale. In another word, the world is afloat in digital data ocean. These data may be from sensors in the context of Internet of Things (IoT) used to collect climate information, post to the social media site, digital photos and videos posted online, mobile GPS signal [1] and many other resources that generates data. Indeed gathering and storing these large amounts of data for eventual use is a *Big Data*.

The main characteristics of big data are *volume*, *variety* and *velocity*, or the three **V's**. Data volume refers the size of data in terms of standard information metrics such as terabytes (TBs) and petabytes (Pbs). Petabytes data sets are common these days. 72 hours of video are uploaded to You Tube every single minute [1] is one of the example, how data volume is increasing at data warehouse. Data volume play important role in storage and processing of data. However, due to less price per gigabyte and huge storage capacity in the cloud, storage capacity issue has become less pressing [11].

Velocity refers to the rate of data change, or how frequent data are created and delivered. The leading edge of big data is streaming data, which is collected into the server in real time [10]. In addition, 140 million tweets per day on average can be a good example of data velocity [2].

Variety, as other characteristics, refers to the diversity of data. Various kind of data generator source generates data in various form. Single mobile device is enough to generate multiple types of data, such as voice, images, videos and etc. This diversity in data is responsible to make big data really big. Text, human language, audio, video, logs, clickstreams, XML further can be categorized

[1]https://www.datasciencecentral.com/forum/topics/the-3vs-that-define-big-data

[2]https://blog.twitter.com/official/en$_{u}s/a/2011/numbers.html$

as structured, semi-structured and unstructured data. Moreover, multi-dimensional data can be stored and drawn from the data warehouse in a systematic way is an exciting area of research.

The use of big data is a key basis that can unlock significant values by making the information transparent and usable at the much higher frequency. As individual firms generate and store data in digital form, they can extract much accurate inventories and customer behavior. Therefore, it is easy to make decisions and boost business. Indeed big data can act as a power of competition and growth for every single organization. Finally, big data can be the most significant commodity which helps to improve and develop the next generation product and services [26].

2.1.1 Big Data Processing and Challenges

Data only stored as a raw fact doesn't have any point until they get processed and analyzed. Processing such a big dataset in an efficient way is a clear need for any user. Big data sets requires sophisticated application and software in order to process and analyze those voluminous data. Therefore, traditional applications and software used for data processing are almost outstripped as data volume increase. Big data processing itself is a challenging job. Moreover, high-performance network, disk, memory and CPU's are the most important hardware factor which accounts for overall data processing time and cost. First data will be loaded for processing but at the same time if the network traffic and disk interferes with another job then the time for data loading goes high. It is very necessary to minimize the time consumption by data loading job. Every user needs their data processing results in short time, if possible in real-time. Real-time response is another critical factor as many jobs and queries might be response-time critical in order to satisfy heavy workloads.

2.1.2 Big Data Analytics Framework

Data are processed to find hidden insight in the data. Big data concerns the huge amount of heterogeneous datasets from autonomous sources. To store and analyze such massive complex datasets, many specialized analytical platforms come in various form, from software to analytical services that run in third-party hosted environment [5]. Currently, the number of analytical platforms are available in the market in order to analyze complex unstructured, semi-structured and structured datasets. Hadoop, for instance, is one of the top framework used today. Not only Hadoop as a big data analytics framework but also Spark, Flink, Storm, Samza and many more are trying to get adopted in the same industry [32].

2.1.3 Big Data and Clouds

In general, the process of extracting valuable information using analytics framework over complex datasets is not easy due to data characteristics as volume, velocity, and variety. In addition, big data analytics require compute-intensive data mining algorithms which need high-performance memory and processor to produce results [29]. Moreover, infrastructure provided by cloud computing platform can serve effective and efficient addressing for both data storage and analytic processing. The rise of big data cloud computing and cloud data stores has significant advantages over the traditional physical deployments. On the other hand, pay-as-you-go model as a facilitator of the cloud platforms makes the user enjoy more with high-performance infrastructure with low cost [3]. Thus, unnecessary burden of the cost and hardware performance to user eliminates.

Many cloud service providers are offering various kind of services with their platform. Among them, Amazon as Amazon Web Services provides EMR (Elastic MapReduce), Google as Google Cloud Platform provides DataPro and Microsoft as Microsoft Azure provides Cloud Native as a big data analytics framework. Likewise, others cloud vendors are also providing different services on the infrastructure level, software level, and platform level. Similarly, Cloudera and Hortonworks are two popular software companies which provide Apache Hadoop and Apache Spark software supports and services.

2.2 Clustering

A *computer cluster* is a group of computers connected together as a distributed or parallel computing system to form a single virtual and powerful hardware platform [30]. In other words, the cluster refers to a single logical unit consisting of multiple computing hardware that is linked through Local Area Network (LAN). Clustering mechanism leverages much faster processing speed, large storage capacity, better data integrity, superior reliability and wider availability of resources [7]. Another key thing to remember with computer cluster technology is that it enables high availability, load balancing, and parallel processing. The way of current cluster computing is changing with the advent of commodity high-performance processor, low latency/ high-bandwidth networks and software infrastructure and development tools in order to facilitate the use of cluster [2]. Not only high availability, parallel processing and load balancing with cluster computing but also flexibility with scale-up and scale-down of nodes in the cluster is another important feature of cluster computing. Moreover, adding the node in the cluster in order to supply sufficient storage and boost up data processing speed for appropriate corresponding throughput is to scale-up its flexibility. However, reducing the number of node in the cluster limits unused resources unit in the cluster. Hadoop cluster is specifically designed for storing and processing the massive amount of data paralleled in a distributed computing environment. All the machines in the cluster run Hadoop's open source distributed processing software on a low-cost commodity computers [17]. Components of Hadoop cluster are further described in this thesis from section **2.3** to its subsection **2.3.3**.

2.3 Hadoop

Fundamentally, Hadoop is an open source infrastructure level software for storing and processing large datasets in the cluster. Basically, the massive amount of data cannot be stored and processed by single node hardware. Whats more is, these large datasets required the systematic and scientific way for both storing and processing. Hadoop as a software under Apache license comes up with master-slave architecture to omit storing and processing of big dataset problems. There are two points in support of this view. First, Hadoop provides data storage service in an organized distributed way with its Hadoop Distributed File System (HDFS) which separately stores actual data and its corresponding metadata in its different components. Not only that, Hadoop also replicates the actual data across multiple machines in the cluster. Second, Hadoop provides MapReduce framework to process data. Unlike, processing data in a central node, MapReduce framework processes data in every single machine where data stored. Finally the result after processing are sent to the master node. Another key point to remember with such framework mechanism is high throughput access to the datasets that guarantees the reliability of database [36]. The components of Hadoop basis is further described in next section. The two main subsystem of Hadoop basis is HDFS and YARN (Yet Another Resource Negotiator).

2.3.1 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File system is the design to store and process very large files across machines in the cluster. HDFS has many similarities with other existing distributed file system like GFS (Google File System) and S3 (Amazon Simple Storage System). However, the differences are significant form other Distributed File System (DFS). Basic features of HDFS are:

- Highly fault tolerant
- High throughput
- Suitable for application with large datasets
- Streaming access to file system data
- Can built out of commodity hardware

HDFS fault tolerance mechanism by which system works properly even if some component in the system is non-functional. Thousands of server machines storing some part of file's system data has the non-trivial probability of failure. For this reason, detecting the failure and take an action for quick automatic recovery is the main goal of HDFS. In the same way, HDFS is a user-space file system, so there is single central node called NameNode which contains all in-memory directory of where all the blocks and their replicas are stored across the cluster. In order to read those files application code ask to NameNode for a list of block and then starts reading sequentially. Meanwhile, data is streamed off the hard-drive by maintaining the maximum I/O rate that drives can sustain [33]. Of course, streaming access to the file system data is another spectacular feature of HDFS which implies constant bit-rate above the certain threshold when transferring data. HDFS concerns for high throughput as well. Applications write once and ready many access is the design model of HDFS to leverage throughput. HDFS divide a big job into different blocks and processing is done in parallel and independent to each other. Due to this, amount of work done in unit time is high. Thus, high throughput is achieved. Correspondingly, HDFS is suitable for application with

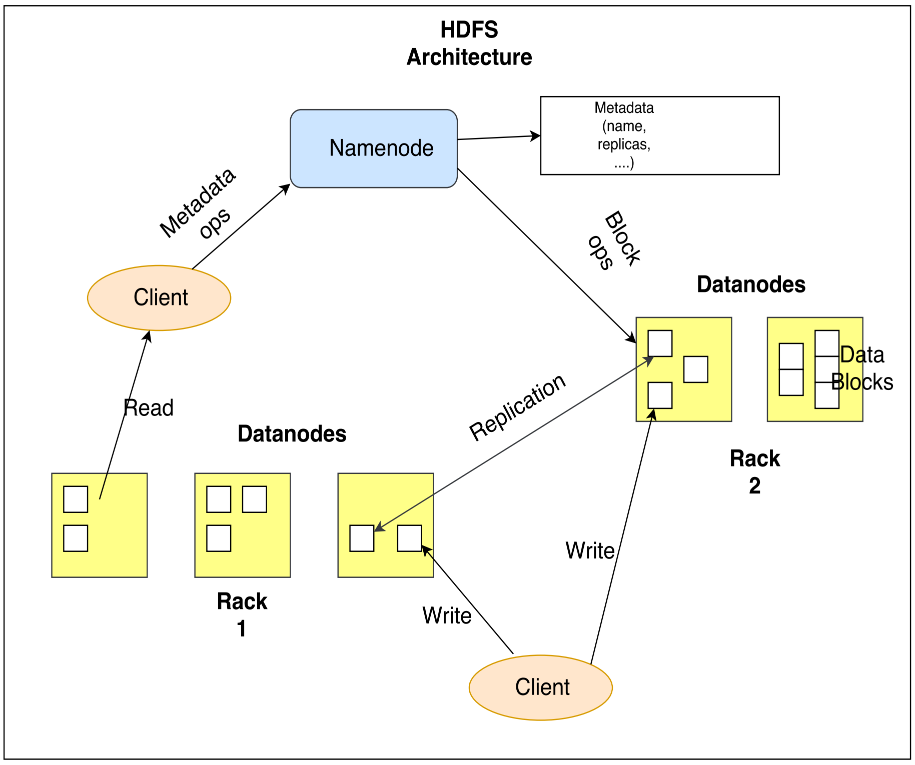


Figure 2.1: High level architectural design of HDFS.

large datasets. A typical file in HDFS is gigabyte to terabytes in size, so millions of files can be supported by the single instance in the cluster.

2.3.1.1 HDFS NameNode and DataNode

The NameNode and DataNode are pieces of software designed to run on commodity machine [4]. HDFS is built using java language; any machine can run NameNode or DataNode if they support Java. As mentioned earlier, HDFS has a master/slave architecture NameNode as master and DataNode as slaves. Furthermore, HDFS cluster consists of single NameNode and multiple DataNode, usually one per every machine in the cluster. NameNode manages the file system namespace operation like opening, closing, renaming files and directories and also regulates access to the files by file system clients. On the other hand, DataNodes are responsible for managing HDFS data storage in which they execute read and write operation as soon as the request from the file system client perceived.

2.3.1.2 HDFS Data Replication

HDFS is designed in a way that can reliably store large data files as a sequence of blocks by breaking the large file into many data blocks. In general, all the blocks size are equal by default except the last one. As a result, HDFS replicates these data blocks in the cluster as it is loaded. Replicating data is an integral part of what makes the overall system effective at all. Replication of data block does not only provide fault tolerance but also helps running the map tasks close to the data which avoid putting extra load on the network. Moreover, block size and replication factor are configurable. NameNode makes all the decisions, where it periodically receive message and block reports from DataNodes in the cluster [4].

2.3.1.3 HDFS Commands

All the HDFS commands are invoked by the *bin/hdfs* script [3]. Running the HDFS script without any arguments prints the description for all commands. Table **2.1** presents some of the important HDFS commands used in this thesis.

Table 2.1: HDFS Commands

Commands	**Actions**
hadoop fs -cat	To list existing directory in hdfs
hadoop fs -mkdir	To create directory
hadoop fs -chmod	Change the permissions of files
hadoop fs -copyFromLocal	To copy file form local machine
hadoop fs -copyToLocal	To copy file from hdfs to local machine
hadoop dfs -df	Displays free space of hdfs
hadoop fs -dus	Displays summary of file length
hadoop fs -mv	Move file form source to destination
hadoop fs -rm	To delete files
hadoop fs -rmdir	Recursive version of delete

2.3.2 Yet Another Resource Negotiator (YARN)

Hadoop has introduced new component YARN (Yet Another Resource Negotiator) in 2012. Thus, the inability of previous version of Hadoop for resource sharing among multiple computational frameworks is no longer in existence, since Hadoop with YARN component released. YARN is a cluster level computing resource manager responsible for resource allocations and overall job orchestration [36]. It provides a central platform to deliver consistent operations, security and many others useful tools across Hadoop cluster. YARN consist of two main components: **ResourceManager** (*RM*) and **NodeManager** (*NM*). RM is the global component one per cluster where NMs presents per node in the cluster. First component act as a global computing resource arbiter, likewise the second component is responsible for managing node-level resources and reporting their usage to RM. RM in the Hadoop cluster further consist two component: **Scheduler** and **ApplicationManager\ApplicationMaster** (*AM*). The scheduler allocates the resources among running applications. In the same way, AM is responsible for accepting job-submission, negotiating container which executes applications and provides the service for restarting the AM container in the case of failure [23]. *AM* in another word, is designed specifically for applications process which negotiates resources from *RM* and collaborates with *NMs* in order to execute and monitor its individual task. Similarly, scheduling of resources for application is based on the requirement which is realized using an abstract notion of containers.

[3]https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html

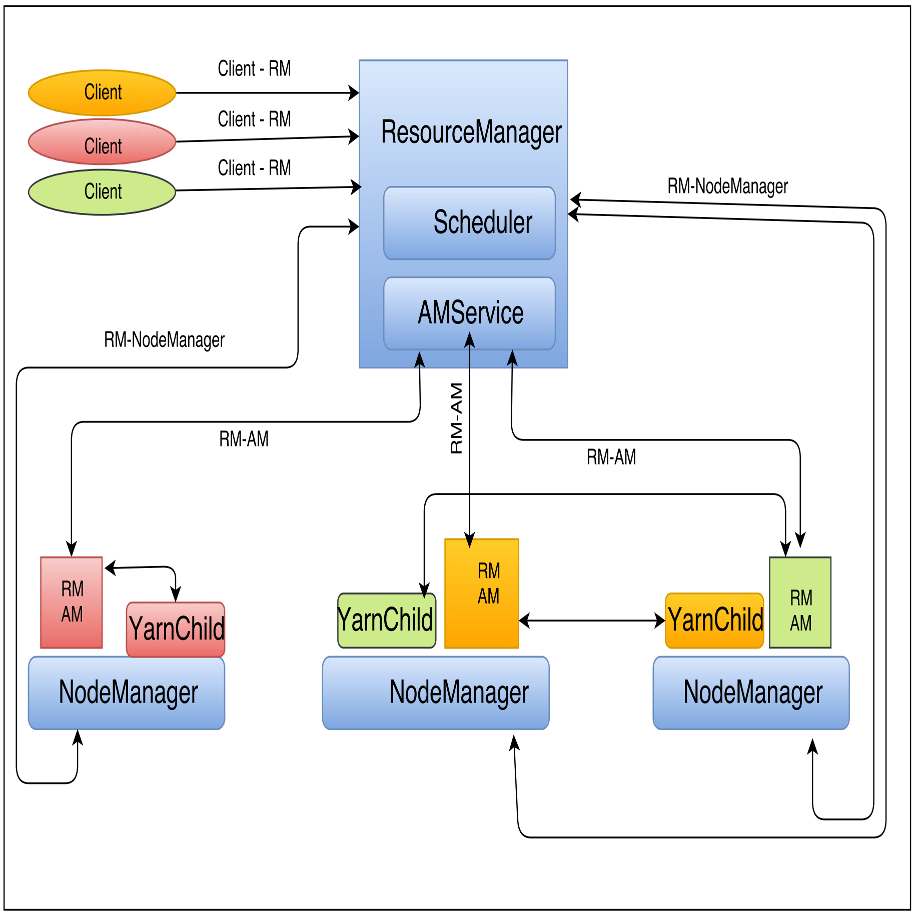


Figure 2.2: Yarn Architecture (Blue color represents system components and in yellow, pink and green three applications running.)

2.3.2.1 ApplicationMaster (AM)

Application in Hadoop can be the static set of the processes of a logical description of work. In order to execute Hadoop job in the cluster AM as a process co-ordinates with applications. AM itself run in a cluster container ApplicationMaster which is same as other jobs executed in a YarnChild container. The Figure, **2.3 on page 12** presents more clear view regarding the concept of containers allocation. Each application to be executed in a Hadoop cluster has its own dedicated ApplicationMaster container. The AM running as application frequently sends message to ResourceManager notifying its status and the state of the application's additional resources need [35]. In the same fashion, AM can update its plan to execution job. Based on the resources available ResourceManager assigns resources to AM in a specific cluster node. On the contrary, container status will not be interpreted by ResourceManager because AM itself running as ApplicationMaster container in a unreliable hardware. Therefore, container running might be the failure. To put it in another way, NodeManager does all kind of reporting task to the ResourceManager in the cluster.

2.3.2.2 Containers

Container simply is the mixed form of resources like memory and CPU where the unit of work occurs. To say, a set of resources are defined for containers in Hadoop Yarn. In Hadoop cluster there are two types of containers: *MRAppMaster* container and *YarnChild* containers. Each node in the Hadoop cluster can have multiple containers. For instance, while the MapReduce job is submitted then ResourceManager first deploy MRAppMaster container in order to execute its specific ApplicationMaster process. Now, MRAppMaster will ask for more resources with ResourceManager in order to run MapReduce task. Moreover, ResourceManager allocates an address to run the container on the basis of available resources in the cluster. Finally, MRAppMaster contact to the NodeManager based on the address it gets form the ResourceManager to run its container. This container which runs actual MapReduce task is YarnChild container.

2.3.2.3 Maximum Percent of Resources in the Cluster (MARP)

MARP is the value in percent of resources in the cluster to run ApplicationMaster container. Additionally, MARP value is directly proportional to the number of application (job) execution. More value for MARP provides more resources to ApplicationMaster in order to run ApplicationMaster container for application daemon. The more ApplicationMaster container in the cluster means many application get chance to run MapReduce operation in parallel. So, in another word, MARP parameter value is a kind of limits which define the maximum amount of resources in the cluster to run ApplicationMaster. MARP also seems to be responsible for controlling the number of concurrently active applications in the cluster. This value is configurable and can be configured with the property named *yarn.scheduler.capacity.maximum-am-resource-percent*. By default, MARP value is set for 10%.

Table 2.2: Yarn Command

commands	**Actions**
yarn application -list	Lists applications from the RM
yarn application -status applicationid	Prints the status of the application.
yarn node -list	Lists all running nodes
yarn application -status Nodeid	Prints the status report of the node
yarn resourcemanager	Start the ResourceManager
yarn nodemanager	Start the NodeManager
yarn proxyserver	Start the web proxy server
yarn rmadmin -refreshQueues	ResourceManager reloads the mapred-queues configuration file

The figure, **2.3 on the following page** shows two cases of MARP parameter value set in the cluster. Due to the reason, small MARP parameter value set in the cluster, comparatively small number of resources get utilized by ApplicationMastaer with Nodemanager 1. Therefore, rest of the resources can be utilized by actual job execution container Yarn Child. In the same figure by Nodemanager 2, one can notice that ApplicationMaster utilized majority number of the resources unit because of big MARP parameter value set in the cluster. Thus, small number of resources is available for actual job execution container Yarn Child.

Configuring the appropriate optimum value of MARP during Hadoop job execution is a challenging task. However, providing greater value to MARP leverage by small job as job execution time gets reduced but at the same time unused resources unit in the cluster increases. Thus, significantly degrades the performance of the cluster. Alternatively, the small value of MARP directly affects big jobs with more time consumption. Again, resources unit in the cluster seems to be wasted as only few resources get consumed which leads to increase in overall time. For these reason, configuring an appropriate optimum value of MARP plays the vital role in order to achieve maximum advantages of resource unit in the cluster.

2.3.2.4 Yarn Commands

Yarn provides a rich set of command-line commands in order to help with various aspects of yarn packages including installation, administration, publishing etc. Yarn commands are invoked by the bin/yarn script [4]. Running the yarn script with any arguments prints the description for all commands. Table **2.2** presents the Yarn commands used in this thesis.

2.3.3 MapReduce Paradigm

Hadoop employs a MapReduce execution mechanism in order to implement its fault-tolerant distributed data processing system over the large datasets in the

[4]https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/YarnCommands.html

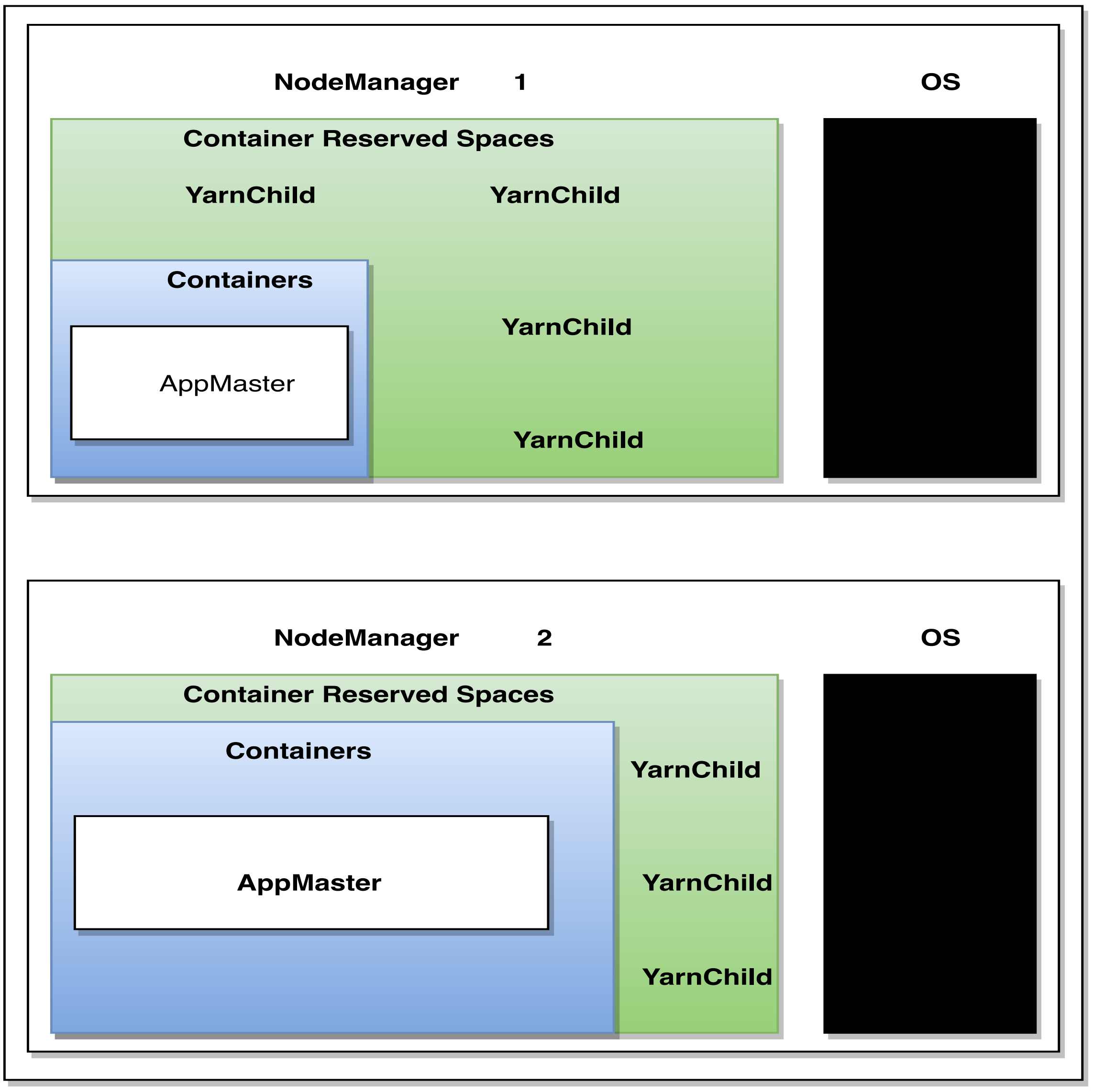


Figure 2.3: Resources consumption by ApplicationMaster.

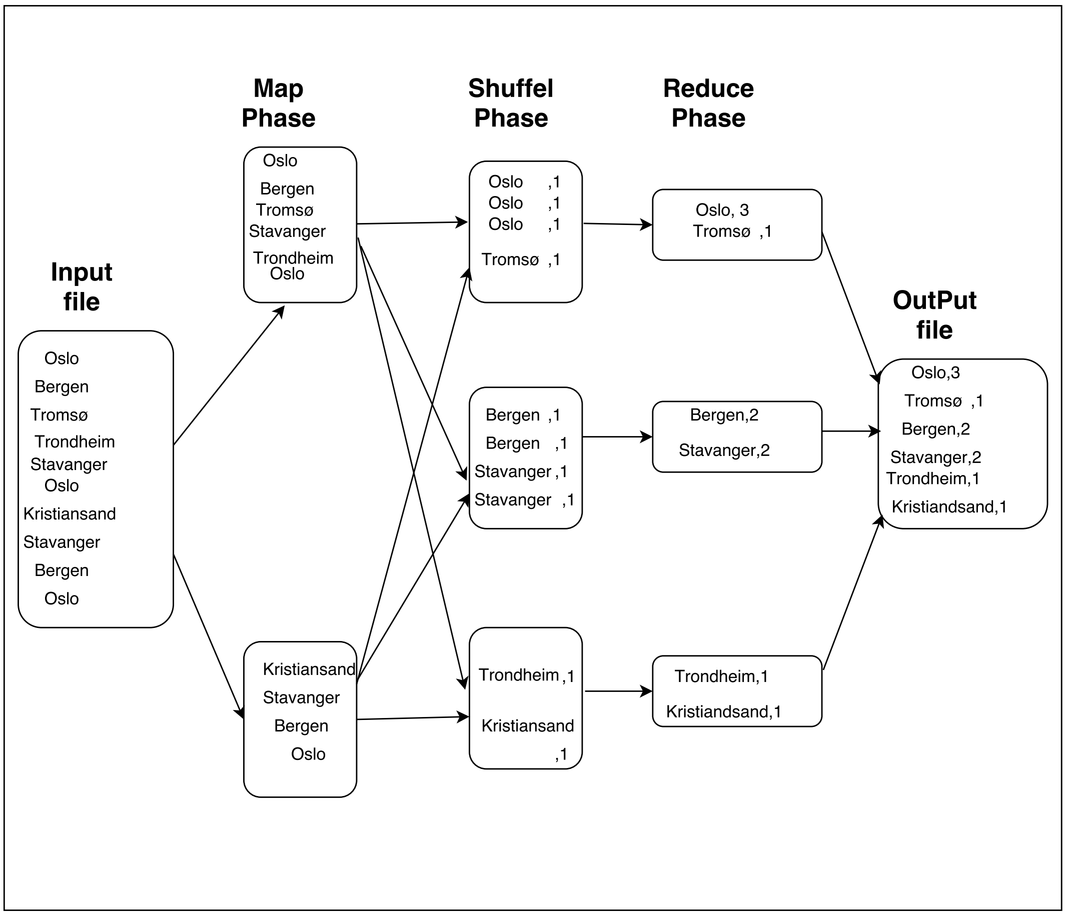


Figure 2.4: MapReduce framework Architecture.

cluster [31]. MapReduce engine is an effective solution for parallel program development and massive data processing in a distributed environment [8, 9]. The main point with such MapReduce framework lies on its two functions: Map function and Reduce function. Both the functions with MapReduce framework works parallel, operating on sets of a key-value pair (k, v) [31]. Initially, Map function gets executed over input datasets in the cluster which gives zero or more key-value pair (k, v). Then, the framework calls and implement Reduce function on those key-value pairs grouped form Map function. Based on the key, all the collected records are now ready to get transferred to the node which runs Reduce function. In this point, data transfer between nodes in the cluster takes place. Finally, as a final output key-value pair (k' v') get produced. Furthermore, despite user-defined Map and Reduce function MapReduce framework itself monitor parallelization and fail-over in the system while running Map and Reduce function [31]. Figure **2.4** shows different working phases during job execution by the MapReduce framework.

2.4 Adaptive Variable Learning (AVLR) Algorithm

Adaptive variable learning is a kind of algorithm which dynamically tunes the learning rate in accordance with the change in the specific variable value [27]. The figure, **2.5 on the next page** provides more clear view regarding the algorithm working mechanism. Furthermore, the same state continues as a reward until and unless the favorable condition detected. However, state transition takes place as the penalty if unfavorable condition is detected. The author of this thesis also aims to use the same logic of the state change between defined actions in order to allocate cluster resources unit to the MapReduce jobs.

2.5 Google Cloud Platform (GCP)

Google Cloud Platform is a suite of cloud computing services runs in the same infrastructure that Google uses internally for its end-user products, such as Google Search and YouTube [5]. Moreover, GCP consist of the set of management tools and physical hardware, such as computer hard disk drives, CPUs, RAMs. Not only that, GCP also provide series of modular cloud services including compute, data storage, data analytics and Machine learning in order to make GCP more handy to their end users [15]. This thesis has used Google Cloud Platform (GCP) as one of the vital components for creating the cluster and running the Hadoop benchmarks in order to render appropriate results.

2.5.1 Google Compute Engine (GCE)

Google Compute Engine in a Google Cloud Platform (GCP) is pay-per-usage service with a single second minimum [16]. GCE platform with features such as scale, performance, and value in order to create and run the large cluster of virtual machines (VMs) on Google infrastructure. To put it in another way, GCE does not contain any upfront investment and can run thousands of virtual CPUs on a system that has to be designed to be fast and to offer strong consistency of performance [14]. VMs are available in a number of CPU and RAM configuration and with Linux distribution, including Debian, CentOS, and Ubuntu. Furthermore, customers may use their own system images as well. Mainly three points of the GCP attracts the attention of author of this thesis.

- **High-Performance, Scalable Virtual Machines (VM)**

Google Compute Engine delivers large scale of virtual machines running in its innovative data centers and with high performances because of its own worldwide fiber network. Thus, Compute Engine VMs boot quickly and deliver consistency performance.

- **Low Cost, Automatic Discount**

[5]https://en.wikipedia.org/wiki/Google$_{C}loud_{P}latform$

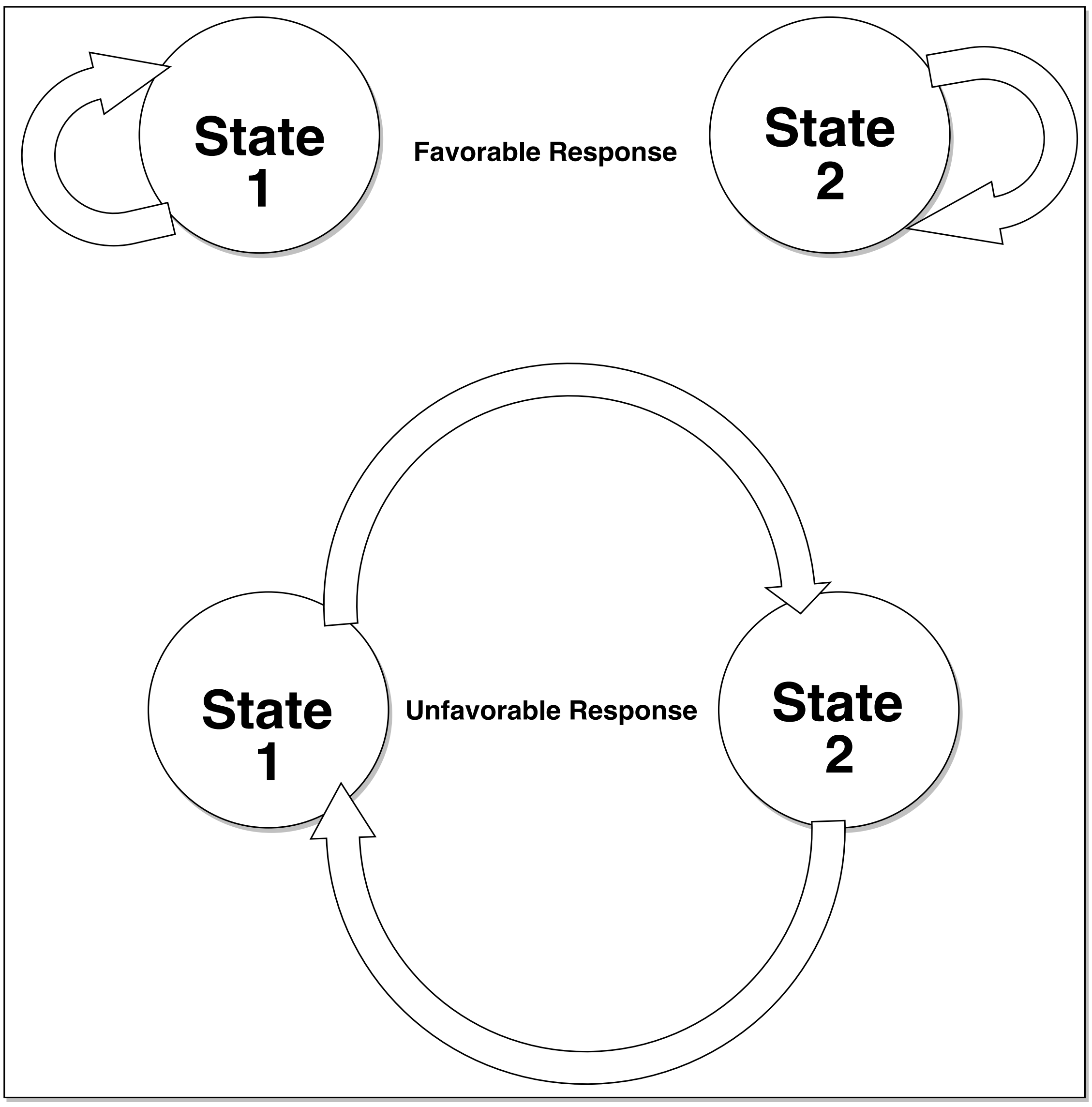


Figure 2.5: The state transation graph.

Google bills in second-level increments, so user can only pay for compute time of VMs user used for. Surprisingly, Google automatically provides discounted prices for long running workloads with no up-front commitment required [12].

- **Environmentally Friendly Global Network**

The most spectacular side of Google Cloud Platform is that their infrastructure are entirely carbon-neutral. According to Google, their global network of data centers consumes 50% less energy than typical data centers and they purchase enough renewable energy to match 100% of the energy consumed by their global operation [13].

2.5.2 Hadoop Performance Tuning

Hadoop is Java-based distributed computing framework which is designed to support applications via the MapReduce programming model. In general workload, dependent Hadoop performance optimization efforts have to focus on three major categories: the system hardware, the system software, and the configuration and optimization of the Hadoop components [18]. This thesis aims to have a deep dive into fine-tuning of the Hadoop components which can make the optimum utilization of the physical resources of the cluster.

2.5.3 Related Works

Authors in paper [25] proposed an online MRONLINE model that tunes the parameter in order to improve the performance of MapReduce job. MRONLINE monitors job execution and tunes parameter based on collected statistics and provides fine-grained control over parameter configuration. The main goal of MRONLINE is to provide the different configuration for every single task instead of using the same configuration for all jobs. Furthermore, the authors designed a gray-box based smart hill climbing algorithm that can effectively converge to a near-optimal configuration with high probability. Thus, authors claim 30% improvement on performance compared to default configuration with MRONLINE model.

JellyFish in research work [8] is an online performance tuning system that improved the performance of MapReduce job by increasing resource utilization in Hadoop YARN. JellyFish collects real-time statistics to optimize configuration and does resource allocation dynamically during the job execution. During the time of performance tuning, first JellyFish tunes configuration parameter by reducing the dimensionality of search space with a divide-and-conquer approach. In fact, model-based hill climbing algorithm is designed and developed to improve tuning efficiently. Secondly, JellyFish reschedules resources in nodes using an elastic container with expand and shrink dynamically based on resources usages. Authors of this research work claim that developed model can improve the performance by 24% compared to default YARN configuration.

In the paper [34] also addresses self-tuning system based on application profiling and performance. Especially, two distinct phases were designed as Analyzer and Recognizer. Analyzer phase trains the developed model with machine learning techniques in order to form a set of equivalence classes of MapReduce applications for which the most suitable Hadoop configuration parameter that maximally improves performance for that class are identified. Moreover, this paper includes modifications to K - $mean$ ++ algorithm in Analyzer phase as a key research contribution. On the other hand, Recognizer phase classifies unknown incoming job to one of the equivalence classes. Thus, Hadoop configuration parameter can be self-tuned.

Part I

The project

Chapter 3

Approach

This chapter is all about the methodologies, processes and steps taken in order to address the defined problem statement of this thesis *-How the number of parallel running MapReduce job and associated throughput can increased by utilizing appropriate optimum resources unit in the Hadoop cluster ?*

3.1 Objective

The main objective of this project is to run the possible maximum number of concurrent jobs with their associated throughput by having appropriate resources utilization accessible from the Hadoop cluster. Another goal of this project is to overcome the limitations of static configuration over Hadoop MapReduce framework. Thus, this paper proposed an approach named *Adaptive Parameter Tuning Of Hadoop (APTH)* which tune one of the system level parameters at the run-time of MapReduce operation. The proposed approach APTH will consists of tuning algorithm. The initial part of the algorithm is responsible for calculating the current progress of the running MapReduce jobs and the second part of the algorithm takes an action for allocating resources unit by tuning the parameter based on the current progress value. In this way, APTH approach keeps the balance between the concurrent running number of jobs and their corresponding throughput. This leads to reduction on ideal resources in the Hadoop cluster which significantly reduces the overall job completion time compared to default Hadoop configuration.

3.1.1 Loss of Input Job (LOIJ) and Loss of Task Throughput (LOTT)

To increase in MapReduce job parallelism or in order to increase the MapReduce task throughput of jobs *MARP*(**see Section 2.3.2.3**) parameter plays a significant role. MARP parameter value is to increase or decrease the resources unit for *ApplicatonMaster*(**see Section 2.3.2.1**). Let's say total configured resources (TR) of the cluster get utilized by two components, ApplicationMaster (AMR) and YarnChild (YCR) i.e., TR = AMR + YCR, (assumpton is that configured resources in the cluster gets 100% utilized). But, the most of the time resources configured for Hadoop MapReduce job may not be 100% utilized, so there might be some

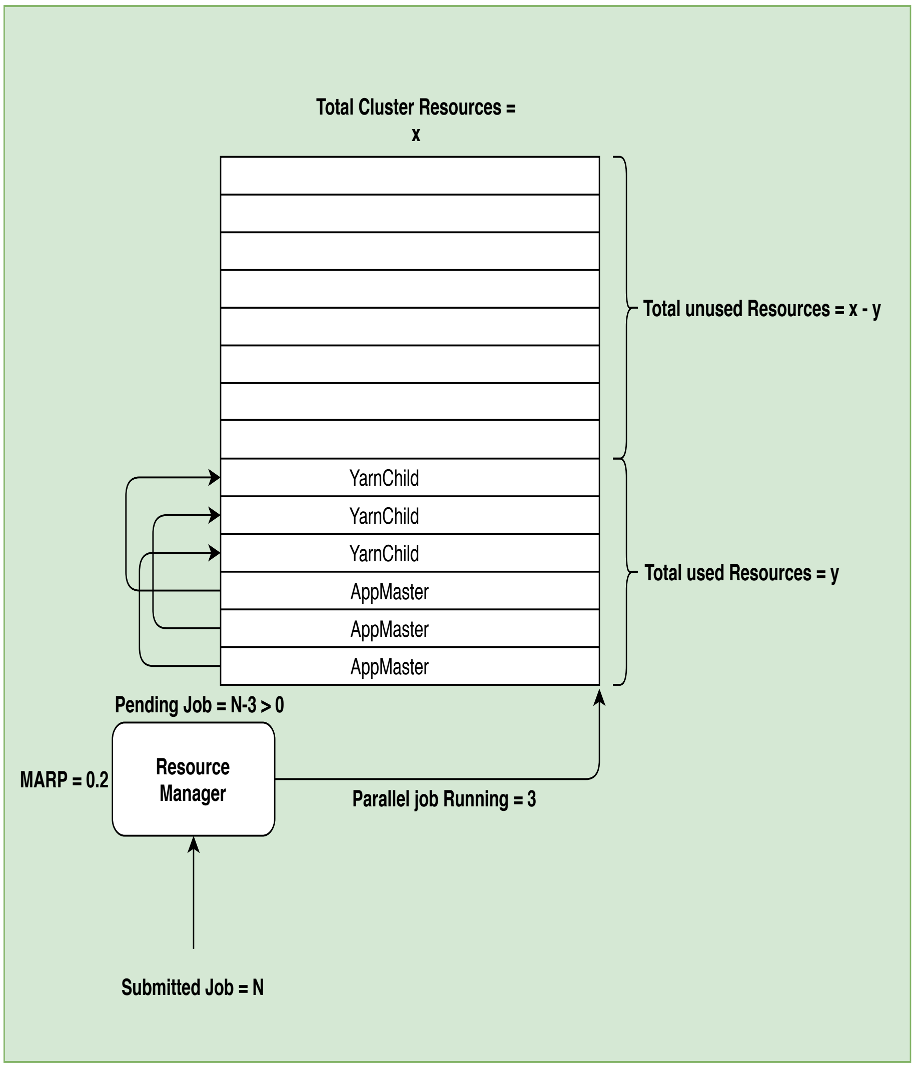


Figure 3.1: Loss of input job with small MARP value and increase in ideal resources.

ideal resources (IR) in the cluster. At this moment, TR = AMR + YCR + IR .

There are some situations like how ideal resources can make space in the cluster during runtime? The first reason is simple because of the small size of MapReduce job or because of less number of input MapReduce job. The second reason is that, because of a small MARP value which allocates fewer resources for ApplicationMaster. The figure, **3.1** provides more clear views for how small amount of resources allocation with small MARP value helps on increasing ideal resources in the cluster during MapReduce runtime. In the figure, **3.1** MARP parameter is set to be 0.20, means 20% of configured cluster resources can be used by ApplicationMaster.

Among submitted N numbers of MapReduce jobs, only 3 jobs get a chance to get executed at a time. Still, N-3 > 0, MapReduce jobs are pending, which means that jobs are not yet scheduled for execution as there is lack of allocated resources for ApplicationMaster. However, total unused resources = x - y in the cluster is full of wastage during the MapReduce job runtime. The main key point to remember here is a pending number of jobs waiting for resources and the wastage of significant resources in the cluster. Moreover, this types of static or default configuration consume more time in order to process big data and of course, user achieves less in parallel execution of jobs due to poor resource utilization. Discussed problem with ideal resources can be called as *Loss of Input Job (LOIJ)* because the additionally greater number of MapReduce jobs has the possibility for execution if correct MARP parameter is to be configured. Significantly, increase in MARP parameter that allocates more additional resources for ApplicationMaster is the solution for LOIJ.

Another key point is that MARP is a job level controller that allocates resources to run ApplicationMaster as a job daemon in a container and has no involvement in actual job execution. Therefore, only YarnChild(**see Section 2.3.2.2**) container is responsible executing the actual job. As many as YarnChild container executes, gives better throughput gained. As mentioned earlier, with increment in MARP parameter provides more additional resources for ApplicationMaster, so every time increment in MARP parameter means decrement in the number of resources for actual job execution. Therefore, with high MARP parameter configuration, there is less space for actual job execution. This situation leads to reduction in throughput problem. In addition, this problem can be called as *Loss of Task Throughput (LOTT).*

Figure, **3.2 on the following page** shows the situation, how throughput gets lost during MapReduce job execution. In the figure, **3.2 on the next page** MARP value is set to be 0.80, which means 80% resources form the configured total resources can be used by ApplicationMaster. Having said that in another way, 20% resources are allocated for actual job execution. If N is the total number of jobs submitted and 11 are running in parallel, still N-11 > 0 are in pending. However, those 11 jobs running in parallel are competing for resources released by YarnChild because only a few containers run the actual job. As more resources allocated for ApplicationMaster, actual job execution takes place in few YarnChild containers which leads less throughput and more time consumption. Even though here with LOTT, 100% resources are utilized but not in an appropriate way that balances the number of parallel running job and their corresponding throughput. Therefore, this kind of poor resource utilization can degrade the performance of Hadoop cluster.

In order to overcome the discussed LOTT and to achieved eventual better throughput MARP parameter again plays a vital role here. Decrement on MARP parameter limits additional job to be scheduled for execution. For instance, in the figure **3.2 on the following page**, by the time while job is running, if it is supposed to be MARP parameter value 0.50, then with the next interval of time, less number of ApplicationMaster as compared to parameter value of MARP 0.80. This process limits the number of jobs that are pending and supposed to get executed very soon with previous MARP configuration. Hence, now the resources for ApplicationMaster and YarnChild is supposed to be equal that is 50% each.

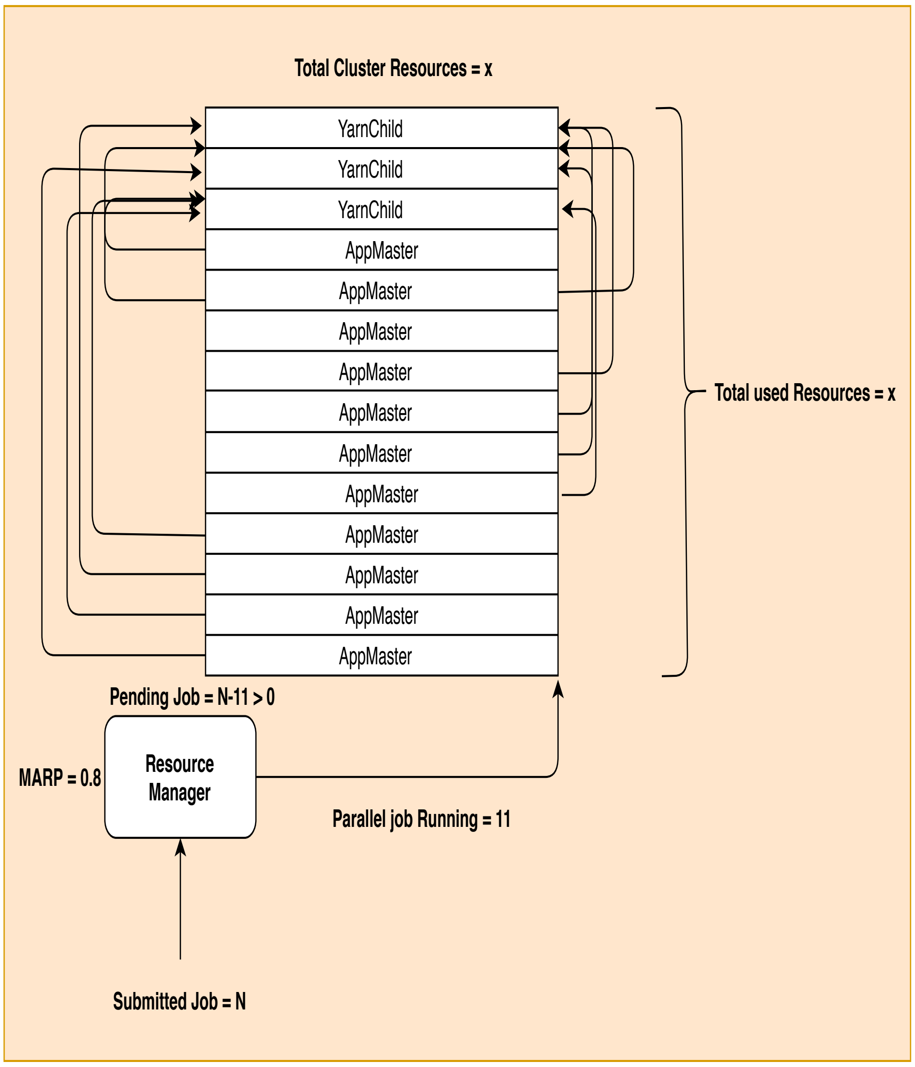


Figure 3.2: Loss of Task Throughput with high MARP value and inappropriate way of full resource utilization.

Let's assume *AMmax* and *AMmin* to be the maximum and minimum number of **ApplicationMaster** respectively running jobs while *YCmax* and *YCmin* as a respective maximum and a minimum number of running **YarnChild**. If configured total resources used is 100% then

Job parallelism is increased but less throughput is achieved with following equation

- **AMmax + YCmin = TR**

Task throughput is increased but less job parallelism is achieved with following equation

- **AMmin + YCmax = TR**

This thesis aims to solve both of these problems occurred by inappropriate MARP parameter setting. Hence, one of the better ways is to get a solution that is to tune the parameter and reconfigure it to the whole cluster at the job run-time. Static tuning of parameter sounds difficult, time-consuming and inefficient. However, in this thesis, the author aims to design a dynamic approach that tunes MARP parameter based on feedback from the cluster. Moreover, this self-adaptive approach for balancing job parallelism and their corresponding task throughput during job run-time will be implemented by designing an optimum algorithm and developing it in the script as an automation tool. Detail technical description of this algorithm can be found in **Section 4**. The project work has been structured into five major phase, these are:

- **Hadoop Cluster Design Phase**
- **The Algorithm Design Phase**
- **Implementation Stage**
- **Measurement, Analysis and Comparison Stage**

3.1.2 Hadoop Cluster Design Phase

All the experiments carried out in this project by creating Hadoop cluster in Google cloud. Since appropriate optimum cluster resources utilization is one of the main goals of this project, attention is on the constitution of node servers in terms of their CPU, memory, disk, and operating system. Moreover, comparatively big size of CPU and memory has been chosen in order to eliminate the possible problem of CPU and memory overhead for master nodes in the cluster. On the other hand, all nodes except master node in the cluster has the same size of CPU and memory. Finally, 3 nodes with the same operating system where 1 of it is master node and rest are worker nodes has been designed and developed for this project.

As mentioned earlier, hardware for the cluster has been sourced using compute engine form Google cloud as the flavor shown by the table **3.1 on the next page**.

Table 3.1: Compute Engine Flavor

SN	NODE NAME	ZONE	TYPE	CPU	MEMORY
1	project-master-01	europe-west-4-b	n1-standard-4	4vCPUs	15GB
2	Project-worker-01	europe-west-4-b	n1-standard-2	2vCPUs	7.5GB
3	project-worker-02	europe-west-4-b	n1-standard-2	2vCPUs	7.5GB

3.1.3 The Algorithm Design Phase

One of the main part of this project is to design and develop an optimum algorithm. The algorithm should be designed in a way that can address the problem as stated in the **Section 1.1** and simplified version of two problems described in earlier **Section 3.1.1**. Step by step operation that can leverage maximum resources form the designed and developed Hadoop cluster is the main goal of this algorithm. Moreover, the algorithm designed is focused on progress value of each running job over time where progress value of each running MapReduce job can be fetched by using Hadoop Yarn command. Finally, based on those progress rates algorithm is able to change the number of resources allocated to the MapReduce jobs in the cluster at run-time so the new job could be added or stopped being added to the cluster.

3.1.4 Implementation Stage

Implementation stage is where all the design gets actual shape in order to achieve the goal of the project. In this thesis, the implementation stage is one of the important stage where the designed Hadoop cluster is developed. Not only cluster but also the algorithm design is developed into the bash script as an automation tool. Moreover, specific functions for the particular action are developed in the script.

The project itself is a combination of various complex sub-task with many challenging responsibilities to be taken into account. Fortunately, various available tools and components help to assist in accomplishing tasks and executing responsibilities. Similarly, in order to achieve optimum solution for the problem explained in problem statement (**see Section 1.1**), many tools and their related components has to be recognized, designed and developed as the part of a system. Following are the tools required to develop this project for the proposed approach.

3.1.4.1 Cluster

Clusters are usually deployed in order to improve performance and availability over that of the single computer. Moreover, this project is all about big data processing with the help of computing nodes in the cluster. Therefore, to implement project design and to evaluate the results 5 node Hadoop cluster will be created in Google Cloud Platform using Google Compute Engine.

3.1.4.2 Software Development Kit (SDK)

The cloud SDK is the set of tools such as gcloud, gsutil etc., for accessing Google Compute Engine (GCE), Google Cloud Storage, Google Big Query and other products and services from the command line. Furthermore, these tools can be run interactively or can be included as a part of automated script [6]. For the purpose of this thesis, gcloud tools has been frequently used in order to manage authentication, local configuration, developer workflow, and interactions with the Cloud Platform APIs.

3.1.4.3 Java Development Kit (JDK)

The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. JDK includes the Java Runtime Environment (JRE), and Interpreter/Loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development [20]. For this thesis, it is also required JDK to install over the Hadoop cluster as Hadoop is written in Java. Moreover, the author of this thesis has used OpenJDK 7.

3.1.4.4 MapReduce Framework

Hadoop MapReduce is a software framework for easily writing applications which process the huge amount of data in parallel on the large cluster in a reliable fault-tolerant manner. In this project MapReduce framework is one of the important tools which splits the input data-set into independent chunks and are processed in parallel. Thus, author of this thesis has chosen Hadoop-2.8.1 version MapReduce framework to work with.

3.1.4.5 REST API's

The Hadoop YARN web service Rest API's are a set of URI resources that provide access to the cluster, nodes, applications and application's historical information. In order to extract real-time system data related to resource unit and applications, this project has used REST API's.

3.1.4.6 Script Development and Automation

Bash script has been developed as an automation tool in order to check the real-time data access form REST API's. Moreover, feedback control loop has been implemented in the cluster through the script. Thus, the script is responsible for the decision whether to increment or decrement MARP value.

3.1.4.7 R-Programming

Results obtained has been visualized using R-Programming for analysis and easy understanding. LibreOffice Calc has been used for data storing during project experiment phase.

3.1.4.8 Benchmarking Tools

Benchmarks are important tools to evaluates the system, as long as their results are transparent, reproducible and they are conducted due diligence [19]. Therefore, it is essential to quantitatively evaluate and characterize the Hadoop framework through the extensive benchmark. This thesis has used one benchmarking tools to evaluate the performance and to understand the tradeoff between job parallelism and corresponding throughput. HiBench is the benchmarking tool that has been used in this project.

3.1.5 Measurement, Analysis and Comparison

Experiments has been performed in an environment where Hadoop cluster is created with the Ubuntu Linux virtual machine. At least one machine with normal cluster configuration acted as the master machine of the cluster and rest of the machines as workers nodes. Experimentation has involved the processing of a set of workloads. Moreover, HiBench benchmark tool has been used to supply the workloads to the Hadoop cluster. HiBench is a benchmark suite which consists of a set of a set of Hadoop programs that helps to evaluate the Hadoop framework in terms of speed, throughput, HDFS bandwidth, system resources utilization and data access patterns [1]. HiBench benchmark suite was originally developed by Intel to stress and test Hadoop system.

HiBench contains 10 different workloads dividing into 4 categories.

- **Micro Benchmarks (Sort, WordCount, TeraSort, Enhanced DFSIO)**
- **Web Search (Nutch Indexing, PageRank)**
- **Machine Learning (Bayesian Classification, K-means Clustering)**
- **Analytical Queries (Hive Join, Hive Aggregation)**

HiBench benchmark suite has been downloaded and configured in order to run workloads in this project. Moreover, slight modification has been carried out with its configuration files and shell scripts provided in the benchmark suite so as to make the HiBench benchmark adjustable in the developed Hadoop cluster. The key point to remember regarding the workloads is that each workload in HiBench has its own specific parameters. Furthermore, after each workload finished MapReduce execution HiBench benchmark suite generate the report with time in seconds and throughputs in bytes per second.

3.1.5.1 Benchmarking Methodology

This section is all about benchmarking methodology which is defined and used throughout the experiments. In addition, to achieve job parallelism as the main goal of the project, it is supposed that HiBench benchmark suite can supply enough

[1]www.ibm.com

numbers of jobs to be run in the Hadoop cluster. However, as described earlier Hi-Bench benchmark suite only can run a maximum of 10 number of different jobs at a time which is not good enough number of jobs to test and investigate along with this project.

As HiBench benchmark has its own limitation with input number of different jobs at a time, enough input workload for this project is designed in a way that can repeatedly take as an input for the defined workload. Especially, the separate shell script has been developed that selects defined workload and size that has to be pushed multiple times into MapReduce framework. Only three types of workloads **Wordcount**, **Sort**, and **Terasort** are selected for this project.

- **WordCount**

This workload counts the occurrence of each word in the input data, which are generated using RandomTextWriter.

- **Sort**

This workload sorts its text input data, which is generated using RandomTextWriter.

- **TeraSort**

TeraSort is a standard benchmark created by Jim Gray. Its input data is generated by Hadoop TeraGen example program.

During the experiment time, 3 separate experiments with input job numbers 15, 30 and 45 has been taken into account. Figure, **6.1 on page 64** gives more clear view about the experiments. Each experiment runs with three different configuration as default configuration, random MARP value, and with APTH approach. Moreover, the algorithm developed as a shell script is responsible to fetch and write all the cluster metrics during MapReduce job runs in one CSV file. Similarly, each experiment gets implemented with designed APTH approach. Here, developed shell script for the designed algorithm as an automation tool parallelly gets executed that takes action based on the progress value. Furthermore, this algorithm also fetches the metrics form the cluster and write into one CSV file.

In order to ensure the accurate performance measurement, each experiment with APTH approach has been run 15 number of times. Thus, the average value has been calculated and considered as a representative result. This average value for time in second is to complete jobs for particular experiment and average throughput in bytes per second get compared with average values of experiment taken with the default MARP.

Finally, this phase is also all about data analysis and performance comparison in order to see if and to what extent this project objectives have been met. CSV files created by the scripts which contain several columns of data with Hadoop cluster metrics in the experiments has been imported to RStudio (a tool used for data visualization). The key variables within those file like total memory used, a maximum number of parallel running jobs, average throughput, the total time to complete experiment, average total resources used during experiments, total CPU used during experiments has been analyzed.

Chapter 4

Design

This chapter discusses the algorithm designed in order to achieve the objectives. The algorithm was formulated to address the goal of design section from chapter 3 and also to develop the tools for the project that will make appropriate resources allocation to the Hadoop cluster.

4.1 ΔProgress Aware Algorithm Overview

The designed algorithm is mainly aware of the progress values of the running MapReduce jobs. Each time the algorithm is executed will fetch the running application id with their corresponding progress values and sums up those values in order to form single accumulated progress value. Moreover, the algorithm also calculates different accumulated progress values with different MARP parameter in a different time. Based on those different accumulated progress values Δprogress value will be calculated. Hence, Δprogress is the difference between the current progress value with previous progress value. Furthermore, the algorithm decides the action to be taken in order to add more jobs or pull out the allocated resources so that the additional job cannot be added in the Hadoop cluster.

For the first time when the algorithm runs will calculate accumulated progress value by simply summing up the associated progress values of respective running applications in the Hadoop cluster. Meanwhile, the situation might change in running number of applications and their corresponding status. Due to the possibility of change in the status of running application, designed algorithm calculates current accumulated progress value each time it is executed except for the first time.

Starting from default MARP parameter value, our proposed algorithm calculates accumulated progress value Progress1. As soon as progress1 is calculated, algorithm makes an increment on MARP parameter by the small change and calculates current accumulated progress Progress2. Similarly, again after having small increment on MARP parameter, the algorithm calculates the current accumulated progress i.e., Progress3. As soon as the algorithm calculates three Progress values i.e., Progress1, Progress2, and Progress3, the algorithm calculate ΔProgress1 and ΔProgress2 (**see Section 4.1.2**).

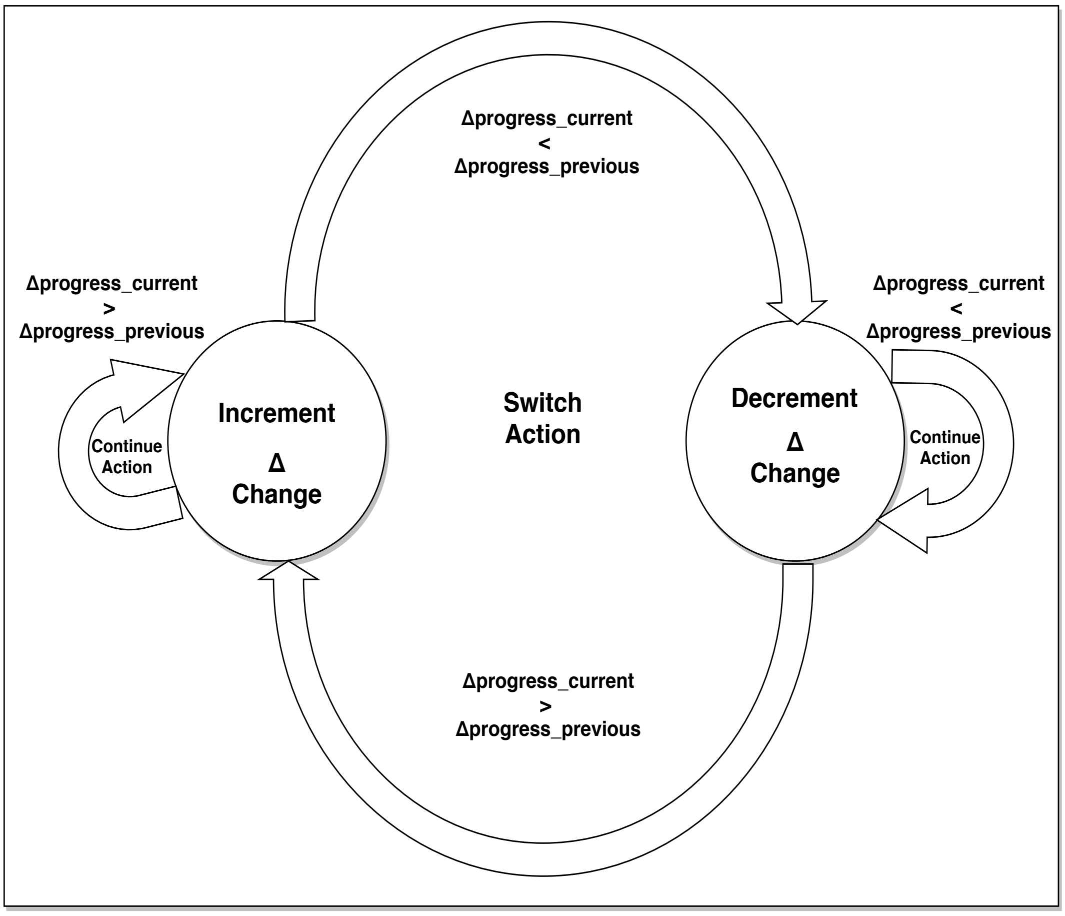


Figure 4.1: Conditions and actions of the algorithm.

- **Δprogress1 = (Progress2) - (Progress1)**
- **Δprogress2 = (Progress3) - (Progress2)**

Along with the calculation of ΔProgress1 and ΔProgress2, the first part of the algorithm is finished and the second part of the algorithm is initiated. This part is action-oriented either to make an increment on MARP parameter in order to check if the additional number of jobs can be executed (to make increment on job parallelism) or decrement on MARP parameter in order to stop numbers of additional jobs to be added in the cluster (to make sure less number of jobs share resource unit in the cluster).

The figure, **4.1** presents the second part of the algorithm to be executed and check if ΔProgress2 (current progress) is greater than ΔProgress1 (previous progress), which means the throughput is in increasing order. Therefore, the algorithm first checks the throughput if it will still continue the increasing order by adding the numbers of the additional jobs to the cluster. If yes then, it makes an increment on MARP parameter. Alternatively, the algorithm also makes a decrement on MARP parameter in order to stop scheduling more number of jobs if ΔProgress2 is smaller than ΔProgress1 which means that the current throughput is not in an increasing order as more number of jobs are sharing the same resource unit. On the other hand, the algorithm also has one more action i.e., to be in ideal state, neither increment nor decrement on MARP parameter. Therefore, this action is defined as 'Ideal Action'. For instance, the condition when ΔProgress1 and ΔProgress2 are equal, then the algorithm does nothing but it only calculate the current accumulated progress and update the Progress3 value.

After each action taken by the algorithm i.e., increment, decrement or ideal, the algorithm is defined to sleep for certain time interval. Figure, **4.2 on the following page** aims to give a clear view regarding how new ΔProgress1 and ΔProgress2 gets calculated. In fact, this newly calculated progress value now becomes Progress3 and previous Progress3 value now becomes new progress value for Progress2. Similarly, the algorithm sets the previous progress2 value as a new value for Progress1. In this way, each time when the second part of the algorithm executes, it calculates new progress values for Progress1, Progress2, and Progress3. As a result, the algorithm always have the chance to calculate new ΔProgress1 and ΔProgress2. As long as the algorithm finds new values Progress3, it keeps on continuing execution of the algorithm.

Finally, the designed algorithm keeps the balance between job parallelism and associated throughput by simply increasing the resources to the ApplicationMaster which schedule the number of additional jobs to the cluster. Thus, result in job parallelism. On the other hand, when the algorithm decides that the current accumulated progress value is less than the previous, the algorithm stops adding more number of additional jobs to the cluster by simply decreasing the resources to the ApplicationMaster. The key point to remember at this moment is that by increasing the number of the parallel running job with an eventual increase in their associated throughput within constant resources is hard to achieve in reality.

4.1.1 Calculating Current Accumulated Progress of Running MapReduce Jobs

For the easy understanding figure **4.3 on page 35** shows how the actual accumulated current progress value is calculated over time. Here in figure **4.3 on page 35**, two jobs with their current progress values are running at time t-1 with MARP parameter x. Thus, the algorithm calculates the current accumulated progress value by adding those two values. Eventually, when the algorithm tries to calculate another current accumulated progress value over time then the same method that was used before doesn't work at this time. The reason is that there might be the changes in the situation of running MapReduce job. There might be three situations as:

- **Same job running at time t-1 and at time t with different progress values**
- **Job running at time t-1 finished at time t**
- **The new job might be added at time t**

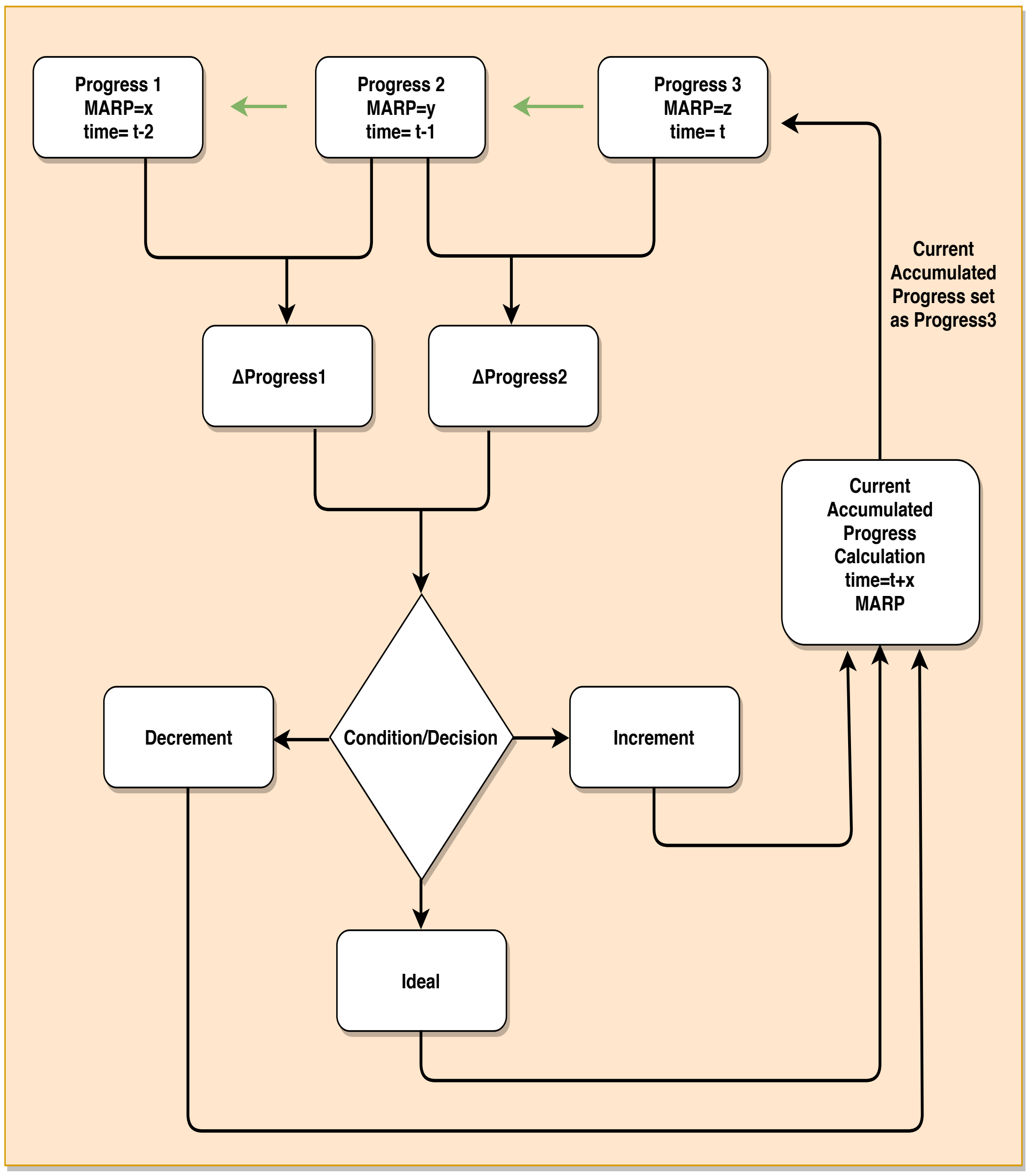


Figure 4.2: Architecture for current progress calculation each time the algorithm executed.

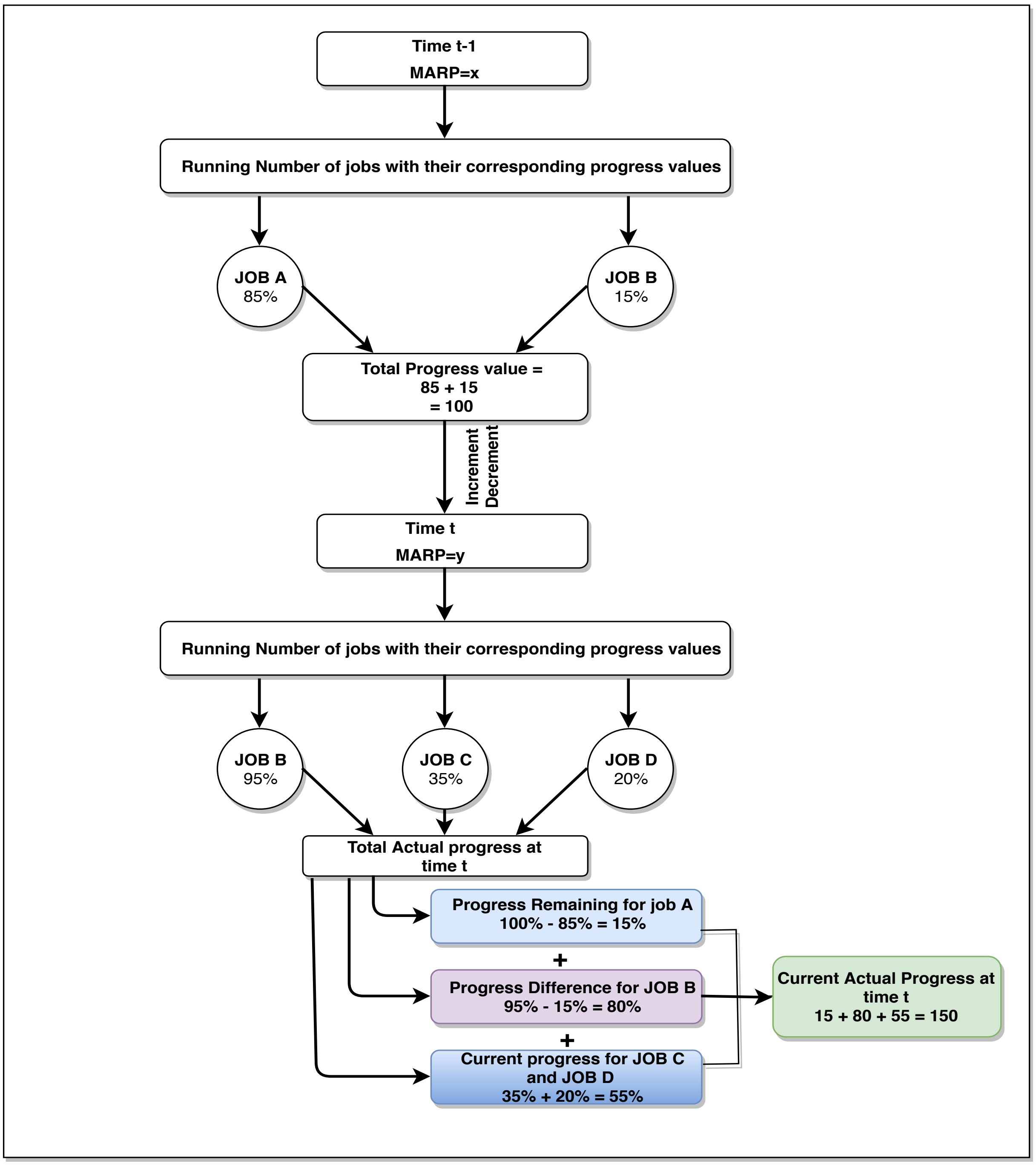

Time t-1
MARP=x
Running Number of jobs with their corresponding progress values
JOB A
85%
JOB B
15%
Total Progress value = 85 + 15 = 100
Increment
Decrement
Time t
MARP=y
Running Number of jobs with their corresponding progress values
JOB B
95%
JOB C
35%
JOB D
20%
Total Actual progress at time t
Progress Remaining for job A
100% - 85% = 15%
+
Progress Difference for JOB B
95% - 15% = 80%
+
Current progress for JOB C and JOB D
35% + 20% = 55%
Current Actual Progress at time t
15 + 80 + 55 = 150

Figure 4.3: Technical overview of accumulated current progress calculation over time.

Figure **4.3 on the previous page** aims to show the calculation of current accumulated progress value in all three situations. Each time the algorithm runs to catch the progress value, first, it will identify the changes in the situation of running MapReduce job and based on those changed situations, the algorithm applies different specific logic to gain actual current progress. For instance, the algorithm takes the difference in progress value between current progress value and previous progress value of the particular running MapReduce jobs and finally take the sum of those difference. The figure, **4.3 on the preceding page** helps to understand the same situation with JOB B. Running MapReduce job JOB B at time t-1 with progress value 15% continue at time t with progress value of 95%. Thus, the actual progress of JOB B at time t is 95-15=80.

Likewise, the algorithm subtracts the values of particular jobs form 100 and sum up those difference if the jobs running at time t-1 disappears at time t. The same situation in figure **4.3 on the previous page** is related to running MapReduce job-JOB A. JOB A at time t-1 with progress value of 85% is finished at time t, so the progress of job JOB A at time t can be calculated as 100-85=15.

Similarly, the corresponding progress values are added for the newly added running MapReduce jobs at time t. The same situation is illustrated by jobs JOB C and JOB D at time t with their corresponding progress values 35% and 20% respectively. So the current accumulated progress of the newly added jobs is 35+20=55.
Finally, the algorithm also sums up those current accumulated progress values which omes from different situations (80+15+55) and make the current accumulated progress value (150) of the running MapReduce jobs at time t. All of the above steps will be carried out by the algorithm each time it is supposed to calculate the accumulated current progress value.

4.1.1.1 Details Work-flow of Algorithm to Calculated Current Accumulated Progress

Initially, designed and developed algorithm is responsible for extracting running job id of the jobs and their corresponding progress value by using yarn command in two different arrays. The key point to notice here is that every running job has its own specific id which is unique. Another key thing to remember is, both the arrays have the same numbers of items stored each time the algorithm runs. Not only the same numbers of items but also the index value which is assigned to store job id that is equal to the index value assigned to store the corresponding progress values of a particular job id's. Figure, **4.4 on the facing page** gives more clear view on how two arrays get formed in order to store unique job ids and their associated progress value at different time t-1 and t with different MARP x and y respectively.

When the algorithm runs to get the current accumulated progress value at time t then the algorithm compares the currently stored ids with previous stored ids which were stored at time t-1 and identify the difference and similarities on the job ids. The figure, **4.5 on page 39** presents a clear view about all the technical steps in

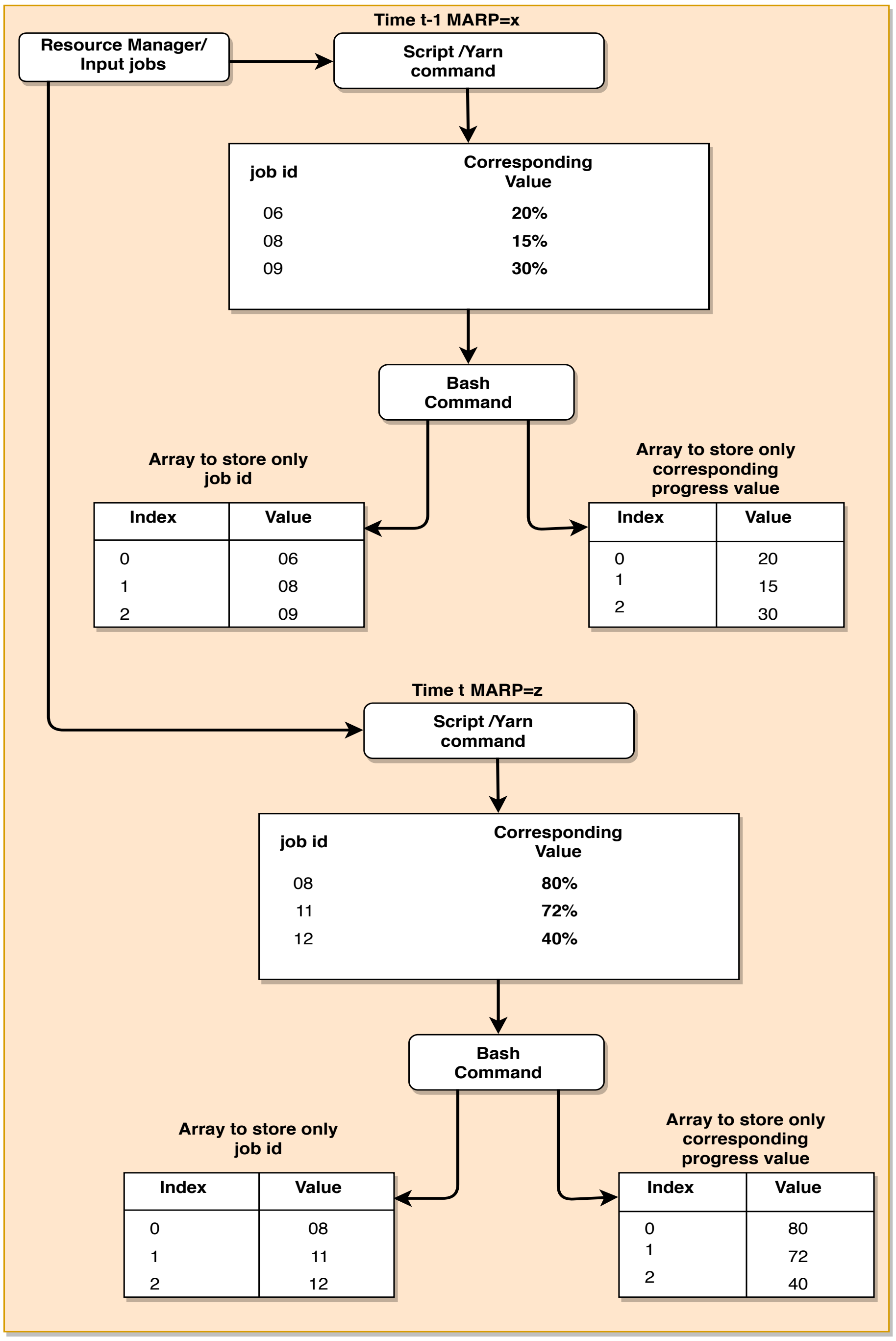


Figure 4.4: Two arrays storing unique job ids and corresponding progress value extract by algorithm.

order to calculate the current accumulated progress value. In the figure **4.5 on the facing page**, all the box containing number are a symbolic form of the different array. Boxes with color light blue shows the job ids and their corresponding values stored in arrays at time t-1 with MARP x. Likewise, the boxes with color light green means the current job ids and their corresponding progress values in arrays.

The array with different jobs now gets compared with arrays that store only job ids at time t-1 and at time t. Moreover, the algorithm will also form two new arrays that store the intersection part of job ids those were stored at time t-1 and at t. Meanwhile, the algorithm will find the corresponding index number of those intersected job ids form those particular arrays (light blue and light green) and search the same corresponding index number in the arrays that stored only values.

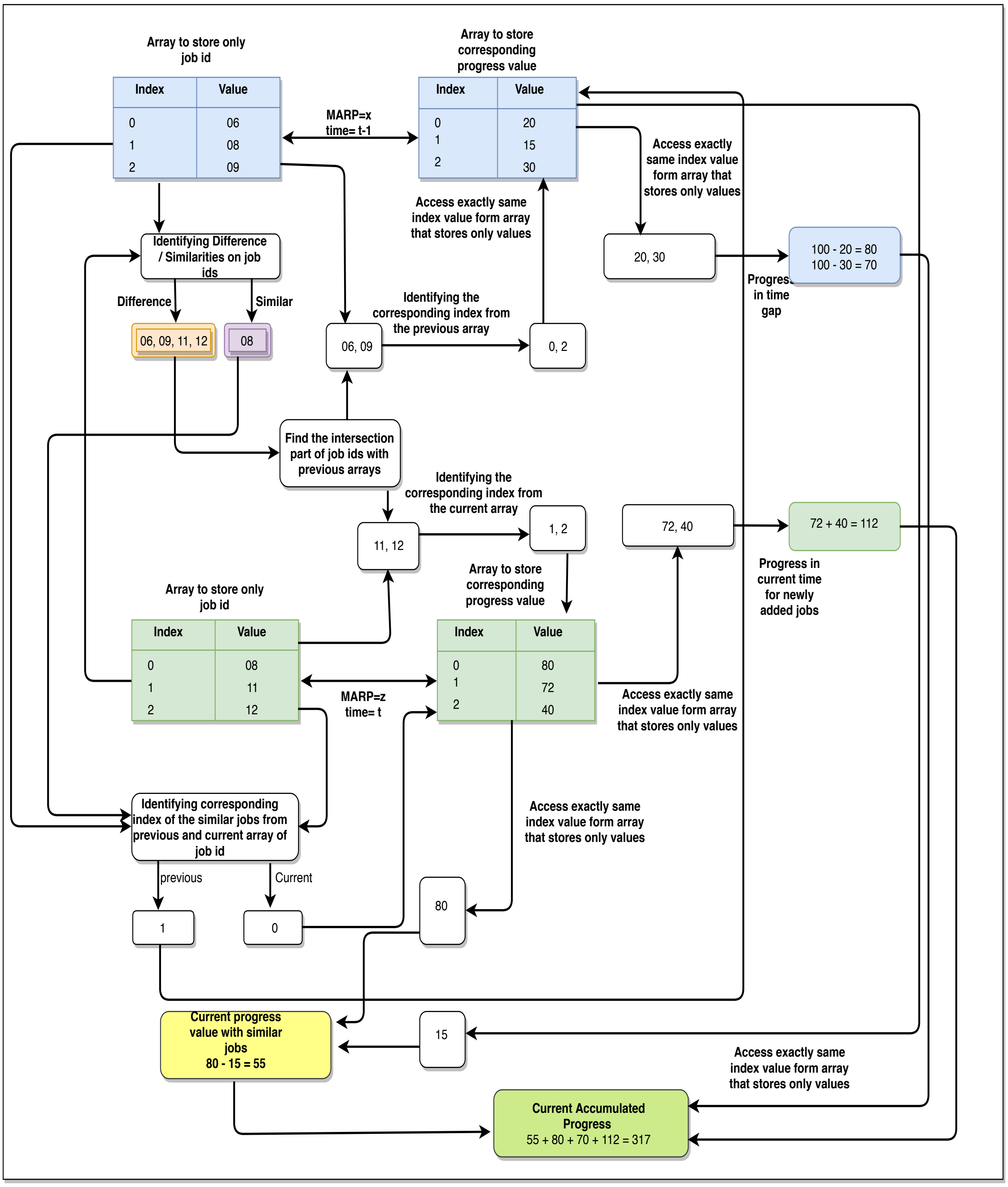
The algorithm is aware of the scenarios described in section (**see Section 4.1.1**) and which is being developed as an automation tool to calculate the different progress values on different scenarios. Indeed, the algorithms calculates the current accumulated progress value by adding the values obtained from different scenarios as shown by the dark green color box in the figure, **4.5 on the next page**.

4.1.2 Calculation of ΔProgress

In this thesis, by reading the progress values of running jobs, the algorithm identifies the amount of task finished for the particular MapReduce job over time. Moreover, progress values are different over time that depends upon the different scenarios i.e., resources allocated for ApplicationMaster, size of the scheduled jobs, number of scheduled jobs, resources allocation for actual job execution container and etc. For instance, fewer numbers of small size jobs in the Hadoop cluster has faster progress rate over time. On the other hand, more numbers of big size jobs with the same resources have slow progress rate over time.

Δprogress shows the improvement or decrement of the progress rate over time of currently executing job as compared to previous ones. Therefore, in order to achieve Δprogress1 and Δprogress2, the algorithm make the difference between progress value at time t-2 from t-1 and t-1 from t respectively.

The figure **4.6 on page 40** gives a more clear view of initial first part of the algorithm. In order to get the difference in progress rate over time, the algorithm checks and store value of progress at time t-2 with default configuration. As soon as the first progress from default configuration is achieved, the algorithm dynamically makes small change by increment on MARP parameter which means that the additional resources get added to the ApplicationMaster in the Hadoop cluster. Thus, resource manager might schedule an additional job to the cluster if added resources are good enough to execute the new job. The algorithm calculates the current accumulated progress value at time t-1. In the same way, the algorithm calculates Progress3 at time t by having a small increment on MARP parameter.



Array to store only job id
Index
Value
0 06
1 08
2 09
MARP=x time= t-1
Array to store corresponding progress value
Index
Value
0 20
1 15
2 30
Access exactly same index value form array that stores only values
Access exactly same index value form array that stores only values
20, 30
100 - 20 = 80
100 - 30 = 70
Progress in time gap
Identifying Difference / Similarities on job ids
Difference
Similar
06, 09, 11, 12
08
06, 09
Identifying the corresponding index from the previous array
0, 2
Find the intersection part of job ids with previous arrays
Identifying the corresponding index from the current array
11, 12
1, 2
72, 40
72 + 40 = 112
Progress in current time for newly added jobs
Array to store only job id
Index
Value
0 08
1 11
2 12
MARP=z time= t
Array to store corresponding progress value
Index
Value
0 80
1 72
2 40
Access exactly same index value form array that stores only values
Identifying corresponding index of the similar jobs from previous and current array of job id
previous
Current
1
0
80
Access exactly same index value form array that stores only values
15
Current progress value with similar jobs 80 - 15 = 55
Access exactly same index value form array that stores only values
Current Accumulated Progress 55 + 80 + 70 + 112 = 317

Figure 4.5: Figure exploring how data stored on arrays and get calculated in order to find current accumulated progress.

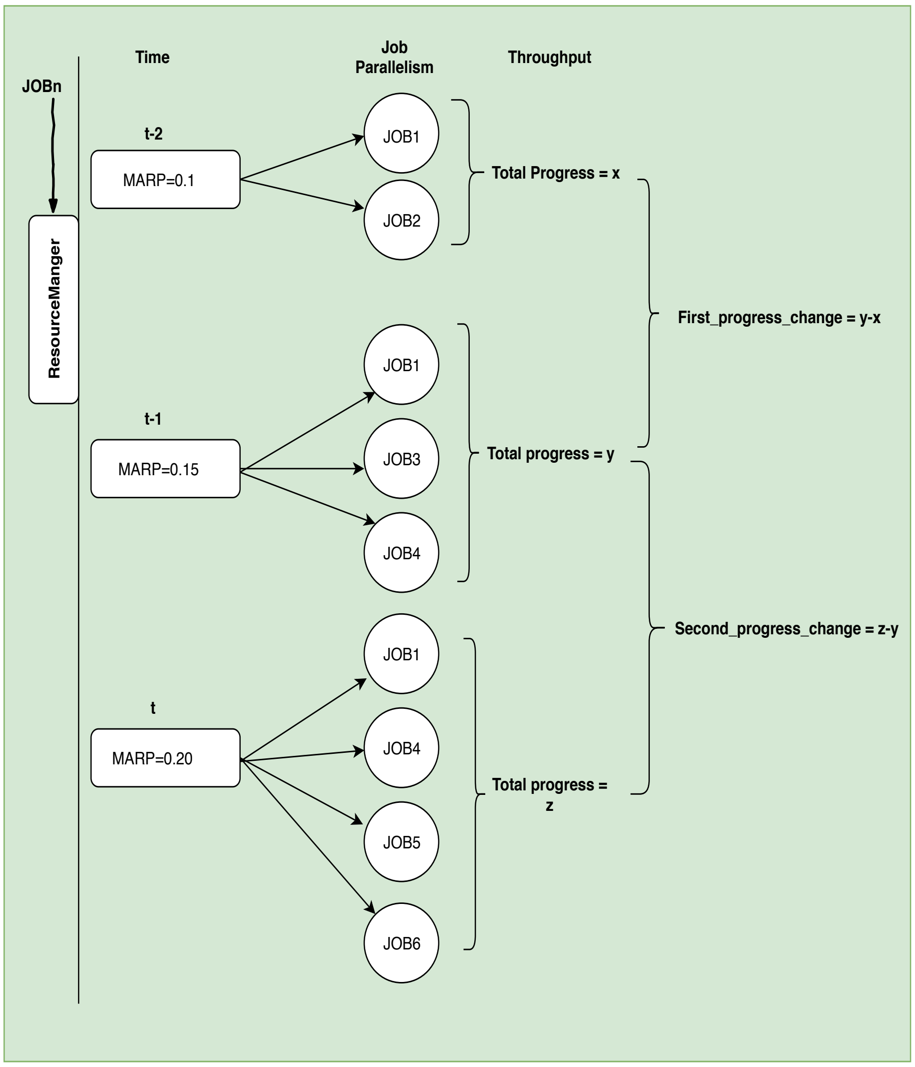


Figure 4.6: Architecture calculating progress at different time and MARP parameter during MapReduce job execution and calculates the change in progress.

From the figure **4.6 on the preceding page**, it can seen that the two MapReduce jobs are running at default configuration. Therefore, corresponding total progress x is the sum of the progress of all the running MapReduce job at time t-2. Similarly, as the MARP parameter increment to 0.15, 3 jobs are running. Among 3 jobs, 2 jobs are the newly scheduled jobs by the scheduler as soon as additional resources allocation for ApplicationMaster is realized. The remaining one is the job that starts executing with default configuration and is still running. Therefore, at time t-1 algorithm calculates the total progress y form three running job. Finally, the algorithm increments the MARP parameter to 0.20 as soon as it calculates and stores the progress y. At time t with MARP 0.20, 4 MapReduce jobs are running. Algorithm again calculates and stores the progress z at time t for those 4 MapReduce running jobs.

As soon as the algorithm has progress values x, y and z, then it calculates Δprogress over time.

- **Δprogress1 = (Total Progress at t-2) - (Total Progress at t-1)**
- **Δprogress2 = (Total Progress at t) - (Total Progress at t-2)**

4.1.3 Algorithm

The algorithm 1 is all about the overall design and the algorithm 2 is another algorithm that gets implements within the algorithm 1.

Algorithm 1 Progress Aware Algorithm

```
 1: Start
 2: Input number of jobs
 3: Set MARP at default configuration
 4: Calculate progress $P_1$ at default configuration
 5: Calculate **Current Accumulated Progress (Algorithm 2)** for $P_2$ , $P_3$ at
    different MARP values at different time
 6: Calculate $\Delta_{P_2}$ as $P_3 - P_2$ and $\Delta_{P_1}$ as $P_2 - P_1$
 7:             Check: if $\Delta_{P_2} > \Delta_{P_1}$ then
 8: ΔChange on MARP
 9:             Check: else if $\Delta_{P_2} < \Delta_{P_1}$ then
10: ΔChange on MARP
11: Calculate **Current Accumulated Progress (Algorithm 2)**
12: Update $P_1, P_2$ , $P_3$
13: Repeat step 6
14: else
15: Calculate **Current Accumulated Progress (Algorithm 2)**
16: Update $P_1, P_2$ , $P_3$
17: Repeat step 6
18: End of Algorithm 1
```

4.1.4 Flow Chart

The flow chart is shown in the figure **4.7 on the following page**.

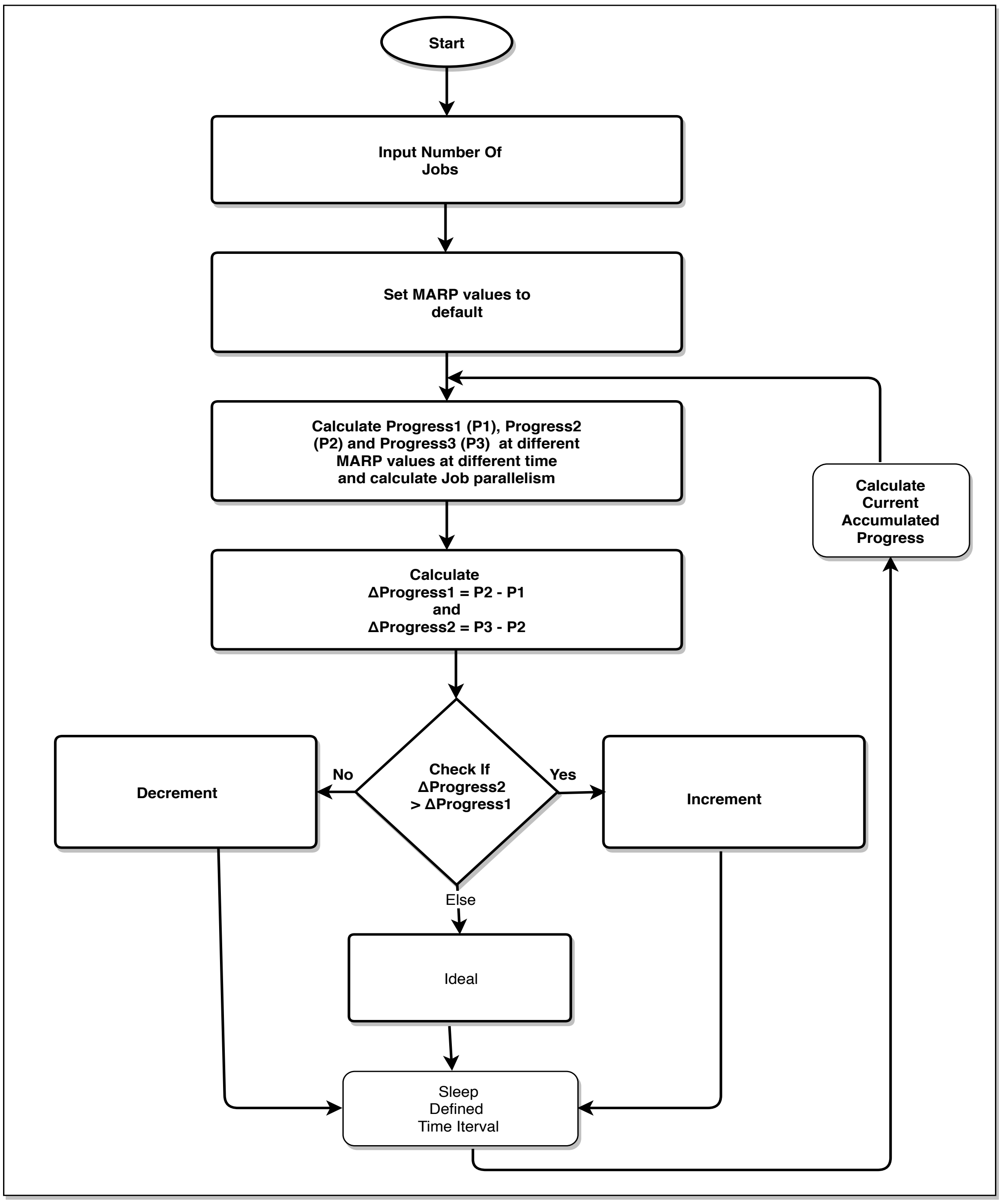

Start
Input Number Of Jobs
Set MARP values to default
Calculate Progress1 (P1), Progress2 (P2) and Progress3 (P3) at different MARP values at different time and calculate Job parallelism
Calculate ΔProgress1 = P2 - P1 and ΔProgress2 = P3 - P2
Check If ΔProgress2 > ΔProgress1
No
Yes
Decrement
Increment
Else
Ideal
Sleep Defined Time Iterval
Calculate Current Accumulated Progress

Figure 4.7: Flow chart for the designed algorithm.

Algorithm 2 Algorithm for Current Accumulated Progress

```
1: Start
2: Store running application id and corresponding progress value at time t-1
3: Store running application id and corresponding progress value at time t
4: Find differences and similarities on ids at time t and t-1
5: Each corresponding progress values of the ids only at time t-1 gets subtracted
   from 100 and make the sum1 by adding
6: Each corresponding progress values of the ids only at time t gets added and
   make sum2
7: Each corresponding progress value at time t gets subtracted form the
   corresponding progress value at time t-1. If the job ids are similar and make
   sum3 by adding those substracted progress values
8: Make **Actual Accumulated Progress value** by summing up sum1, sum2 and
   sum3
9: End of Algorithm 2
```

4.2 Expected Results of APTH Approach with the Progress Aware Algorithm

The illustration for expected results of the algorithm are depicted in figures **4.8 on the next page**, **4.9 on page 45** and **4.10 on page 46**.The key point to note is that there will always be quantitatively average throughput calculated by the end of the MapReduce execution time.

The result shown in the figure, **4.8 on the following page** is all about, how job parallelism can be increased with the dynamic increment of MARP parameter. As the parameter value of MARP is increased, eventually the number in parallel running jobs seems increased. However, static number of jobs are running with default configuration.

Similarly, figure **4.9 on page 45**, shows the dynamic change in increment in average throughput with APTH approach.

Figure, **4.9 on page 45** shows that there are 3 axis X, Y, and Z. X denotes the dynamic MARP during job execution. Similarly, the axis Y denotes the specific point of time. On the other hand, Z axis shows the corresponding throughput. Notably, the meeting point of the time at MARP value is denoted by triangles. The point nearby the triangle to Z-axis corresponds to related associated throughput at that time with particular MARP parameter sets with the proposed APTH approach. Figure aims to demonstrate, what is the effect on the throughput if proper utilization of resources cannot be held during job execution. The point is that, it is impossible always to set exactly perfect MARP value for high throughput requirements.

When configured MARP parameter is very small or very big, it consumes more time in order to finish executing all the submitted job(**see Section 3.1.1**). Therefore, the average throughput achieved will be less, as shown by the Z axis. The better throughput will only be achieved if MARP parameter could keep a good balance between the parallel running number of jobs and resources unit of the cluster.

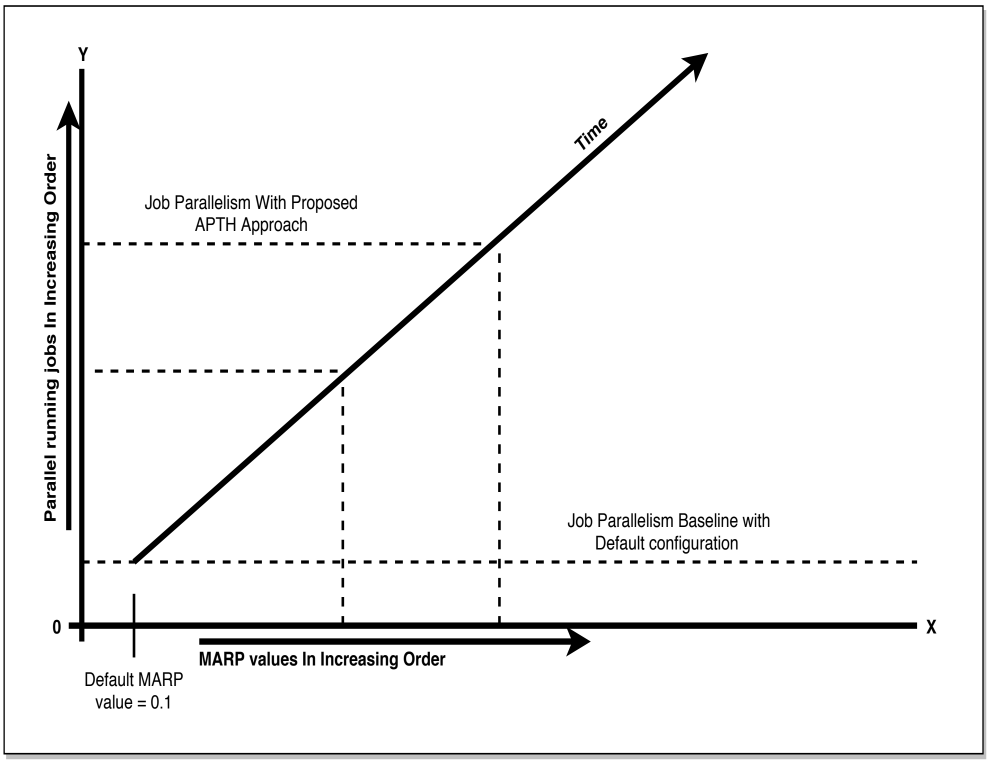

Y
Time
Parallel running jobs In Increasing Order
Job Parallelism With Proposed APTH Approach
Job Parallelism Baseline with Default configuration
0
X
MARP values In Increasing Order
Default MARP value = 0.1

Figure 4.8: Dynamic number of jos running with APTH approach and Static number of jobs running with default configuration.

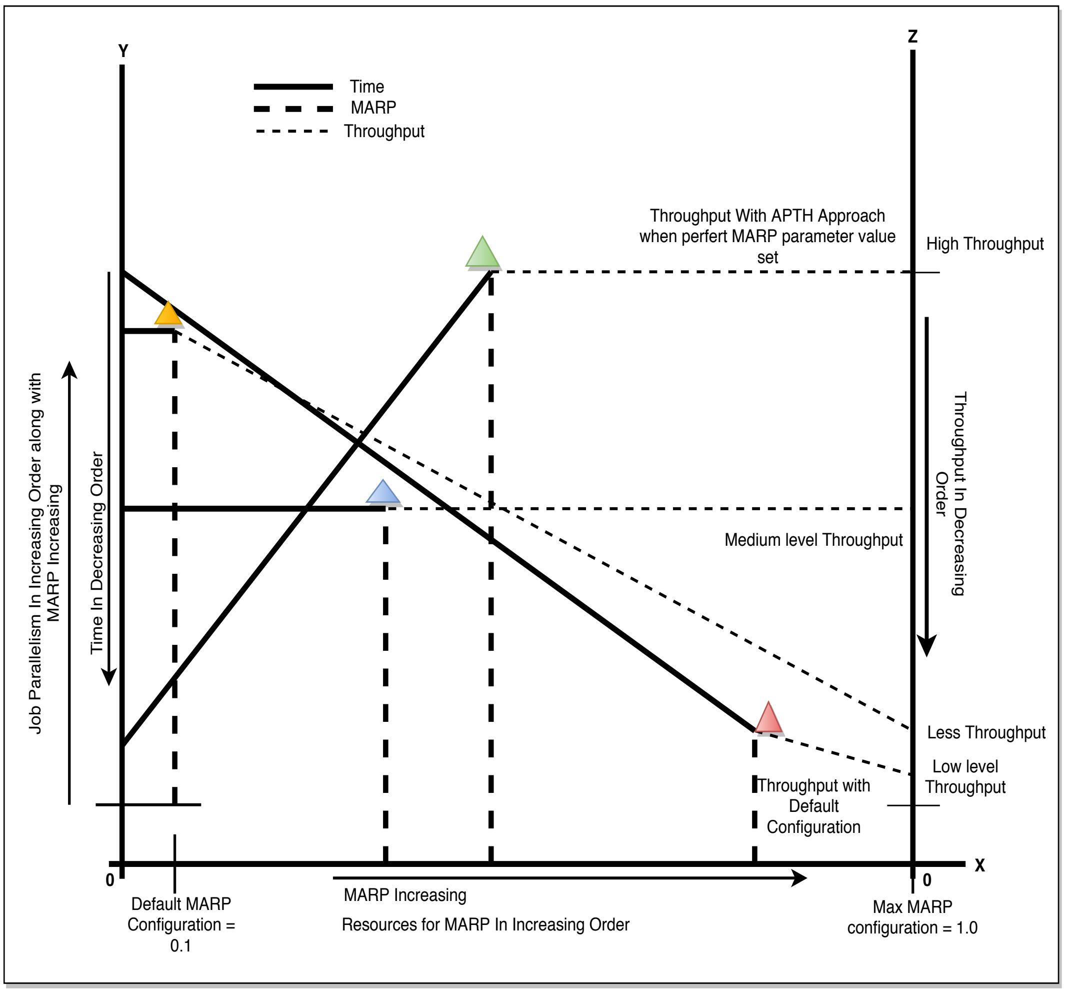

Y
Z
Time
MARP
Throughput
Throughput With APTH Approach when perfert MARP parameter value set
High Throughput
Throughput In Decreasing Order
Job Parallelism In Increasing Order along with MARP Increasing
Time In Decreasing Order
Medium level Throughput
Less Throughput
Low level Throughput
Throughput with Default Configuration
0
X
0
Default MARP Configuration = 0.1
MARP Increasing
Resources for MARP In Increasing Order
Max MARP configuration = 1.0

Figure 4.9: Dynamic throughput with APTH approach respect to MARP parameter value and time.

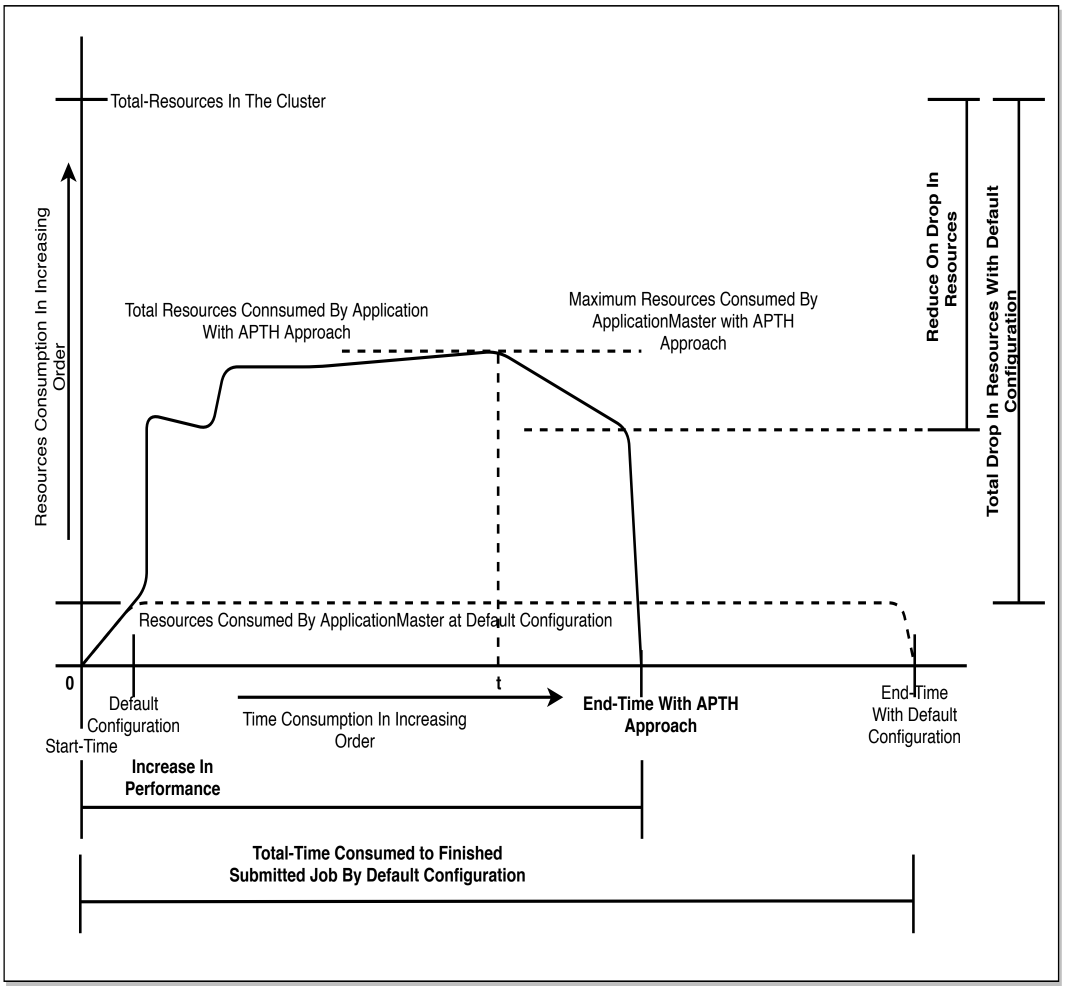

Total-Resources In The Cluster
Resources Consumption In Increasing Order
Total Resources Connsumed By Application With APTH Approach
Maximum Resources Consumed By ApplicationMaster with APTH Approach
Reduce On Drop In Resources
Total Drop In Resources With Default Configuration
Resources Consumed By ApplicationMaster at Default Configuration
0
t
Default Configuration
Start-Time
Time Consumption In Increasing Order
End-Time With APTH Approach
End-Time With Default Configuration
Increase In Performance
Total-Time Consumed to Finished Submitted Job By Default Configuration

Figure 4.10: Execution time difference and drop in resources reduced with APTH approach.

Finally, the figure **4.10** shows the total resources consumed by the ApplicationMaster with the proposed APTH approach and the resources consumed with default configuration. There must be overall execution time difference between the same number of jobs while executing in two different environments. As it can be seen in the figure **4.10**, not only reduction in time but also APTH approach have better resources utilization so, the drop in the amount of resources gets reduced. Therefore, better performance will be achieved with the proposed approach.

Chapter 5

Implementation

This chapter is all about the description of work done following the technical design for hadoop cluster creation and for the algorithm that is developed. This chapter consists of the important code snippets and system details. The staging environment is Google Cloud.

5.1 System Setup

An overall system setup for this project is the combination of Hadoop cluster configuration, Benchmarking tool installation and configuration, developed scripts for the designed progress aware algorithm.

5.1.1 Hadoop Cluster Creation

Initially, Hadoop cluster was created using Google Compute engines on the Google Cloud. Following steps were taken to create and configure Hadoop Cluster.

- Create virtual machines
- Install java on all the machines
- Download stable version of Hadoop (Hadoop-2.8.1) on all the machines
- Unpack the downloaded software on all the machines
- Distribute Authentication Key-pairs for the Hadoop worker nodes

5.1.1.1 Cluster Configuration

- Create Host File on Each Node

```
10.164.0.3 project-master-01
10.164.0.4 project-worker-01
10.164.0.2 project-worker-02
```

- Set JAVA_HOME And Environment Variables

```
export JAVA_HOME=/usr/lib/jvm/jdk
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_COMMON_HOME=/home/ramesh/hadoop-2.8.1
export HADOOP_MAPRED_HOME=$HADOOP_COMMON_HOME
export HADOOP_HDFS_HOME=$HADOOP_COMMON_HOME
export YARN_HOME=$HADOOP_COMMON_HOME
export PATH=$PATH:$HADOOP_COMMON_HOME/bin
export PATH=$PATH:$HADOOP_COMMON_HOME/sbin
```

- XML Files Configuration
  Along with downloaded Hadoop-2.8.1 software package, there are many XML files available in all the machine. Those files contain the property with name and associated value. In order to customize the configuration, those properties need to be edited by changing their corresponding values. Similarly, in order to configure the master node in the cluster, one of the XML files named *core-site.xml* should be set as below for this project.

  ```
  <configuration>
  <property>
  <name>fs.defaultFS</name>
  <value>hdfs://project-master-01</value>
  </property>
  </configuration>
  ```

  Likewise, *yarn-site.xml* XML file gets edited in order to configure the resource manager of the cluster.

  ```
  <configuration>
  <property>
  <name>yarn.resourcemanager.hostname</name>
  <value>project-master-01</value>
  </property>
  <property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
  </property>
  </configuration>
  ```

  Similarly, *hdfs-site.xml* is another configuration file that stores the distributed file system related configuration. For instance, this project makes sure fault tolerance redundancy.

  ```
  <configuration>
  <property>
  <name>dfs.replication</name>
  <value>2</value>
  </property>
  </configuration>
  ```

mapred-site.xml should be configured as YARN is a default framework for MapReduce operation.

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

5.1.1.2 Hadoop Cluster Initialized

- As soon as the Hadoop cluster is created and configured it should be initialized before input jobs given to the cluster. So in order to initialize it, first name node should be formatted with command *hdfs namenode -format*:

```
18/05/31 10:36:26 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   user = ramesh
STARTUP_MSG:   host = project-master-01/10.164.0.3
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.8.1
STARTUP_MSG:   classpath = /home/ramesh/hadoop-2.8.1/etc/hadoop
```

- Hadoop distributed file system (**hdfs**) should be initialized by running script with command *start-dfs.sh*:

```
Starting namenodes on [project-master-01]
project-master-01: starting namenode, logging to /home/ramesh/
hadoop-2.8.1/logs/hadoop-ramesh-namenode-project-master-01.out
project-master-01: starting datanode, logging to /home/ramesh/
hadoop-2.8.1/logs/hadoop-ramesh-datanode-project-master-01.out
project-worker-01: starting datanode, logging to /home/ramesh/
hadoop-2.8.1/logs/hadoop-ramesh-datanode-project-worker-01.out
project-worker-02: starting datanode, logging to /home/ramesh/
hadoop-2.8.1/logs/hadoop-ramesh-datanode-project-worker-02.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/ramesh/hadoop-2.8.1/
logs/hadoop-ramesh-secondarynamenode-project-master-01.out
```

- Moreover, with command *jps* one can see java process running in the cluster. Still resource manager is not running as yarn is not yet initialized. However, Namenode and DataNode are ready to go as:

```
2311 NameNode
2753 SecondaryNameNode
2516 DataNode
2909 Jps
```

- In order to make resource manager running in the cluster one should initialize yarn with command *start-yarn.sh*:

```
starting yarn daemons
starting resourcemanager, logging to /home/ramesh/hadoop-2.8.1/logs/
yarn-ramesh-resourcemanager-project-master-01.out
project-master-01: starting nodemanager, logging to /home/ramesh/
hadoop-2.8.1/logs/yarn-ramesh-nodemanager-project-master-01.out
project-worker-01: starting nodemanager, logging to /home/ramesh/
hadoop-2.8.1/logs/yarn-ramesh-nodemanager-project-worker-01.out
project-worker-02: starting nodemanager, logging to /home/ramesh/
hadoop-2.8.1/logs/yarn-ramesh-nodemanager-project-worker-02.out
```

- Lets check with *jps* command again, if resource manager is running.

```
3033 ResourceManager
2311 NameNode
3207 NodeManager
2753 SecondaryNameNode
4029 Jps
2516 DataNode
```

- In order to check the nodes running in the cluster with command *yarn node -list*:

```
18/05/31 11:07:32 INFO client.RMProxy:
Connecting to ResourceManager at project-master-01/10.164.0.3:8032
Total Nodes:3
         Node-Id             Node-State        Node-Http-Address
Number-of-Running-Containers
project-worker-02:44737           RUNNING       project-worker-02:8042
0
project-worker-01:32785           RUNNING       project-worker-01:8042
0
project-master-01:35231           RUNNING       project-master-01:8042
0
```

- All the configured nodes are running as they supposed to do. In order to check the distributed file system report with command *hdfs dfsadmin -report*:

```
Configured Capacity: 1585237499904 (1.44 TB)
Present Capacity: 1501372780544 (1.37 TB)
DFS Remaining: 1501372694528 (1.37 TB)
DFS Used: 86016 (84 KB)
DFS Used%: 0.00%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0
Pending deletion blocks: 0
```

```
-------------------------------------------------
Live datanodes (3):
```

- To check the application and its status then one can use command *yarn application -list*.

5.1.2 HiBench Benchmark Suite Installation and Configuration

HiBench benchmark suite is used in order to submit the numbers of jobs to the Hadoop cluster. First, HiBench benchmark suite needs to be downloaded to the master node and configured to make it running. To make HiBench adjustable with the Hadoop cluster being deployed, the *hadoop.conf* file should be created and configured as:

```
# Hadoop home
hibench.hadoop.home      /home/ramesh/hadoop-2.8.1

# The path of hadoop executable
hibench.hadoop.executable     ${hibench.hadoop.home}/bin/hadoop

# Hadoop configraution directory
hibench.hadoop.configure.dir  ${hibench.hadoop.home}/etc/hadoop

# The root HDFS path to store HiBench data
hibench.hdfs.master       hdfs://project-master-01:8020


# Hadoop release provider. Supported value: apache, cdh5, hdp
hibench.hadoop.release    apache
```

5.1.3 Testing of the Hadoop Cluster at Different Level of Static Configuration

In order to confirm if the cluster and benchmarking suite configuration works well, 50 jobs were executed for several times at different MARP values. The figure, **5.1 on the next page** shows the total time taken by the cluster in order to complete 50 jobs execution at different MARP configurations. Similarly, figure, **5.2 on the following page** shows the average throughput of those 50 executing jobs at different MARP configurations. After analyzing the data, it is confirmed that Hadoop cluster is ready for further experiment as it can be able to reconfigure the resources for the MapReduce jobs by changing the MARP parameter value.

5.1.4 Deployment of Automation Tool

The designed algorithm was deployed into the bash script to make the proper functioning prototype. This script consists of many functions and every single function has particular task to be done. The key functions of the script deployed are highlighted below with their functioning mechanism:

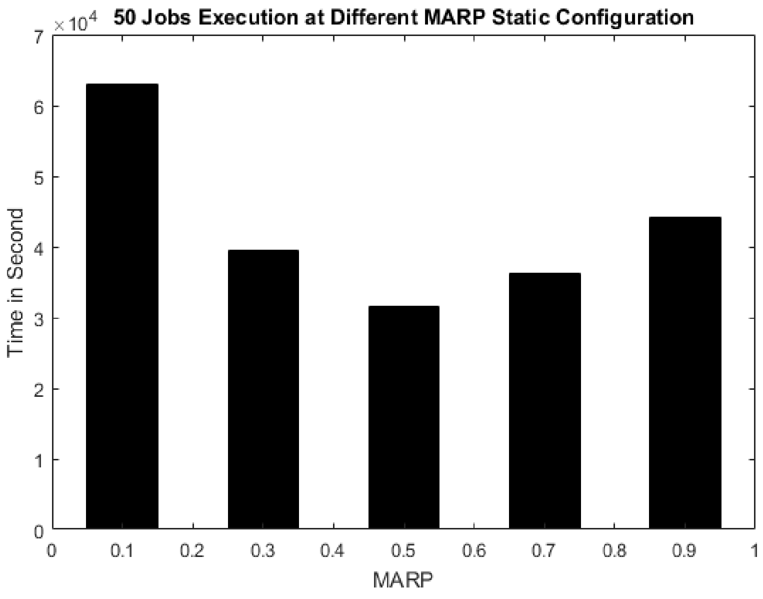


Figure 5.1: Total time consumption by 50 jobs at different MARP configuration.

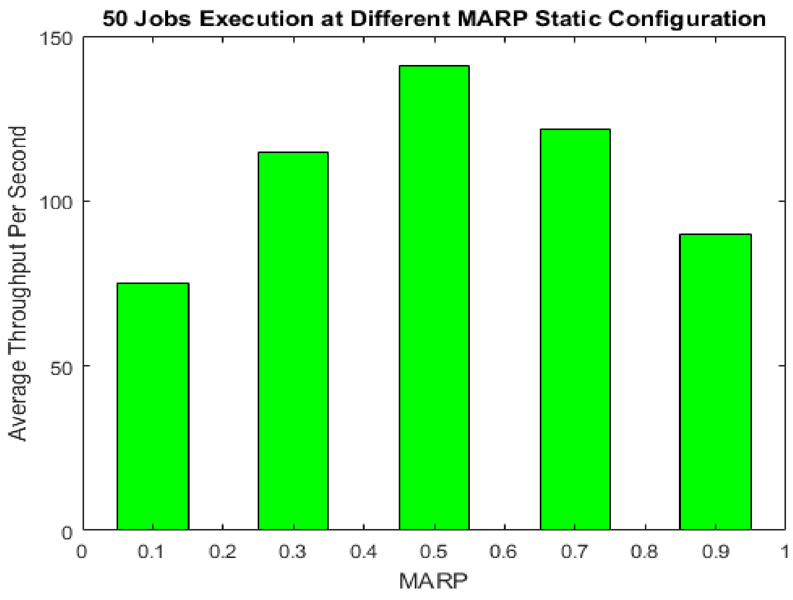


Figure 5.2: Average throughput per second at different MARP configuration.

- **Fetch and Set MARP Value**
  XML file *capacity-scheduler.xml* is the main file in which MARP parameter value dynamically get changed. During the experiment, the deployed script will fetch value of *yarn.scheduler.capacity.maximum-am-resource-percent* property and modify the value. Moreover, modified value again gets set to the same field in the file.

```
<configuration>
  <property>
    <name>yarn.scheduler.capacity.maximum-applications</name>
    <value>1000</value>
    <description>
      Maximum number of applications that can be pending and running.
    </description>
  </property>
  <property>
    <name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
    <value> 0.10 </value>
    <description>
      Maximum percent of resources in the cluster which can be used to run
      application masters i.e. controls number of concurrent running
      applications.
    </description>
  </property>
```

  One can notice the value 0.10, which means 10% of the total resources can be used by ApplicationMaster. If this value needs to be changed then, floating point number needs to be added or subtracted from it.

```
# FUNCTION TO FETCH MARP VALUE FORM FILE
file=/home/ramesh/hadoop-2.8.1/etc/hadoop/capacity-scheduler.xml
marp=$(sudo cat $file | awk -F" " 'NR==26 {print $2}')

# FUNCTION TO INCREMENT MARP VALUE
increment () {
newmarp=$(echo $marp $toadd | awk '{printf "%0.2f", $1 + $2}')
# echo "marp to submit is" $newmarp
awk 'NR==26{$2=a}1' a=$newmarp $file > tmp && sudo mv -f tmp $file
yarn rmadmin -refreshQueues
echo "MARP Increment by 0.10 and the new MARP is " $newmarp
        }

# FUNCTION TO DECREMENT MARP VALUE
decrement () {
newmarp=$(echo $marp $tosub | awk '{printf "%0.2f", $1 - $2}')
# echo "marp to submit is" $newmarp
awk 'NR==26{$2=a}1' a=$newmarp $file > tmp && sudo mv -f tmp $file
yarn rmadmin -refreshQueues
echo "MARP Decrement by 0.05 and the new MARP is" $newmarp
        }
```

Functions *increment* and *decrement* are responsible for actions incrementing or decrementing MARP parameter value to the file *capacity-scheduler.xml* based on the decision made by the algorithm. In the code above *yarn rmadmin -refreshQueues* is responsible for reconfiguring the changed MARP parameter in order to re-allocate the resources to the ApplicationMaster in the cluster. The defined global variable *toadd* and *tosub* contains the floating point number to take action with current MARP parameter value.

- **Fetch Cluster Metrics during MapReduce Operation**
  REST API (**see Section 3.1.4.5**) are used to fetch real-time data from the Hadoop cluster.

  ```
  curl http://project-master-01:8088/ws/v1/cluster/metrics
  ```

  For instance, when MapReduce operation is not running in the cluster then the output of the REST API looks like following. For this project, REST API is used to fetch real-time cluster metrics like, total memory in the cluster, total memory used, unused memory, marp limit, memory used by marp, total number of vcores in the cluster, total used, total number of running container, total number of parallel running job in the cluster etc.

```
{"clusterMetrics":{"appsSubmitted":51,"appsCompleted":32,"appsPending":
15,"appsRunning":4,"appsFailed":0,"appsKilled":0,"reservedMB":
0,"availableMB":0,"allocatedMB":24576,"reservedVirtualCores":
0,"availableVirtualCores":0,"allocatedVirtualCores":
20,"containersAllocated":20,"containersReserved":0,"containersPending":
21,"totalMB":24576,"totalVirtualCores":
20,"totalNodes":3,"lostNodes":0,"unhealthyNodes":
0,"decommissioningNodes":0,"decommissionedNodes":
0,"rebootedNodes":0,"activeNodes":3,"shutdownNodes":0}}
```

```
#### FUNCTION WHICH FETCH METRICES FROM RESOURCE MANAGER
fetch_metrics () {
# TOTAL MEMORY IN A CLUSTER
tot_mem=$(curl http://project-master-01:8088/ws/v1/cluster
/metrics | awk -F':' '{print $18}' | awk -F',' '{print $1}')
# MEMORY USED IN THE CLUSTER DURING JOB EXECUTION
mem_used=$( curl http://project-master-01:8088/ws/v1/cluster
/scheduler | awk -F':' '{print $22}' | cut -d',' -f1)
# UNUSED MEMORY IN THE CLUSTER
mem_unused=$(curl http://project-master-01:8088/ws/v1/cluster
/metrics | awk -F':' '{print $10}' | awk -F',' '{print $1}')
#TOTAL MEMORY ALLOCATED BY MARP VALUE
marp_limit=$( curl http://project-master-01:8088/ws/v1/cluster
/scheduler | awk -F':' '{print $55}' | cut -d',' -f1)
# TOTAL MEMORY USED BY APPLICATION MASTER
am_mem_used=$(curl http://project-master-01:8088/ws/v1/cluster
/scheduler | awk -F':' '{print $52}' | cut -d',' -f1)
                }
```

- **To Calculatle Current Accumulated Progress**
  This part of the script is responsible for calculating current accumulated progress. Actually, this section here is almost programming code for the one part of the algorithm **see Section 4.1.1**. Especially this part of the code represents overall programming modal for figure **4.4 on page 37** and figure **4.5 on page 39**. Initially at time t, running job application ids with their associated progress rate with command *yarn application -list* will be fetched. The output of this command will be application ids and associated progress values. In addition, these outputs will be stored on two different arrays. Furthermore, the array with job ids will be compared with another array with job ids at time t-1 and then get the difference on job ids. The difference and similarities with job ids will be detected and identified such as which job ids were running only at time t-1 and which job ids were at time t-1 and also continues running at time t, and which job ids were only running at time t. Based on this identification the algorithm finds the corresponding index value for those job ids and search progress values with those index on another array that only stores progress value. After fetching those values, the algorithm calculates current accumulated progress as shown in figure **4.3 on page 35**.

```
#### FUNCTION TO CALCULATE THE TOTAL CURRENT ACCUMULATED PROGRESS
progress_current () {
value_current=$(yarn application -list |
grep 'root' | grep 'RUNNING'|
awk '{print $8}' | cut -d'%' -f1 |
awk '{total = total + $1}END{print total}')
#echo "$value_current" > progress.txt
app_id_current=$(yarn application -list | grep "root" |
grep "RUNNING" | awk '{print $1}')
value_current=$(yarn application -list | grep "root" |
grep "RUNNING" | awk '{print $8 $9}'|
cut -d '%' -f1 | awk -F'.' '{print $1}'|
awk -F'[^0-9]*' '{print $1 $2}')
while [[ "${#app_id_current[@]}" != "${#value_current[@]}" ]]; do
        app_id_current=()
        value_current=()
        app_id=$(yarn application -list | grep "root" |
        grep "RUNNING" | awk '{print $1}')
        value_initially=$(yarn application -list | grep "root" |
        grep "RUNNING" |
        awk '{print $8 $9}'| cut -d '%' -f1 | awk -F'.' '{print $1}'|
        awk -F'[^0-9]*' '{print $1 $2}')
done

app_id=$(sudo cat /home/ramesh/progress_third_only_app_id.txt)
value_initially=$(sudo cat /home/ramesh/progress_third_only_value.txt)
```

```
echo "$app_id_current" > progress_third_only_app_id.txt
echo "$value_current" > progress_third_only_value.txt


curr_id_only=($(paste <(echo "$app_id_current")))
echo "CURRENT APP ONLY ID"
echo "${curr_id_only[@]}"

curr_value_only=($(paste <(echo "$value_current")))
echo "CURRENT APP ONLY VALUE"
echo "${curr_value_only[@]}"

curr_id_value=$(paste <(echo "$app_id_current") <(echo "$value_current"))
echo "CURRENT APP ID AND CORRESPONDING VALUES"
echo "$curr_id_value"

#####
pre_id_only=($(paste <(echo "$app_id")))
echo "PREVIOUS APP ONLY ID"
echo "${pre_id_only[@]}"

pre_value_only=($(paste <(echo "$value_initially")))
echo "PREVIOUS APP ONLY VALUE"
echo "${pre_value_only[@]}"
```

The algorithm is comparing currently running job ids at time t with job ids at time t-1. This helps algorithm to identify three scenarios as explained in (**section 4.1.1**). It can be noticed that the two array *app_id* with job ids running at time t-1 and *app_id_current* as currently running job ids are comparing with each other in order to find the difference in job ids.

```
different=$(diff -ia --suppress-common-lines
<( printf "%s\n" "${app_id[@]}" ) <( printf "%s\n" "${app_id_current[@]}"))
```

Now, the algorithm starts identifying the intersection part with the job ids. Moreover, the algorithm finds the ids as a newly added job, previously finished job, and parallel continuing job. Furthermore, the algorithm also searches the values for the corresponding job ids and do the calculation in order to find the current accumulated progress.

```
#intersection_with_current
####
for item1 in ${app_id_current[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_current+=( "$item1" )
        fi
    done
done
echo "FOLLOWING APPS ARE NEWELY ADDED"
echo ${intersection_with_current[@]}
###
#TO FIND THE ID OF THE NEWELY ADDED JOB
###
for ((i=0; i < ${#curr_id_only[@]}; ++i))
   do
        for j in "${intersection_with_current[@]}"
          do
                if [[ "${curr_id_only[$i]}" == "$j" ]]
                then

                        index_arr_curr+=( "$i" )
                fi
        done
  done

echo "The list of the index for newely added jobs are"
echo ${index_arr_curr[@]}

###
for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_arr_curr[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_newly_added+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding are"
echo ${sum_newly_added[@]}

###
sum1=0
for i in ${sum_newly_added[@]}
 do
        sum1=`echo $sum1 + $i | bc`
 done
echo "The total sum of the currently added job progress is" $sum1

###
#intersection_with_previous
###
```

```
for item1 in ${app_id[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_previous+=( "$item1" )
        fi
    done
done
echo "FOLLOWING APPS WERE IN PREVIOUS BUT NOT IN CURRENT"
echo ${intersection_with_previous[@]}
###
# ADD THE VALUES SUBTRACTIONG FROM 100
###

for ((i=0; i < ${#pre_id_only[@]}; ++i))
   do
        for j in "${intersection_with_previous[@]}"
          do
                if [[ "${pre_id_only[$i]}" == "$j" ]]
                then

                        index_arr_pre+=( "$i" )
                fi
        done
  done

echo "The list of the index for previous jobs
#which are not in current job list are"
echo ${index_arr_pre[@]}

###
for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
        for j in "${index_arr_pre[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_pre_added+=( "${pre_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding previous jobs are"
echo ${sum_pre_added[@]}
```

- **Reset Arrays**
  When the algorithm runs frequently at every defined time interval then the

array starts appending values on it. However, this logic is not efficient in this project as index value on both the array will not be same, results will cause mismatch. In order to eliminate mismatch on index values, the author of this thesis resets all the defined array every time it executes.

```
    ####FUNCTION THAT RESET THE ARRAY EVERY TIME LOOP EXECUTE
reset_array () {
        pre_id_only=()
        pre_value_only=()
        curr_id_only=()
        curr_value_only=()
        different=()
        fetch_file_data=()
        app_id_current=()
        intersection_with_current=()
        index_arr_curr=()
        sum_newly_added=()
        app_id=()
        intersection_with_previous=()
        index_arr_pre=()
        sum_pre_added=()
        similar_curr=()
        index_similar_curr=()
        index_similar_pre=()
        sum_similar_curr=()
        sum_similar_pre=()
        }
```

- **To Calculate the ΔProgress**
  In order to calculate the difference in progress value calculated at time t-2 will be subtracted from the value at time t-1 which gives ΔProgress1. Similarly, to get ΔProgress2 progress value at time t-2 get subtracted from progress value at time t. Each time the algorithm runs, the algorithm will calculate Progress3 and previous values from Progress3 and Progress2 are swapped. For more details about the calculation of ΔProgress and swapped values **see Section 4.1.2**.

```
    #### FUNCTION TO CALCULATE THE DIFFERENCE
    #BETWEEN SECOND AND FIRST PROGRESS
diff_first_speed () {
        fetch_second_value=$(sudo cat /home/ramesh/
        progress1.txt | awk '{print $1}')
        fetch_first_value=$(sudo cat /home/ramesh/
        progress2.txt | awk '{print $1}')
        echo "Second progress value and first
        progress value are" $fetch_first_value $fetch_second_value
        speed_first=`echo $fetch_first_value - $fetch_second_value | bc`
        }
    #### FUNCTION TO CALCULATE THE DIFFERENCE
    #BETWEEN THIRD AND SECOND
diff_second_speed () {
        fetch_third_value=$(sudo cat /home/ramesh/
        progress3.txt | awk '{print $1}')
        fetch_second_value=$(sudo cat /home/ramesh/
        progress2.txt | awk '{print $1}')
        echo "Third progress value and second
        progress value are" $fetch_third_value $fetch_second_value
        speed_second=`echo $fetch_third_value - $fetch_second_value | bc`
        }
```

- **To Write the Data into File**
  In order to write the data into the file which are collected from the overall algorithm during MapReduce operation one function is created and implemanted. Each time algorithm executes will append data into the file. In addition, these data is written into CSV file which is finally imported into RStudio for analysis.

```
    #### FUNCTION TO WRITE THOSE METRICS INTO FILE
write_file () {
                #### TO WRITE THE METRICS FORM THE CLUSTER INTO FILE
        var=$(paste -d, <(echo "$tot_mem") <(echo "$mem_used")
        <(echo "$mem_unused") <(echo "$marp") <(echo "$marp_limit")
        <(echo "$am_mem_used") <(echo "$am_vcore_used") <(echo "$tot_core")
        <(echo "$core_used") <(echo "$core_unused") <(echo "$app_running")
        <(echo "$app_pending") <(echo "$cont_running")
        <(echo "$cont_pending") <(echo "$capacity_used")
        <(echo "$speed_first") <(echo "$speed_second"))
        echo "$var" >> output_dynamic."csv"
        }
```

5.1.5 Pre-experiment Evaluation

To ensure that the designed prototype works as expected, they were tested multiple times by running the MapReduce job. Hadoop cluster with HiBench benchmark suite was extensively used for MapReduce operation. Figure, **5.3 on the facing page** provides the more clear views. In the figure, **5.3 on the next page** HiBench benchmark suite consists the script design by the author of this thesis. There are two scripts, one for data preparation and another for MapReduce execution. Data preparation script runs only for one time in order to prepare data to be used for MapReduce operation. Moreover, HDFS stored those prepared data. As soon as another script is launched, it starts MapReduce operation for the defined number of jobs. For instance, the script is written in a way that executes MapReduce operation for multiple time with the same input data that is being fetched from HDFS. In order to have the dynamic allocation of resources, concurrently the script developed for the designed algorithm gets to run which calculates, fetches and write the data to the file. In order to run jobs through HiBench, initially, there must

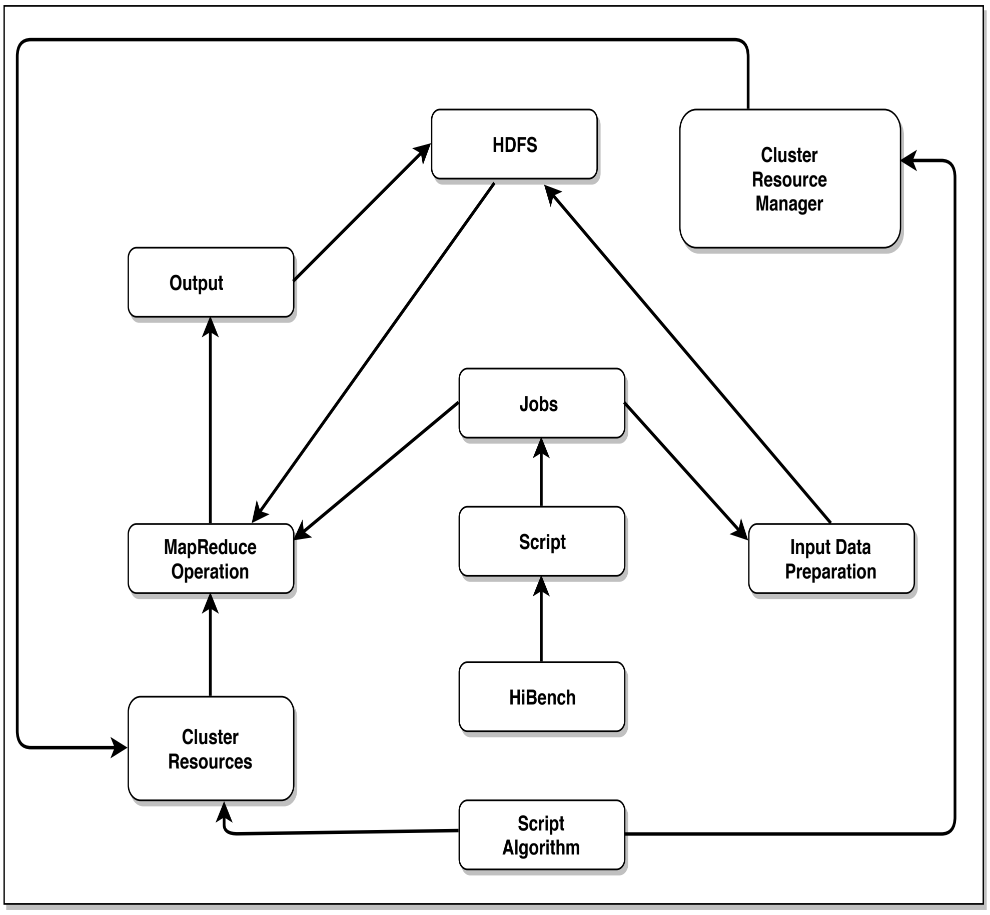


Figure 5.3: Overall implementation of algorithm

be one directory created in *hdfs* with named HiBench and give it permission with *chmod* command.

```
hadoop fs -mkdir /HiBench
hadoop fs -chmod 777 /HiBench
hadoop fs -mkdir /tmp
hadoop fs -chmod 777 /tmp
```

- **Prepare Input Data**
  To run job there must be data. HiBench itself has script in it to produce data before MapReduce operation takes place.

  ```
  bin/workloads/micro/wordcount/prepare/prepare.sh
  ```

  The *prepare.sh* launches a Hadoop job to generate the input data on HDFS[1]. The directory *Input/Wordcount* get created inside the HiBench directory in hdfs just before created.

[1]https://github.com/intel-hadoop/HiBench/blob/master/docs/run-hadoopbench.md

- **Run MapReduce Job**
  To run MapReduce operation into those generated data

  ```
  bin/workloads/micro/wordcount/hadoop/run.sh
  ```

  The *run.sh* submits a Hadoop job to the cluster. The directory *Output/Wordcount* get created inside the HiBench directory in hdfs.

- **Report**
  Inside the root directory of *HiBench* there is another subdirectory *report* which contains file *hibench.report* which is a summarized workload report, including workload name, execution duration, data size, throughput per cluster, throughput per node.

  ```
   Type          Date        Time      Input_data_size      Duration(s)
   Throughput(bytes/s)  Throughput/node

  HadoopWordcount 2018-04-23 12:10:04 37474                71.314
  525                    175
  ```

- **Output of the Deployed Script**
  On the other hand, developed script runs on the master machine while MapReduce job is running in the cluster. Along with many actions taken by the script, it also writes the file with cluster metrics data each time it executes. The file consists of various columns and each time algorithm runs will append the data to the file. Finally the file with data will be download into local machine and are analyzed.

Chapter 6

Measurement, Analysis and Comparison

This chapter is all about the experiments that is conducted in order to capture empirical data, as well the analysis that will be ensured. Figure, **6.1 on the following page** provides a more clear view about the experiment. The experiment initially conducted with default configuration and the data is being captured during MapReduce operation. Likewise, the same experiments is conducted with the random MARP vlue and with APTH approach (**see Section 3.1**). The reason for the experiment conducting for random MARP value is to prove that even an expert system administrator if they do static configuration, it might not give the best performance. Another intention of conducting an experiment for random MARP value is to show two problems that were described in (**Section 3.1.1**). The experiments were performed to get a measure of job parallelism and throughput with default configuration (**see Section 2.3.2.3**) and with the APTH approach. Furthermore, measurement of the average time taken by the experiments were analyzed in order to compare the performance. On the other hand, analysis of the data with consumed resources unit during MapReduce operation is another important factor in order to show the proof-how performance improves or degrades.The data obtained were analyzed in the analysis section to make clear difference in job parallelism and throughput between default configuration and APTH approach.

The total memory available in the cluster was 24576 MB and the total number of the virtual core in the cluster were 24. One of the particular interest of the experiments is to see how the resources used by ApplicationMaster (**see Section 2.3.2.1**) is increased or decreased with the change in MARP parameter value. The experiment aims to see not only resources increased or decreased, but also the change in the concurrent running number of jobs along with resources changes for ApplicationMaster. The difference in the average time taken by overall experiment execution and an average throughput of the experiment helps into further analysis about the optimal and appropriate resources utilization.

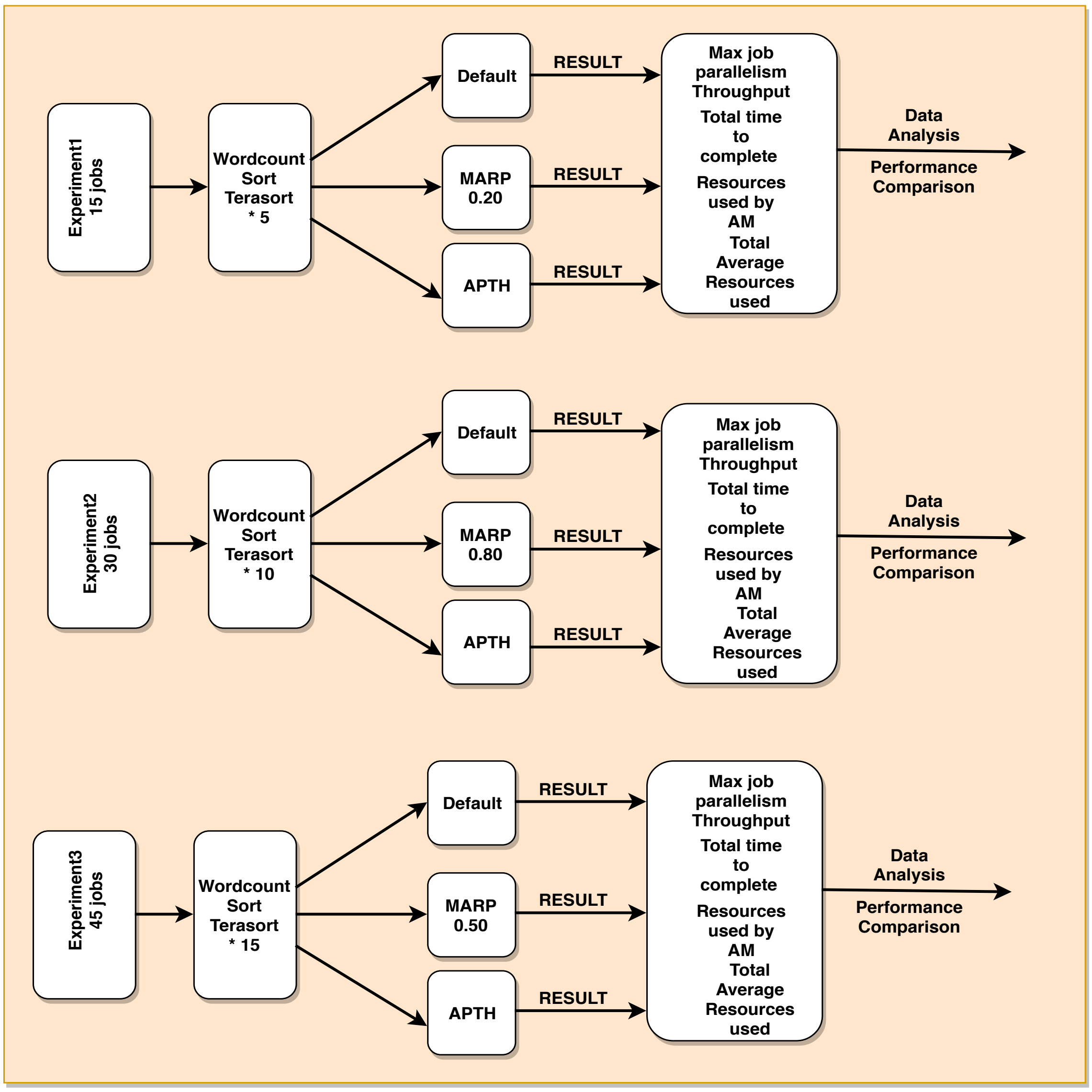


Figure 6.1: Three experiments, each with three configurations.

6.1 The Experiment

Three categories of the experiment has been performed with a distinct input job numbers: 15, 30 and 45. The script that submits jobs and the scripts for the algorithm executed simultaneously. Initially just after job submission, Hadoop cluster take some time to prepared itself. Thus, this cluster preparation time was not the part of the measurement. Furthermore, the executed script of the algorithm only captured data if it fetches the data for running number of jobs in the cluster is greater than zero. Finally, job execution time for every single job will be measured from the time it was accepted by the cluster resource manager. Furthermore, line graph with the different color has been drawn that helps to understand more. Brown, Blue, and Green are the color chosen for the line graph representing the configurations Default, Random and APTH respectively.

Data collection was slightly different on the static default and with random configuration rather than APTH approach. Only Hadoop REST API was used to collect data with static configuration. However, overall algorithm designed and developed collects all data including progress value over time.

6.1.1 Experiment1: Processing 15 Jobs

In the first experiment with 15 jobs, MapReduce operation was run over the Default configuration, Random MARP value, and APTH approach for three separate times and the data were captured separately. Numbers of the parallel running jobs, memory used by the ApplicationMaster, and overall memory used by the experiment were the things to be a studied. The figure, **6.2 on the next page** shows job parallelism on different configurations. Similarly, in the figure **6.3 on page 67**, the red line shows the threshold in the memory for the ApplicationMaster as defined by MARP value and different color shows the used memory resources by ApplicationMaster in different configuration. The figure, **6.4 on page 68** is all about the total memory resources used by the experiment with different configurations. Finally, the summary table **6.1** which shows time, overall resources used, throughput, and job parallelism with the different configuration.

Table 6.1: Experiment-1 Summary Table

Configurations	Max job parallelism	Average throughput	Total time	Capacity Used
Default	1	439 bytes/sec	5559 sec	34.75%
MARP 0.20	2	669 bytes/sec	4171 sec	56.73%
APTH Approach	4	726 bytes/sec	3282 sec	86.57%

It can be noticed that the result of APTH approach by studying statistical data from the summary table **6.1**, as job parallelism is improved by 300% and 100% compared to Default and MARP 0.20 configurations respectively. Likewise, the average throughput has an improvement of 40% and 8%. In the same fashion, there is also 41% and 22% reduction in total time consumption with the APTH approach compared to Default and MARP 0.20 configuration. Finally, this all the results are positive and possible because of APTH approach which does appropriate optimum

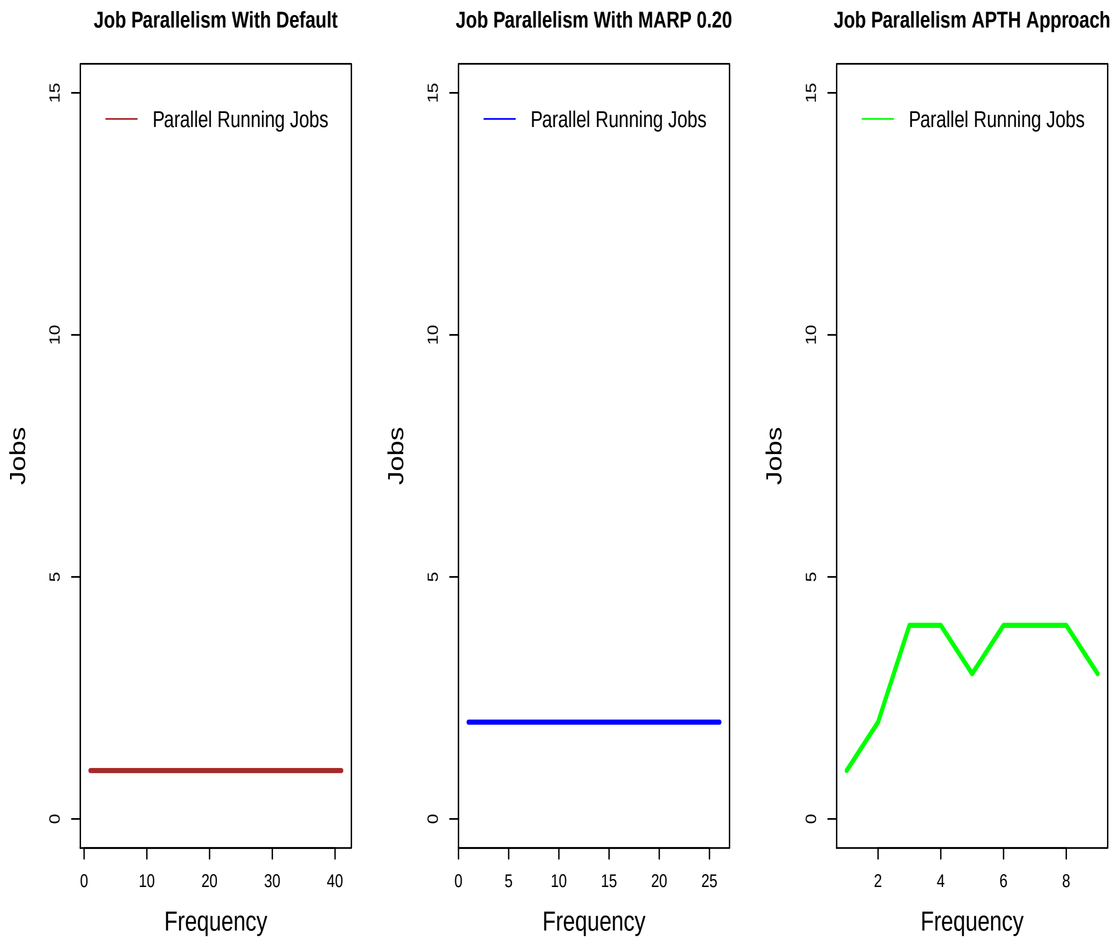


Figure 6.2: Parallel running number of jobs during 15 job execution.

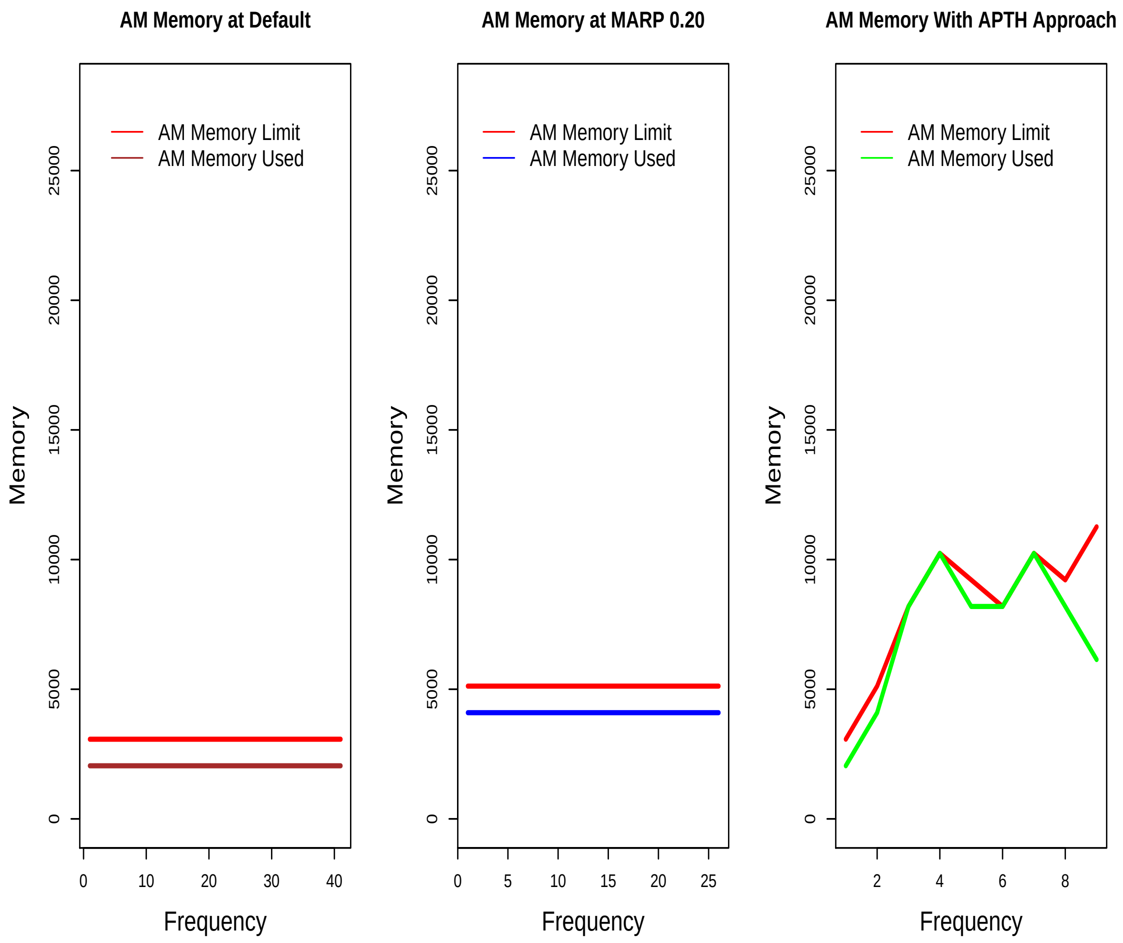


Figure 6.3: Memory consumption by ApplicationMaster(AM) during 15 job execution.

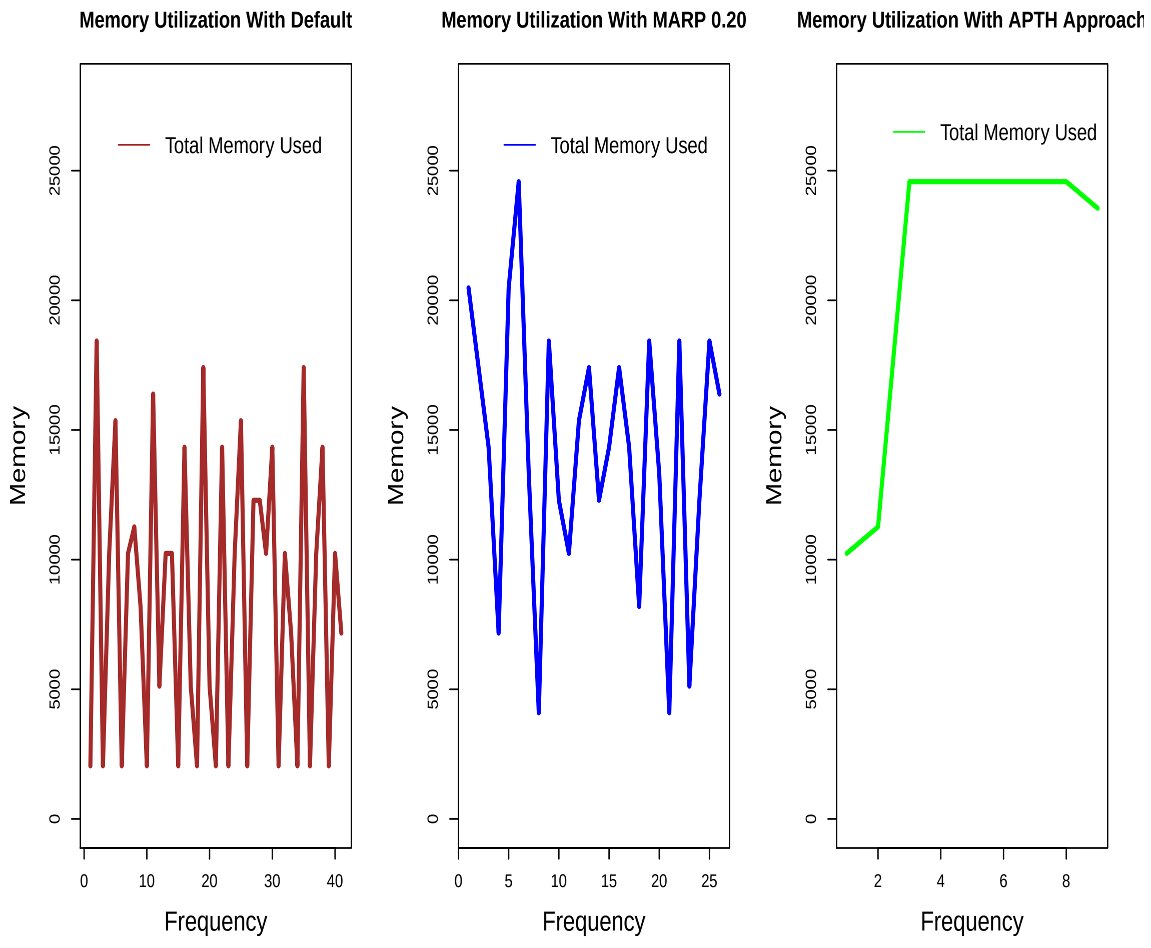


Figure 6.4: Overall Memory consumption during 15 job execution.

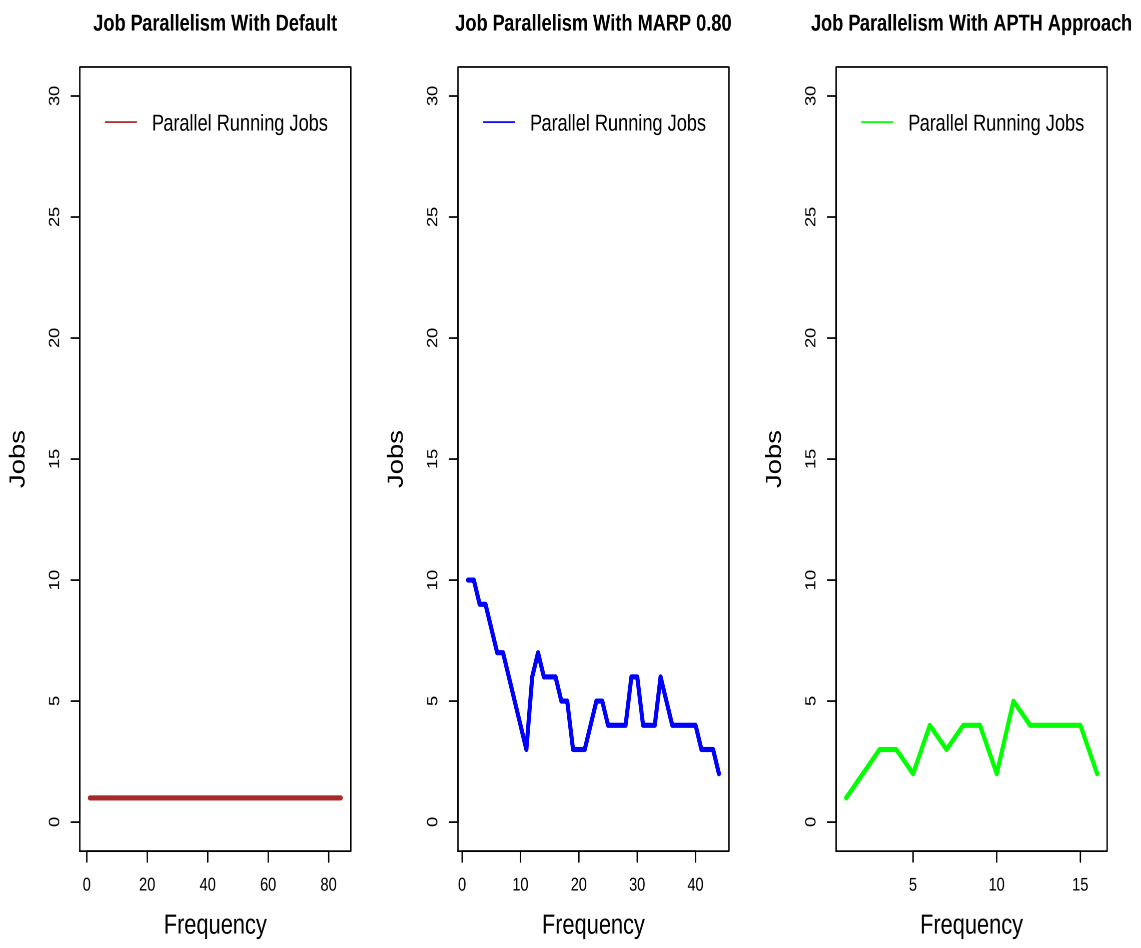


Figure 6.5: Parallel running number of jobs during 30 job execution.

resource utilization. It can also be noticed that the *ideal resources* in the cluster are gradually decreased with MARP 0.20 configuration and with APTH approach.

6.1.2 Experiment-2: Processing 30 Jobs

Figures, **6.5** is showing job parallelism, **6.6 on the next page** is representing memory resource limit and memory resource used by ApplicationMaster, and **6.7 on page 71** which gives an overview of total memory used during the experiment. In addition, Table **6.2 on page 72** summarized the results. In this experiment, 0.80 value is set to be the random value for MARP.

As it can be seen in Experiment-2, numbers in the parallel running job with APTH approach compared to the Default configuration was increased up to 400%. However, there was 50% reduction in the parallel running job compared to MARP 0.80. The reason is that 80% resources is allocated for ApplicationMaster for more running jobs with MARP 0.80. Thus, remaining 20% resources are

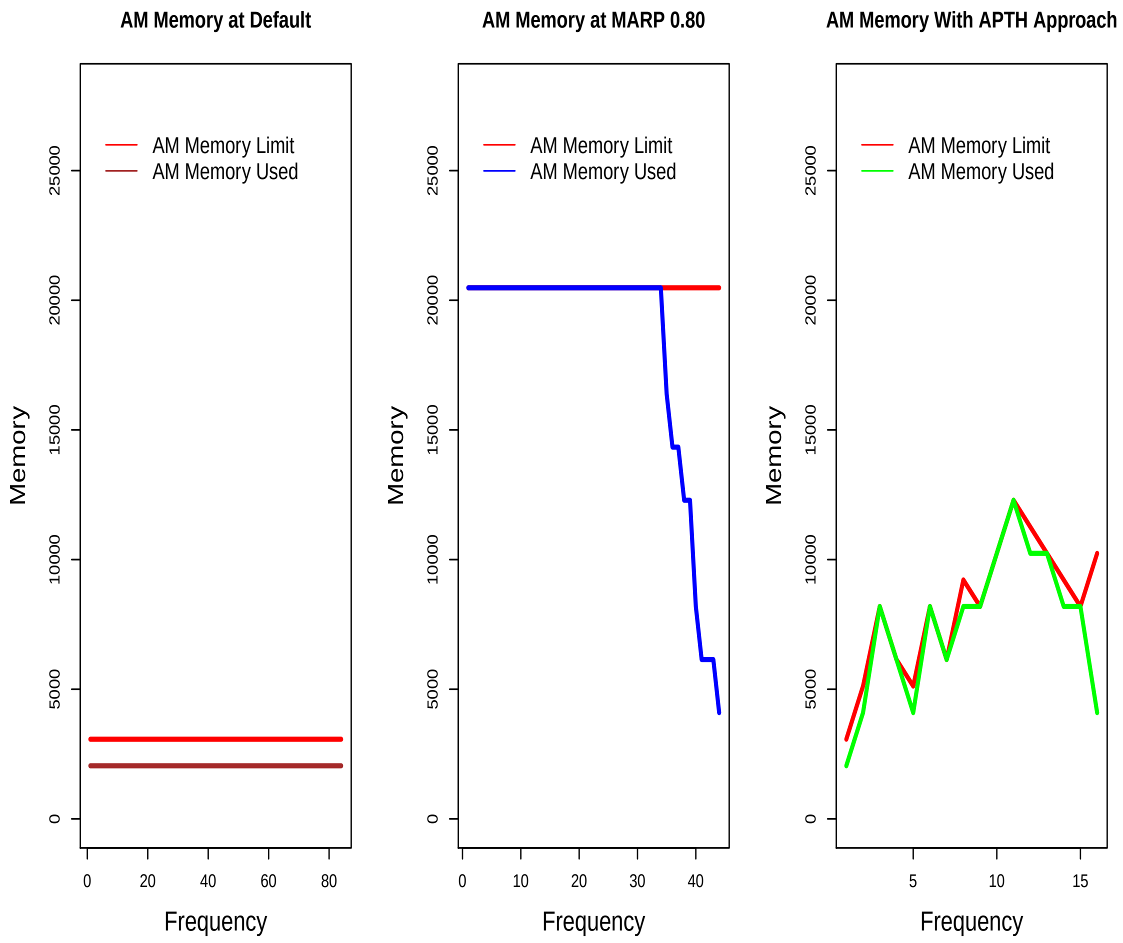


Figure 6.6: Memory consumption by ApplicationMaster(AM) during 30 job execution.

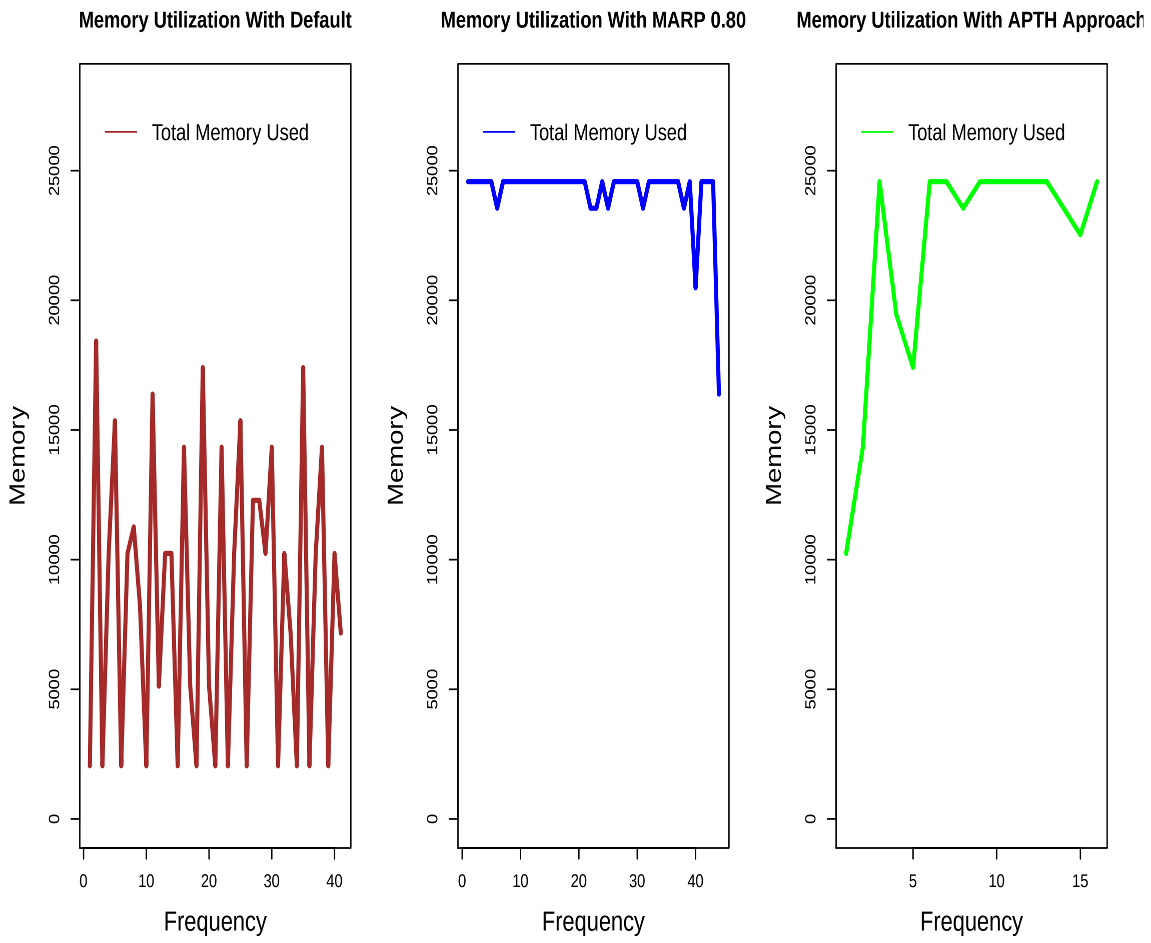


Figure 6.7: Overall Memory consumption during 30 job execution.

Table 6.2: Experiment-2 Summary Table

Cofigurations	Max job parallelism	Average throughput	Total time	Capacity Used
Default	1	151 bytes/sec	22728 sec	36.75%
MARP 0.80	10	180 bytes/sec	16639 sec	98.29%
APTH Approach	5	299 bytes/sec	13908 sec	89.58%

allocated for actual job execution container which results in less throughput. It can also be noticed the inverse proportional relation between throughput and time. Further, while time consumption is more then the throughput is less and vice versa. Not only time, overall memory resources also used was 10% more than APTH approach, even though total time of execution is comparatively more and throughput is comparatively less than APTH approach. At this point, here comes the meaning of *appropriate optimum* resource utilization which is stated in our problem statement. Finally, APTH approach gives the positive result and prove itself that it is the best among two other configurations in terms of performance tuning.

6.1.3 Experiment-3: Processing 45 Jobs

In the third experiment, 45 number of jobs were executed in a 3 different configurations like in experiment 1 and in experiment 2. The figure, **6.8 on the facing page** shows the job parallelism on different configurations. Similarly figure **6.9 on page 74** present memory resource threshold and resource used by ApplicationMaster. Correspondingly figure **6.10 on page 75** provides an overview of total memory resource used during experiment time. Moreover, table **6.3** provides the summarized form of result.

Table 6.3: Experiment-3 Summary Table

Configurations	Max job parallelism	Average throughput	Total time	Capacity Used
Default	1	75 bytes/sec	62916 sec	33.89%
MARP 0.50	6	141 bytes/sec	31492 sec	96.75%
APTH Approach	6	135 bytes/sec	33142 sec	91.58%

Here in the third experiment, the random configuration for MARP value was set to be 0.50 which means 50% resources allocated for ApplicationMaster. APTH approach has a far better result with 83% improvement comparing job parallelism with Default configuration. Similarly, APTH approach has better average throughput compared to the Default configuration. Not only throughput but also total resources during the experiment was appropriate optimum utilized. However, results of APTH approach comparing with MARP 0.50 configuration looks pretty close. The maximum number of the parallel running job was equal with configuration 0.50 and with APTH approach. Similarly, average throughput was increased with configuration MARP 0.50 because of the increase in total resources of the cluster get increased by 5%.

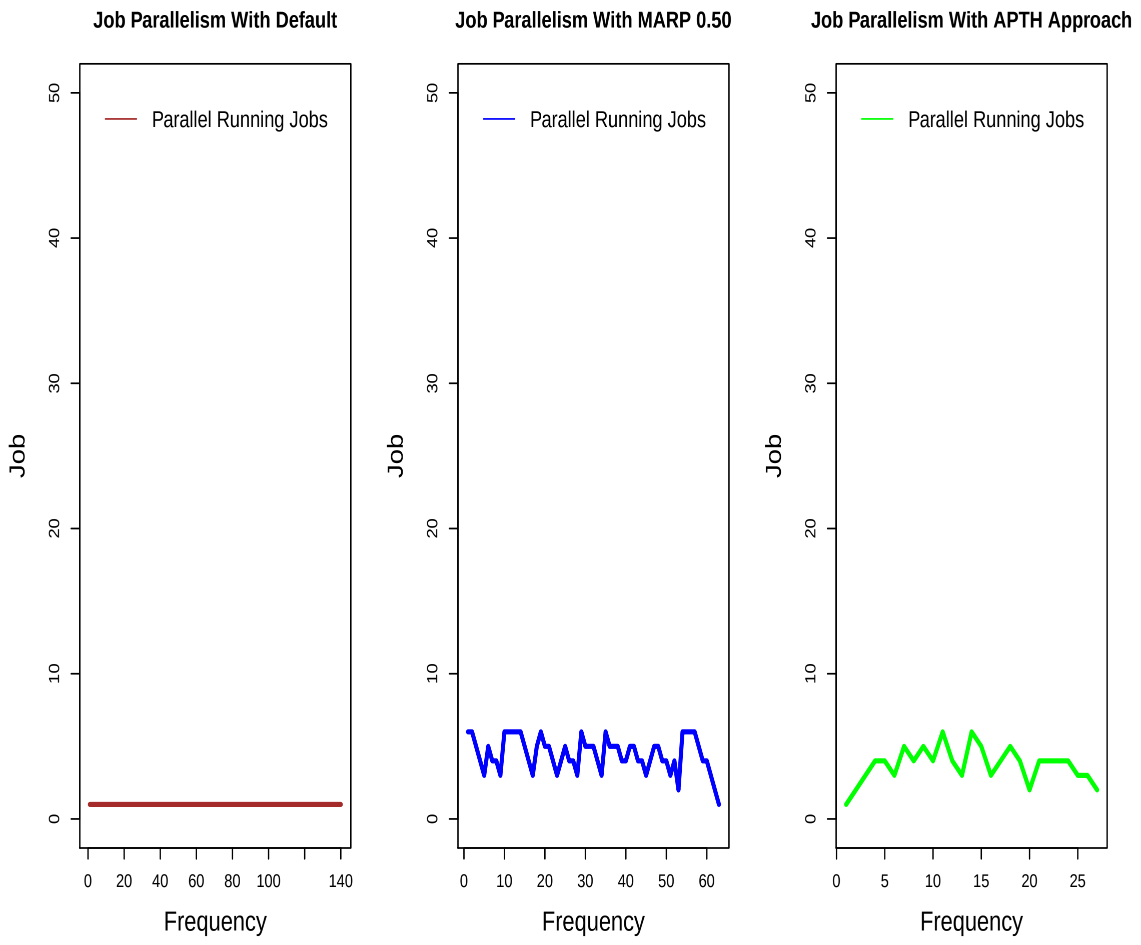


Figure 6.8: Parallel running number of jobs during 45 job execution.

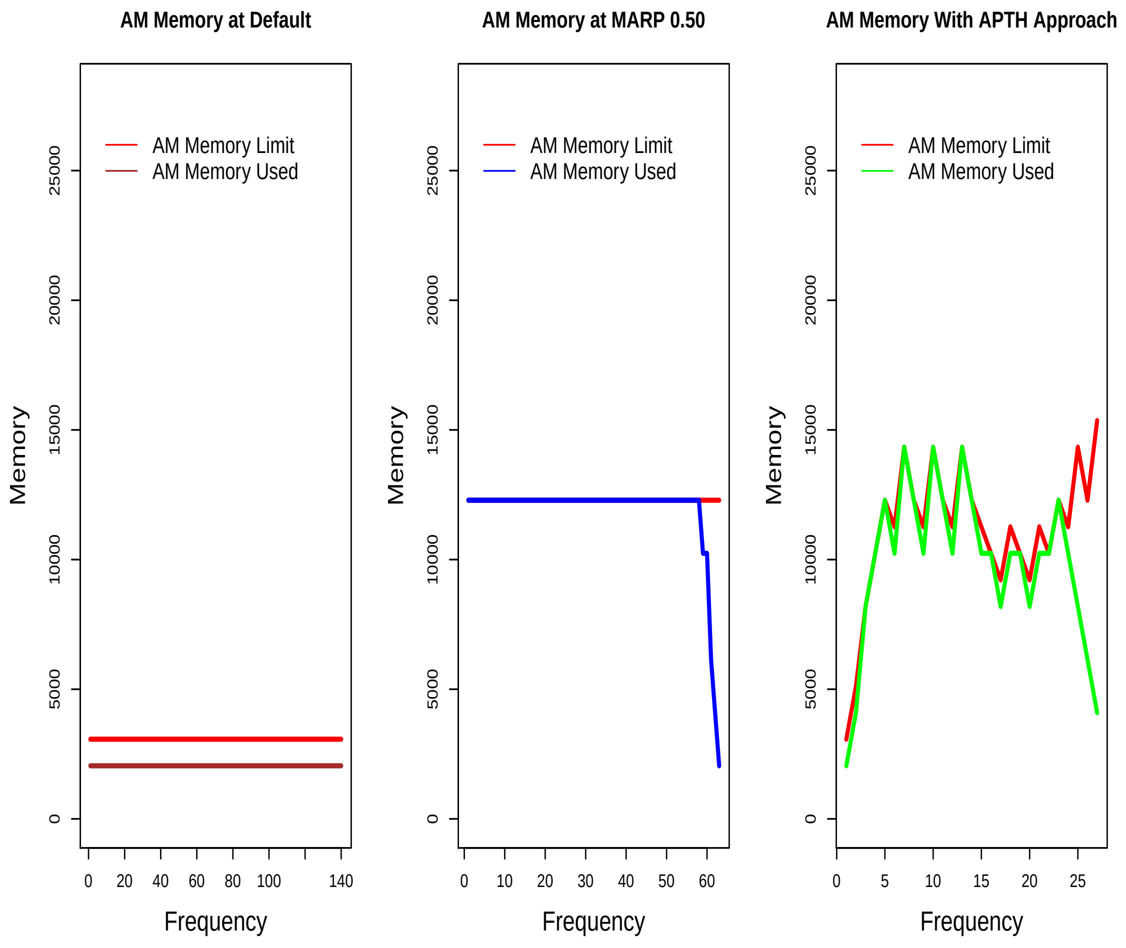


Figure 6.9: Memory consumption by ApplicationMaster(AM) during 45 job execution.

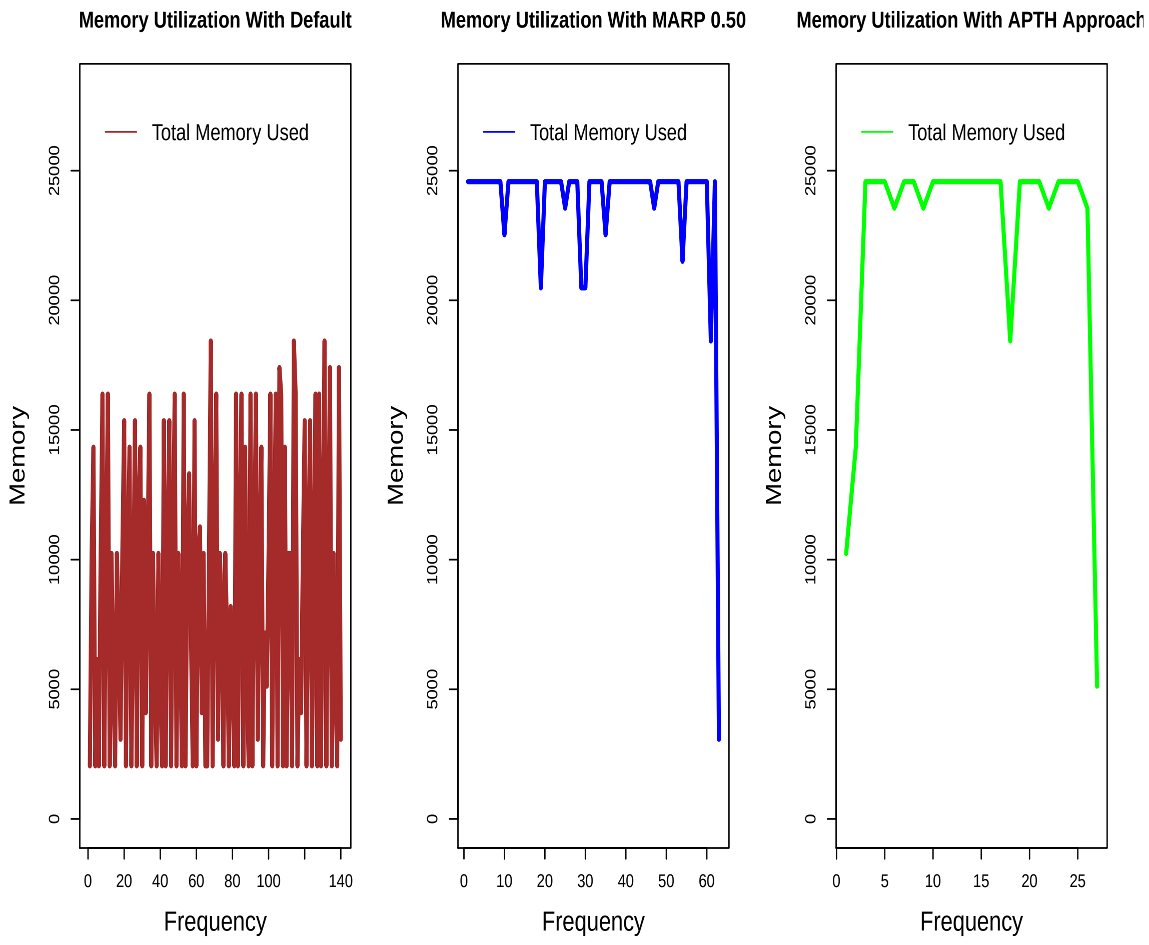


Figure 6.10: Overall Memory consumption during 45 job execution.

Finally, it doesn't unnecessarily means that APTH approach always should have better results even if static MARP value set by the system administrator in the correct form. On the other hand, there is also the possibility that APTH approachcan provide better results with respect to the Default configuration and static inappropriate configuration.

6.2 Data Analysis

This section is all about the analysis of data described in the experiments section. The result obtained with APTH approach was as expected (**see Section 4.2**) in terms of job parallelism and corresponding average throughput. Not only job parallelism and throughput but also appropriate optimum resources were utilized with APTH approach which helps reducing total execution time. Thus, increasing the performance. These are highlighted in the following subsections. The outcome of this analysis will help to determine to what extent the objective of this thesis have been met i.e., how the designed APTH approach which dynamically changes the MARP parameter at runtime that optimally and efficiently utilizes system resources in order to have a reliable process on big data workloads in the Hadoop cluster.

6.2.1 Interpreting Job Parallelism

From the results, it can be seen that as MARP parameter value get changed with designed APTH approach during the experiment, resources unit allocated for ApplicationMaster also gets changed. Thus, there is fluctuation in the number of parallel running jobs with APTH approach. However, experiments with rest of the static configurations show that because of unchanged in MARP parameter in the cluster there is always the same amount of resources unit allocated for ApplicationMaster. Thus, results in the same number of the job parallelism for the whole experiment.

The figure, **6.11 on the facing page** shows increased in the number of parallel running jobs as long as the resources allocation for ApplicationMaster is incremented. The x-axis of the figure represents memory in MB allocated for ApplicationMaster and the y-axis shows the corresponding number of parallel running jobs. Similarly figure, **6.13 on page 80** represent the change in memory allocation for application master with the change in MARP parameter. Notably, it can be concluded that MARP parameter has the linear correlation with job parallelism.

There is a linear correlation between MARP parameter and job parallelism. As parameter value gets increased the corresponding number of the job to be run in parallel also gets increased. Regarding job parallelism in every experiment, it shows that the trend is increasing in the number of jobs running in parallel with increment in MARP parameter value.

6.2.2 Interpreting Throughput

In some extent, it is true that gradually increased in the number of parallel running job provides better throughput. However, maximum throughput only can be

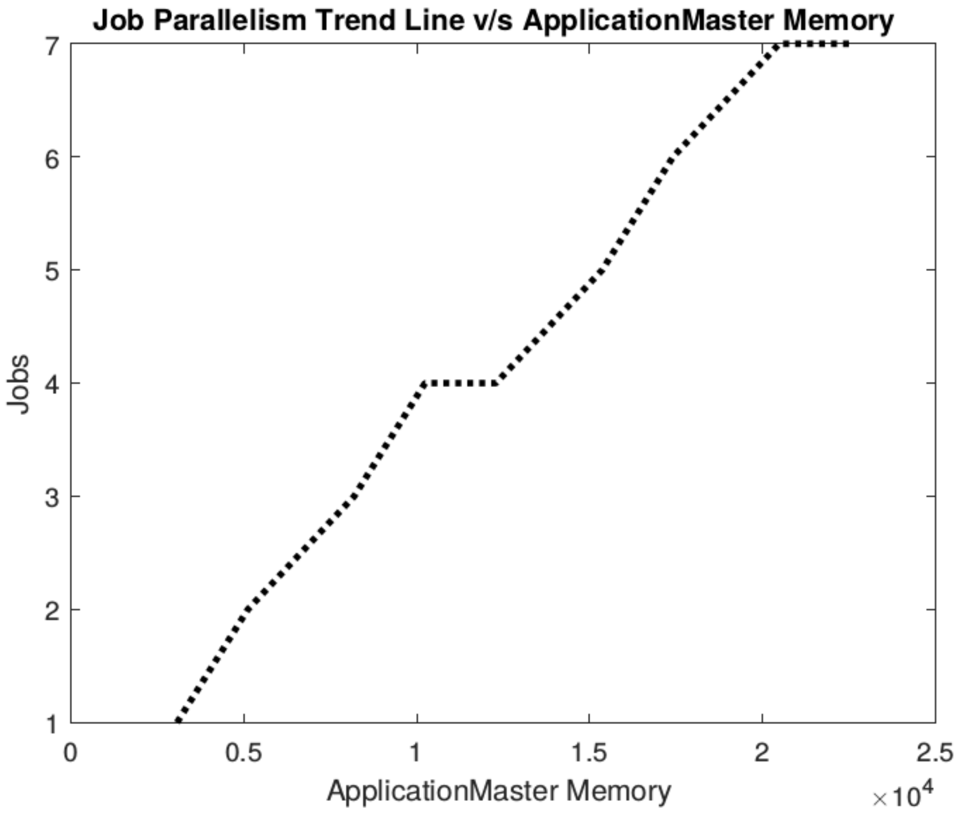


Figure 6.11: Increase in Job parallelism with increase in memory for Application-Master.

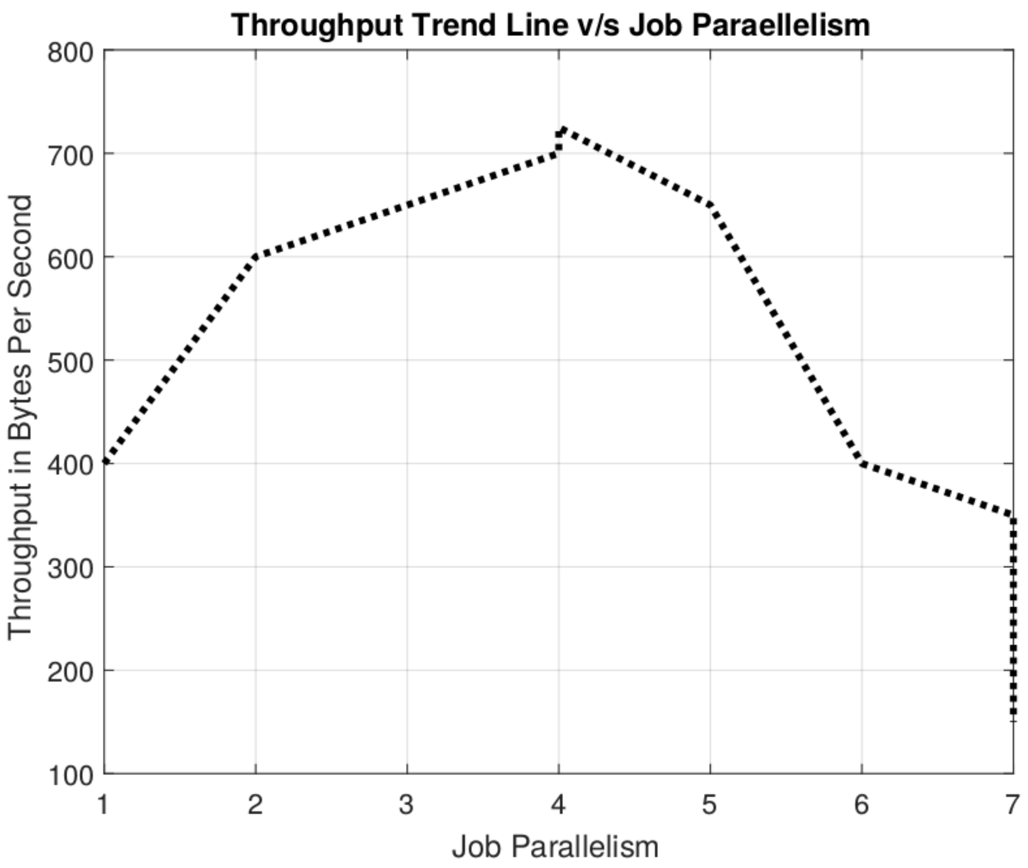


Figure 6.12: Throughput trend line with increase in job parallelism.

achieved when there is well balance in resources used by ApplicationMaster and YarnChild. Thus, till some point, continuous increase in the parallel running job can increase the throughput that balance appropriate resources for ApplicationMaster and YarnChild. Moreover, if the situation is that the running jobs should compete for the releasing resources from YarnChild then the throughput decreases. Results from the experiments show that APTH approach is able to keep the balance between numbers of parallel running jobs and their corresponding average throughput. In the first experiment, APTH approach was able to increase the throughput by 40% compared to the default configuration. Moreover, there was 50% and 45% of improvement with average throughput by APTH approach compared with the default configuration in experiment2 and experiment3 respectively.

In the experiment2 with random configuration, MARP 0.80 shows that the throughput is low compared to APTH approach. The reason for this was 80% of the resources was used by ApplicationMaster which results in running 10 number of job in parallel. Thus, actual job execution container YarnChild can only be used 20% of the resources which was not enough for the executing actual job that results in less throughput. Alternatively, APTH approach in experiment2 balances resources for jobs and throughput. Maximum 5 number of jobs are running in parallel and their average throughput was better.

The figure, **6.12 on the facing page** shows the dynamic nature of the throughput. Throughput is less for the small number of parallel running jobs because there might be the possibility of the increased in an ideal resource in the cluster. Similarly, throughput is less for the big number of parallel running jobs because many jobs are obligated to share same resources in the cluster. APTH approach in this project works in a way which always balances the number of parallel running jobs and the resources unit in the cluster that can provide better throughput.

6.2.3 Interpreting Resources Utilization

Appropriate optimum resource utilization was one of the main goal of this project. Results of the experiment show that APTH approach was able to leverage cluster resources efficiently. In the experiment1, APTH approach was able to use the cluster resources by 86% in average which was 52% increment compared to the default configuration. Similarly, compared with configuration MARP 0.20, APTH approach was still able to improve average resources utilized by 30%. Likewise, in the experiment2 and expriment3, there was an increment of 53% and 56% compared with the default configuration.

It is worth mentioning the resources used by ApplicationMaster at this point. For instance figure **6.3 on page 67**, **6.6 on page 70**, and **6.9 on page 74** from experiment1 and experiment2 and experiment3 respectively shows the amount of resources allocated for ApplicationMaster on different configurations by the red line and different line with colors brown, blue and green showing used resources by ApplicationMaster. Notably, within the static configuration ApplicationMaster cannot use all the resources allocated for it. This might be one reason why more jobs cannot be scheduled for MapReduce operation which results in overall less resource utilization. However, APTH approach significantly utilized almost all the allocated resource over time by changing MARP parameter. This results in more jobs to be scheduled for MapReduce operation. Thus, appropriate optimum resource utilization is achieved.

6.2.4 Performance Comparison

APTH approach reacted differently when applied to the different number of input jobs. In the first experiment, APTH approach spends 40% less time compared to the default configuration. Similarly, APTH approach spends 38% and 45% less time compared to the default configuration in experiment2 and experiment3 respectively. Hence, regarding the performance, APTH approach is able to tune the performance of the Hadoop cluster by 41%.

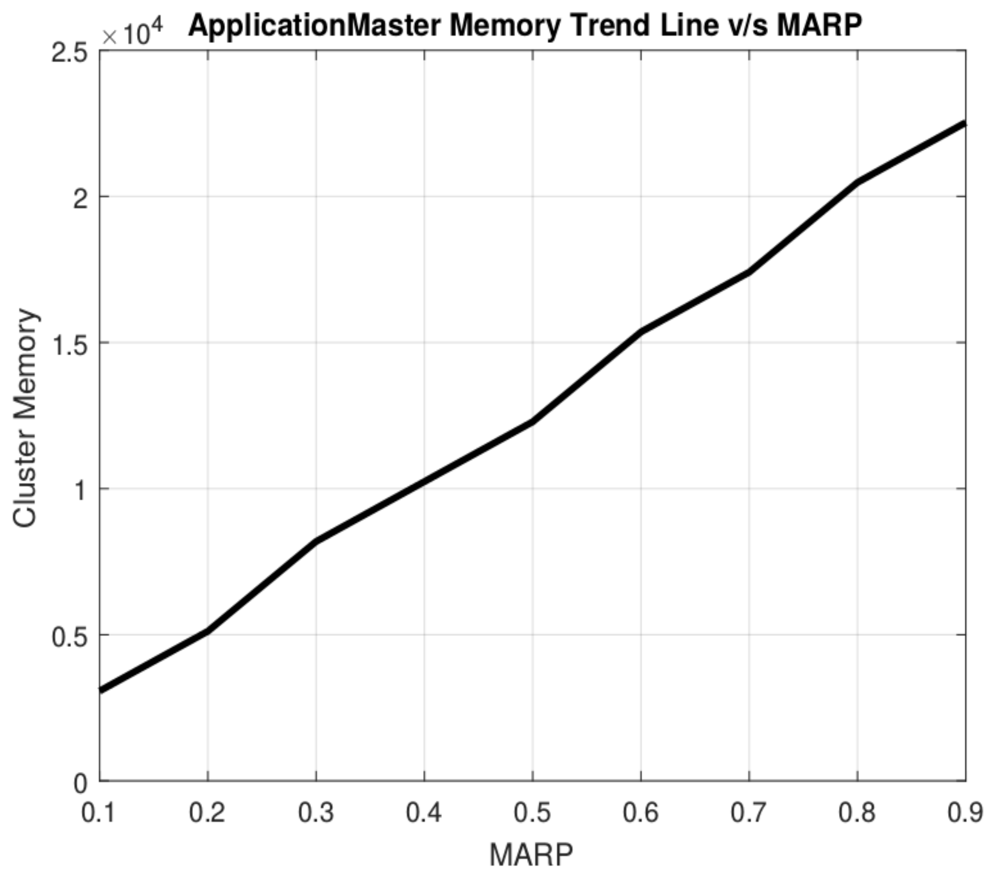


Figure 6.13: ApplicationMaster memory used Trendline.

Chapter 7

Discussion

This chapter gives a reflection on the results obtained in the course of this project work. Implementation steps, facts, and challenges along with suggestions for improvement of the deployed APTH approach are discussed in this chapter.

7.1 Implementation of APTH Approach Design

The goal of this thesis was to explore a new and efficient architecture which can increase the number of parallel running jobs and increase their corresponding throughput by dynamic allocation of resources at runtime in the Hadoop cluster. This was achieved through optimal architectural design that frequently calculates the progress rate of all the running MapReduce jobs in the Hadoop cluster. This design can be served as one of the adaptive performance tuning approaches. Moreover, based on the design an optimal APTH approach was developed, experimented upon to test and analyze job parallelism and throughput.

The results of the experiments were as expected, and the outcome of the analysis showed that design and developed APTH approach was better than the inappropriate static configuration and far better than the default configuration. Moreover, mainly two problem Loss of Input Job (LOIJ) and Loss of Task Throughput (LOTT) (**see Section 3.1.1**) were able to address by APTH approach, results in *appropriate optimum resources utilization*. From the results of the experiments, also it was clear that static configuration is not able to utilize full resources that were allocated to the ApplicationMaster. As a result, it affects the total resources utilization in the cluster during runtime. Therefore, performance degrades with the static configuration. However, our designed and developed APTH approach showed better performance regarding resources utilization allocated for ApplicationMaster. In addition, most of the time, APTH approach was able to utilize 100% resources that were allocated for ApplicationMaster that resulted in increase in job parallelism. Not only allocated 100% resources utilized by ApplicationMaster but also most of the time 100% resources of the whole cluster were utilized during MapReduce operation. Thus, actual job execution container YarnChild get more resources to execute the job which resulted in the increase in throughput.

It was observed that as the MARP parameter value gets changed, the number of parallel running jobs also get changed. Thus, the observations can be generalized in order to support the anticipated linear correlation of MARP parameter value and job parallelism. Furthermore, the observations also made that resources used by ApplicationMaster exibit a linear correlation with the total amount of resources used in the whole cluster.

7.1.1 Project Outcome

In the problem statement, three key requirements were established for progress aware dynamic APTH approach design:

- Increase in job parallelism

Job parallelism is achieved with the designed APTH approach.

- Increase in throughput

Increased in average throughput is achieved with the APTH approach.

- Appropriate optimum resource utilization

Resources unit of the cluster were utilized in an efficient manner which leads reduction on ideal resources in the cluster, results increased in job parallelism and increased in corresponding throughput. It can be concluded that the key requirements of the problem statement are addressed with the design because of the vital role played by MARP parameter value which was dynamically changed and reconfigured during the MapReduce operation in the cluster.

7.2 Implementation Challenges

Many challenges were faced during the development process of the project work. These challenges are outlined in the following subsections.

7.2.1 Setting Sleep Time for the Algorithm Deployed

Defining and setting the sleep time for the algorithm was one of the challenging tasks during the development phase. In fact, higher sleep time increases the risk that small jobs may not be accountable for progress value to be calculated over time. Small size jobs may be started and finished between the defined time interval which will not be counted by the algorithm in order to calculate accumulated progress value. Moreover, this is the situation where the algorithm can take wrong decisions about the action to be taken. However, very small sleep time interval to be set for the algorithm also affects for big size jobs which take some time in order to get prepared itself. When the algorithm is supposed to calculate progress value it might not cover for the big size jobs. In order to get progress values of all the submitted jobs by the algorithm, 15 seconds was chosen as a suitable sleep time.

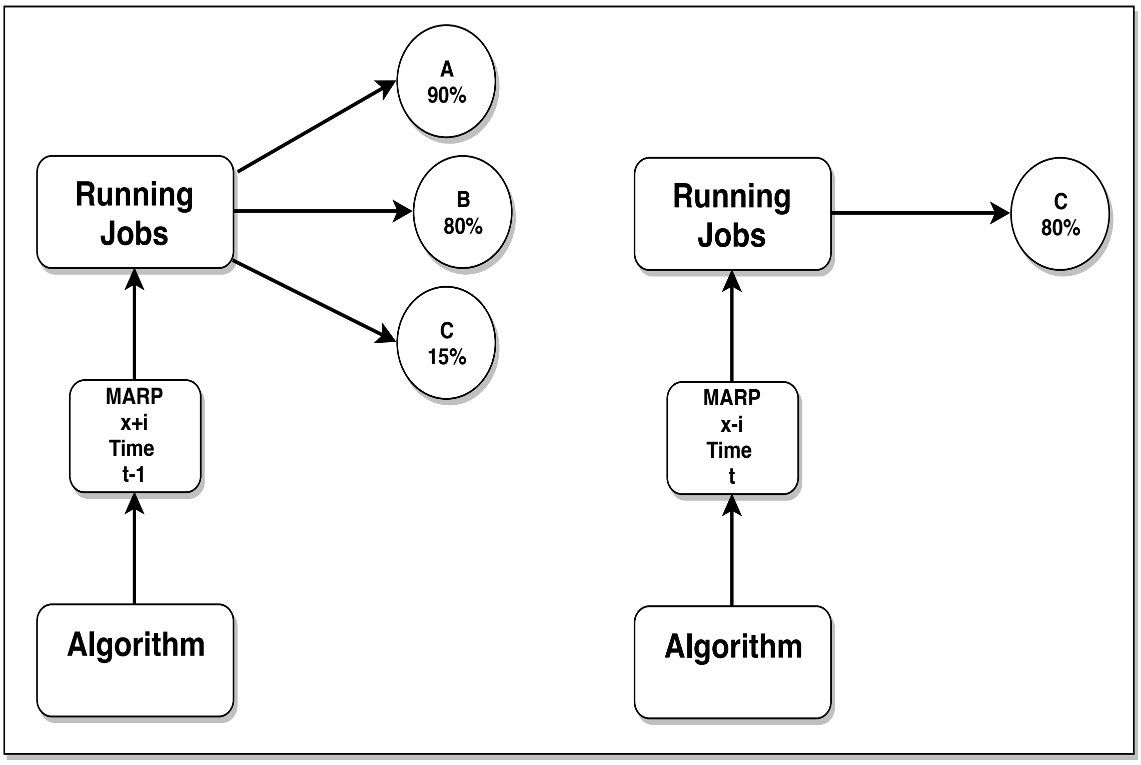


Figure 7.1: Effects on job parallelism with same % of resources increment and decrement.

7.2.2 Defining % of Resources Allocation for the Action

One of the most challenging tasks was to define the percentage of resources to be incremented or decremented based on Δprogress. Moreover, allocation of the same percentage for both of the actions doesn't provide better results in order to achieve the goal of the project. Figure, **7.1** provides better understanding regarding the same % of resources allocation for increment and decrement actions. Multiple experiments with different resources % for the allocation by the algorithm was carried out by running 20 jobs. Table **7.1 on page 85** presents the summary of the experiment carried out. Small % of resource increment limits in job parallelism. Alternatively, big % of resource decrement also limits in job parallelism. Similarly, big % of the resource increment can give job parallelism as the major portion of the resources of the cluster can be utilized by ApplicationMaster, meanwhile, due to fewer resources available for the actual job execution container, it can be realized less throughput which gradually degrades the performance. In the same way, big % of the resource decrement can make more spaces for actual job execution container which leads for better throughput. On the other hand, job parallelism cannot be achieved due to fewer resources allocation for ApplicationMaster.

In this work, we decided not to take action increment or decrement with equal % of resources allocation. Moreover, incrementing % of the resources will be always greater than the decrement % of the resources which still keeps fewer % of

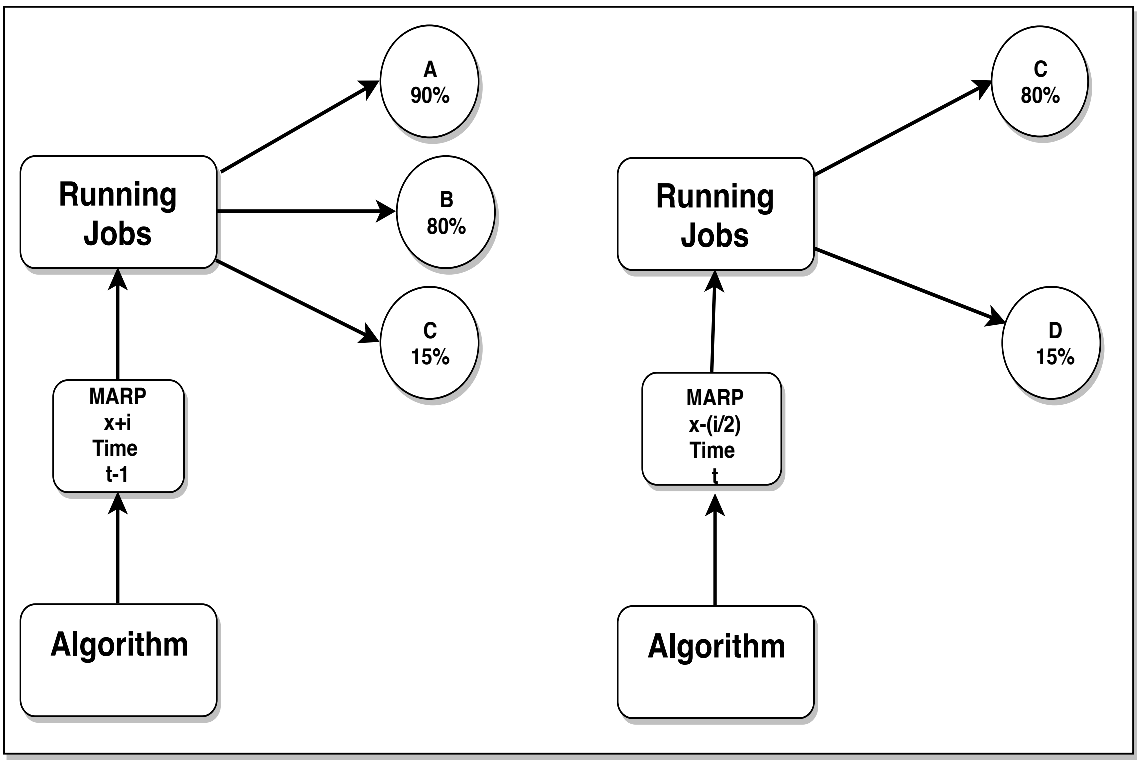


Figure 7.2: Effects on job parallelism with different % of resources for actions.

resources available for job parallelism. Figure, **7.2** presents resources % for increment action is greater than decrement action which still provides some resources for ApplicationMaster.

For instance, 10% of the resources to be added to the cluster by incrementing MARP parameter can give positive results for job parallelism. However, if the resources allocation in the cluster is decreased by same % and if MapReduce operation for the running jobs gets finished at the same time then the parallel running number of jobs get cut off by two aspects, first one by spontaneous finishing of running MapReduce operation and second one by decrementing resources allocation. Therefore, it results in ideal resources and less throughput which leads to degradation on overall performance. In order to maintain above explained situation, the author of this thesis, set resources to be incremented by 10% and decremented by 5% which keeps balance on job parallelism and throughput.

7.2.3 Progress v/s Δprogress

Calculating only progress value and taking action based on that progress value is not efficient. While comparing current progress value with previous than current progress always is greater than previous progress. The point is that action to be

Table 7.1: Action with Different Resources Allocation

Inc	Decr	Job Parallelism	Average Throughput	Time	Used Capacity
5%	2.5%	3	505 bytes/sec	5476 Sec	72%
5%	5%	3	457 bytes/sec	6081 Sec	52%
10%	**5%**	**5**	**487 bytes/sec**	**5723 Sec**	**86%**
10%	10%	4	502 bytes/sec	5852 Sec	79%
20%	10%	5	420 bytes/sec	5255 Sec	94%
20%	20%	6	390 bytes/sec	5690 Sec	89%

taken will be always same. However, the concept of Δprogress provides the speed of work done over time. Based on comparison, the speed decision can be made. Moreover, there is no chance that the current speed will be always high or always low. For instance, when the current speed is high (current Δprogress) increment action taken by APTH approach, results in increase in job parallelism. When many jobs used resources for ApplicationMaster then the static resources in the cluster will not have enough spaces for YarnChild (actual job execution container) which results in comparatively less throughput (progress value). In addition, when APTH approach find current progress is less then the action will be switch accordingly.

7.2.4 Programming Complexity

Writing code for the functionality of the APTH approach was equally challenging. The project was created from the scratch and was facilitated over 500 lines of the bash code. The code in order to make dynamic changes on the same value over time and again and reconfigured along the whole cluster, calculating accumulated progress, calculating Δprogress took lots of trial-and-error leading to lots of debugging during deployment period. Many functions were developed to reduce the same code to be reused. Furthermore, many technical problems were addressed using efficient logic inside the code.

7.2.5 Creating Multiple Concurrent Input Jobs

Bash script was developed in order to submit multiple jobs at the same time. Generally, HiBench benchmark suite used to execute jobs in sequence, new job get chance to be scheduled only when previous job release resources. Multiple numbers of jobs were created using the script repeating the same job for same input data which was then implemented on MapReduce operation. Therefore, many numbers of jobs can be submitted concurrently and can be achieved job parallelism and corresponding throughput.

7.3 Improvements To APTH Design

In this section potential improvements which could be made to the APTH approach is presented.

7.3.1 APTH Adoption with Dynamic Change in the Cluster Resources

The designed and developed approach is adopted in order to address the problem statement question. However, APTH approach could be better by adding new features and modifying existing one. The designed approach is developed in a way which can work only with predefined resources in the Hadoop cluster. Therefore, APTH approach is compelled to play with the static number of resources in the Hadoop cluster. Alternatively, APTH approach can be modified in a way which can adopt the dynamic change in the resource unit of the Hadoop cluster. Moreover, the addition of cloud bursting feature in developed approach can provide the best results in case of job parallelism.

APTH approach also can have the improvement with action to be taken as increment or decrement based on feedback with job parallelism. Moreover, progress value is a kind of indirect form of throughput. Currently, every action that APTH approach takes is based on the progress value. But it can be more logical if both increment and decrement factor can be taken as feedback from the cluster and maintain the action as reward and penalty. For instance, initially, APTH approach can check the progress rate over time and take action accordingly. Afterwhile, APTH approach can check job parallelism and identify if the result is positive or negative.

Chapter 8

Conclusion

The aim of this project was to investigate an optimal way that can reduce the numbers of ideal resources in the Hadoop cluster. This is achieved by scheduling the numbers of pending jobs to be run in parallel that makes an appropriate and optimum resources utilization during MapReduce operation.

Job parallelism, throughput, and appropriate resources utilization as the key elements of the problem statements were addressed through the development of *Adaptive Parameter Tuning Of Hadoop (APTH)* approach. In addition, APTH approach consists of Δprogress aware algorithm that can dynamically change the resources allocation to the ApplicationMaster in the cluster during the run-time. The developed algorithm was thoroughly experimented in order to obtain job parallelism, throughput, and resource utilization measurements as compared to static configuration.

Results from the experiments show that the APTH approach facilitated appropriate and optimum resources utilization which in turn increased in job parallelism and corresponding throughput compared to the default configuration. Likewise, the total time taken by the APTH approach was found out to be 30% less than the total time taken by the default configuration. Hence, it is evident from our experimental analysis that APTH approach is able to tune the MapReduce operation performance by 30%.

Keeping in mind that the high demand for the real-time streaming platform these days, this project contributes to the landscape of real-time data processing in Hadoop via on-run time parameter tuning.

8.1 Future Work

There are many more aspects within this project which can be extended. The designed and developed algorithm explore a novel way of driving a balance between the job parallelism and their associated throughput by having an appropriate and optimal resources utilization in the Hadoop cluster. One of the important thing that can be done with this project is achieving highly available Hadoop cluster design.

In such high availability cluster, two separate machines can be configured as Namenodes. At any point in time, exactly one of the Namenode can be configured as in a *active* state and other by in a *standby* state. The active Namenode will be responsible for all client operations in the cluster, while the standby will be simply acting as a slave, maintaining enough state to provide a fast failover if necessary [1].

It can be interesting to use machine learning approach in order to increment or decrement action with MARP parameter on the fly. Machine learning can be used to classify based on pattern of the job whether it is memory bound or CPU bound. In addition, if the majority portion of the parallel running job is memory bound then the approach can calculate tentative necessary additional resources for ApplicationMaster in the cluster and increased the MARP parameter value automatically.

[1] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html

Bibliography

[1] Nrusimham Ammu and Mohd Irfanuddin. 'Big data challenges'. In: *International Journal of Advanced Trends in Computer Science and Engineering* 2.1 (2013), pp. 613–615.

[2] *bader-papers*. https://www.cc.gatech.edu/~bader/papers. Accessed: 2018-03-12.

[3] *Big-Data-Cloud-Database-and-Computing*. https://www.qubole.com/resources/big-data-cloud-database-and-computing/. Accessed: 2018-02-03.

[4] Dhruba Borthakur. 'The hadoop distributed file system: Architecture and design'. In: *Hadoop Project Website* 11.2007 (2007), p. 21.

[5] Parth Chandarana and M Vijayalakshmi. 'Big data analytics frameworks'. In: *Circuits, Systems, Communication and Information Technology Applications (CSCITA), 2014 international conference on*. IEEE. 2014, pp. 430–434.

[6] *Cloud SDK*. https://cloud.google.com/sdk/. Accessed: 2018-03-21.

[7] *computer-cluster*. https://www.techopedia.com/definition/6581/computer-cluster. Accessed: 2018-03-12.

[8] Xiaoan Ding, Yi Liu and Depei Qian. 'Jellyfish: Online performance tuning with adaptive configuration and elastic container in hadoop yarn'. In: *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. IEEE. 2015, pp. 831–836.

[9] Jens Dittrich and Jorge-Arnulfo Quiané-Ruiz. 'Efficient big data processing in Hadoop MapReduce'. In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 2014–2015.

[10] Nada Elgendy and Ahmed Elragal. 'Big data analytics: a literature review paper'. In: *Industrial Conference on Data Mining*. Springer. 2014, pp. 214–227.

[11] Peter Géczy. 'Big data characteristics'. In: *The Macrotheme Review* 3.6 (2014), pp. 94–104.

[12] *Google Cloud Compute Engine*. https://cloud.google.com/compute/. Accessed: 2018-03-16.

[13] *Google Cloud Compute Engine*. https://cloud.google.com/compute/. Accessed: 2018-03-16.

[14] *Google Cloud Compute Engine Documents*. https://cloud.google.com/compute/docs/. Accessed: 2018-03-16.

[15] *Google Cloud Overview*. https://cloud.google.com/docs/overview/. Accessed: 2018-03-15.

[16] *Google Compute Engine*. http://searchaws.techtarget.com/definition/Google-Compute-Engine. Accessed: 2018-03-16.

[17] *hadoop-cluster*. https://www.techtarget.com. Accessed: 2018-03-12.

[18] Dominique Heger. 'Hadoop performance tuning-a pragmatic & iterative approach'. In: *CMG Journal* 4 (2013), pp. 97–113.

[19] Shengsheng Huang et al. 'Hibench: A representative and comprehensive hadoop benchmark suite'. In: *Proc. ICDE Workshops*. 2010.

[20] *Java Developmemt Kit JDK*. https://www.techopedia.com/definition/5594/java-development-kit-jdk. Accessed: 2018-03-21.

[21] Zahid Javed et al. 'Big Data and Hadoop'. In: ().

[22] Shrinivas B Joshi. 'Apache hadoop performance-tuning methodologies and best practices'. In: *Proceedings of the 3rd acm/spec international conference on performance engineering*. ACM. 2012, pp. 241–242.

[23] Amogh Pramod Kulkarni and Mahesh Khandewal. 'Survey on Hadoop and Introduction to YARN'. In: *International Journal of Emerging Technology and Advanced Engineering* 4.5 (2014), pp. 82–87.

[24] Gil Jae Lee and José AB Fortes. 'Hierarchical Self-Tuning of Concurrency and Resource Units in Data-Analytics Frameworks'. In: *Autonomic Computing (ICAC), 2017 IEEE International Conference on*. IEEE. 2017, pp. 49–58.

[25] Min Li et al. 'Mronline: Mapreduce online performance tuning'. In: *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM. 2014, pp. 165–176.

[26] James Manyika et al. 'Big data: The next frontier for innovation, competition, and productivity'. In: (2011).

[27] Behbood Mashoufi et al. 'Introducing an adaptive VLR algorithm using learning automata for multilayer perceptron'. In: *IEICE TRANSACTIONS on Information and Systems* 86.3 (2003), pp. 594–609.

[28] Seref Sagiroglu and Duygu Sinanc. 'Big data: A review'. In: *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. IEEE. 2013, pp. 42–47.

[29] Domenico Talia. 'Clouds for scalable big data analytics'. In: *Computer* 46.5 (2013), pp. 98–101.

[30] Kabin Tamrakar, Anis Yazidi and Hårek Haugerud. 'Cost Efficient Batch Processing in Amazon Cloud with Deadline Awareness'. In: *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*. IEEE. 2017, pp. 963–971.

[31] Ronald C Taylor. 'An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics'. In: *BMC bioinformatics*. Vol. 11. 12. BioMed Central. 2010, S1.

[32] *Top Big Data Processing Framework*. https://www.kdnuggets.com/2016/03/top-big-data-processing-frameworks.html. Accessed: 2018-03-03.

[33] *what-is-meant-by-streaming-data-access-in-hdfs*. https://serverfault.com/questions/40370/what-is-meant-by-streaming-data-access-in-hdfs. Accessed: 2018-03-03.

[34] Dili Wu and Aniruddha Gokhale. 'A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration'. In: *High Performance Computing (HiPC), 2013 20th International Conference on*. IEEE. 2013, pp. 89–98.

[35] *Yarn-application-master-in-hadoop*. http://www.dummies.com/programming/big-data/hadoop/yarns-application-master-in-hadoop/. Accessed: 2018-03-03.

[36] Bo Zhang. 'Self-optimization of Infrastructure and Platform Resources in Cloud Computing'. PhD thesis. Lille1, 2016.

[37] Bo Zhang et al. 'Hadoop-benchmark: rapid prototyping and evaluation of self-adaptive behaviors in Hadoop clusters'. In: *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press. 2017, pp. 175–181.

Chapter 9

Appendix

```
#!/bin/bash
#### GLOBAL VARIABLES
        toadd=0.10
        tosub=0.05
        maxmarp=0.90
        minmarp=0.10
        marpthirty=0.30
        default_value=0

while true; do
#### FUNCTION TO INCREMENT MARP VALUE
increment () {
       newmarp=$(echo $marp $toadd | awk '{printf "%0.2f", $1 + $2}')
#       echo "marp to submit is" $newmarp
       awk 'NR==26{$2=a}1' a=$newmarp $file > tmp && sudo mv -f tmp $file
       yarn rmadmin -refreshQueues
        echo "MARP Increment by 0.10 and the new MARP is " $newmarp
        }

#increment

#### FUNCTION TO DECREMENT MARP VALUE
decrement () {
        newmarp=$(echo $marp $tosub | awk '{printf "%0.2f", $1 - $2}')
#        echo "marp to submit is" $newmarp
        awk 'NR==26{$2=a}1' a=$newmarp $file > tmp && sudo mv -f tmp $file
        yarn rmadmin -refreshQueues
        echo "MARP Decrement by 0.05 and the new MARP is" $newmarp
        }

#decrement

#### FUNCTION TO ALLOCATE 40% RESOURCES TO APPLICATION MASTER
marp_thirty () {
```

```
awk 'NR==26{$2=a}1' a=$marpthirty $file > tmp && sudo mv -f tmp $file
yarn rmadmin -refreshQueues
echo "MARP set to" $marpthirty
}



#### FUNCTION WHICH FETCH METRICES FROM RESOURCE MANAGER
fetch_metrics () {

        #### TOTAL MEMORY IN A CLUSTER
        tot_mem=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $18}' | awk -F',' '{print $1}')

        #### MEMORY USED IN THE CLUSTER DURING JOB EXECUTION
        mem_used=$( curl http://project-master-01:8088/ws/v1/
cluster/scheduler | awk -F':' '{print $22}' | cut -d',' -f1)

        #### UNUSED MEMORY IN THE CLUSTER
        mem_unused=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $10}' | awk -F',' '{print $1}')

        ####TOTAL MEMORY ALLOCATED BY MARP VALUE
        marp_limit=$( curl http://project-master-01:8088/ws/v1/
cluster/scheduler | awk -F':' '{print $55}' | cut -d',' -f1)

        #### TOTAL MEMORY USED BY APPLICATION MASTER
        am_mem_used=$(curl http://project-master-01:8088/ws/v1/
cluster/scheduler | awk -F':' '{print $52}' | cut -d',' -f1)

        #### TOTAL VCORES USED BY APPLICATION MASTER
        am_vcore_used=$(curl http://project-master-01:8088/ws/v1/
cluster/scheduler | awk -F':' '{print $53}' |
cut -d',' -f1 | awk -F'}' '{print $1}')

        ####TOTAL NUMBER OF VIRTUAL CORE IN THE CLUSTER
        tot_core=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $19}' | awk -F',' '{print $1}')

        ####USED NUMBER OF CORE DURING JOB EXECUTION
        core_used=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $14}' | awk -F',' '{print $1}')

        ####UNUSED VIRTUAL CORE
        core_unused=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $13}' | awk -F',' '{print $1}')

        ####NUMBER OF APPLICATION RUNNING IN THE CLUSTER
        app_running=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $6}' | awk -F',' '{print $1}')

        ####NUMBER OF APPLICATION PENDING IN THE CLUSTER
        app_pending=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $5}' | awk -F',' '{print $1}')
        echo "$app_pending" > pending_app.txt
        ####CONTAINER RUNNING
        cont_running=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $15}' | awk -F',' '{print $1}')

        ####CONTAINER PENDING
        cont_pending=$(curl http://project-master-01:8088/ws/v1/
cluster/metrics | awk -F':' '{print $17}' | awk -F',' '{print $1}')

        ####TOTAL CAPACITY USED IN THE CLUSTER
        capacity_used=$(curl http://project-master-01:8088/ws/v1/
cluster/scheduler | awk -F':' '{print $6}' | cut -d',' -f1)
                }

#### FUNCTION TO CALCULATE PROGRESS FOR 10% RESOURCE ALLOCATION
progress_first () {

app_id=$(yarn application -list | grep "root" | grep "RUNNING" | awk '{print $1}')
value_initially=$(yarn application -list | grep "root" |
grep "RUNNING" | awk '{print $8 $9}'| cut -d '%' -f1 |
awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')

while [[ "${#app_id[@]}" != "${#value_initially[@]}" ]]; do
        app_id=()
        value_initially=()
        app_id=$(yarn application -list | grep "root" |
 grep "RUNNING" | awk '{print $1}')
        value_initially=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $8 $9}'| cut -d '%' -f1 |
awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')
done

echo "$app_id" > progress_first_only_app_id.txt
echo "$value_initially" > progress_first_only_value.txt

        pre_id_only=($(paste <(echo "$app_id")))
        echo "PREVIOUS APP ONLY ID"
        echo "${pre_id_only[@]}"

        pre_value_only=($(paste <(echo "$value_initially")))
        echo "PREVIOUS APP ONLY VALUE"
        echo "${pre_value_only[@]}"


        pre_id_value=$(paste <(echo "$app_id") <(echo "$value_initially"))
        echo "PREVIOUS APP ID AND CORRESPONDING VALUES"
        echo "$pre_id_value"

        progress_initial=0
        for i in ${pre_value_only[@]}
            do
             progress_initial=`echo $progress_initial + $i | bc`
        done

        echo $progress_initial > progress1.txt
        echo "The initial progress is" $progress_initial
        }



#### FUNCTION TO CALCULATE PROGRESS FOR 20% RESOURCE ALLOCATION
progress_second () {
app_id_current=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $1}')
value_current=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $8 $9}'|
cut -d '%' -f1 | awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')
while [[ "${#app_id_current[@]}" != "${#value_current[@]}" ]]; do
        app_id_current=()
        value_current=()
        app_id=$(yarn application -list | grep "root" |
 grep "RUNNING" | awk '{print $1}')
        value_initially=$(yarn application -list |
grep "root" | grep "RUNNING" | awk '{print $8 $9}'|
cut -d '%' -f1 | awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')
done
echo "$app_id_current" > progress_second_only_app_id.txt
echo "$value_current" > progress_second_only_value.txt

app_id=$(sudo cat /home/ramesh/progress_first_only_app_id.txt)
value_initially=$(sudo cat /home/ramesh/progress_first_only_value.txt)
```

```
curr_id_only=($(paste <(echo "$app_id_current")))
echo "CURRENT APP ONLY ID"
echo "${curr_id_only[@]}"

curr_value_only=($(paste <(echo "$value_current")))
echo "CURRENT APP ONLY VALUE"
echo "${curr_value_only[@]}"

curr_id_value=$(paste <(echo "$app_id_current") <(echo "$value_current"))
echo "CURRENT APP ID AND CORRESPONDING VALUES"
echo "$curr_id_value"

##
pre_id_only=($(paste <(echo "$app_id")))
echo "PREVIOUS APP ONLY ID"
echo "${pre_id_only[@]}"

pre_value_only=($(paste <(echo "$value_initially")))
echo "PREVIOUS APP ONLY VALUE"
echo "${pre_value_only[@]}"


##

#different=$(diff <( echo "$app_id" ) <( echo "$app_id_current" ))
different=$(diff -ia --suppress-common-lines
<( printf "%s\n" "${app_id[@]}" ) <( printf "%s\n" "${app_id_current[@]}"))
echo "DIFFERENCE BETWEEN THE APPLICATION
IDS IN THE CURRENT STATE, WHETHER LOST OR ADDED ARE"
#echo ${different[@]}

for i in ${different[@]}
do
echo $i | grep "application_" | awk '{print $1}' >> app_id_changed.txt
done
fetch_file_data=$(sudo cat /home/ramesh/app_id_changed.txt)
echo ${fetch_file_data[@]}

sudo truncate -s 0 app_id_changed.txt


#######################################
#intersection_with_current
######################################
for item1 in ${app_id_current[@]}
 do
    for item2 in ${fetch_file_data[@]}
```

```
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_current+=( "$item1" )
        fi
    done
done
echo "FOLLOWING APPS ARE NEWELY ADDED"
echo ${intersection_with_current[@]}
#############################################
#TO ADD THE VALUES OF THE NEWELY ADDED JOB
#############################################

for ((i=0; i < ${#curr_id_only[@]}; ++i))
   do
        for j in "${intersection_with_current[@]}"
          do
                if [[ "${curr_id_only[$i]}" == "$j" ]]
                then

                        index_arr_curr+=( "$i" )
                fi
        done
  done

echo "The list of the index for newely added jobs are"
echo ${index_arr_curr[@]}

########################

for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_arr_curr[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_newly_added+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding are"
echo ${sum_newly_added[@]}

##########################
sum1=0
for i in ${sum_newly_added[@]}
 do
```

```
        sum1=`echo $sum1 + $i | bc`
 done
echo "The total sum of the currently added job progress is" $sum1


##########################
#intersection_with_previous
##########################
for item1 in ${app_id[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_previous+=( "$item1" )
        fi
    done
done


echo "FOLLOWING APPS WERE IN PREVIOUS BUT NOT IN CURRENT"
echo ${intersection_with_previous[@]}

###########################
#  ADD THE VALUES SUBTRACTIONG FROM 100
###########################

for ((i=0; i < ${#pre_id_only[@]}; ++i))
   do
        for j in "${intersection_with_previous[@]}"
          do
                if [[ "${pre_id_only[$i]}" == "$j" ]]
                then

                        index_arr_pre+=( "$i" )
                fi
        done
  done

echo "The list of the index for
previous jobs which are not in current job list are"
echo ${index_arr_pre[@]}

########################

for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
```

```
        for j in "${index_arr_pre[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_pre_added+=( "${pre_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding previous jobs are"
echo ${sum_pre_added[@]}

##########################
sum2=0
for i in ${sum_pre_added[@]}
 do
        sum2=`echo $sum2 + $i | bc`
 done
echo "The total sum of the previous job progress is" $sum2

########################
#tot_line=$(echo ${#sum_pre_added[@]} | bc)
#echo $tot_line

a=$((echo "${#sum_pre_added[@]}*100")|bc)
#a=$((echo "$tot_line*100")|bc)
#echo $a

total_progress=`echo $a - $sum2 | bc`
echo "The total done progress between the gap time was" $total_progress

#######################
#TO FIND OUT THE TOTAL PROGRESS
#OF THE JOB WHICH ARE IN BOTH STATE (PREVIOUS AND CURRENT)
##FOR THIS, CURRENT TOTAL PROGRESS AND
#PREVIOUS TOTAL PROGRESS WILL BE CALCULATED


#####FIRST TO FIND THE SIMILAR JOB IDS
for ((i=0; i < ${#curr_id_only[@]}; ++i))
 do
     for ((j=0; j < ${#pre_id_only[@]}; ++j))
        do
                if [[ "${curr_id_only[$i]}" == "${pre_id_only[$j]}" ]]
                then
                        similar_curr+=( "${curr_id_only[$i]}" )
                        index_similar_curr+=( "$i" )
                        index_similar_pre+=( "$j" )
                fi
        done
done

echo "THE SIMILAR IDS IN PREVIOUS AND CURRENT STATE ARE"
echo ${similar_curr[@]}
echo "THE INDEX OF THE SIMILAR VALUES IN CURRENT STATE ARE"
echo ${index_similar_curr[@]}
echo "THE INDEX OF THE SIMILAR VALUES IN PREVIOUS STATE ARE"
echo ${index_similar_pre[@]}

#################
#TO FIND THE CORRESPONDING VALUES
for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_similar_curr[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_similar_curr+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "THE CORRESPONDING VALUES OF THE SIMILAR JOBS IN CURRENT STATE ARE"
echo ${sum_similar_curr[@]}
#################
##TO CALCULATE THE SUM
sumsimilarcurr=0
for i in ${sum_similar_curr[@]}
 do
        sumsimilarcurr=`echo $sumsimilarcurr + $i | bc`
done
echo "The current value of sum of similar job progress is " $sumsimilarcurr
#################
##TO FIND THE CORRESPONDING VALUES OF THE PREVIOUS

for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
        for j in "${index_similar_pre[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_similar_pre+=( "${pre_value_only[$i]}" )
                fi
        done
done
```

```
echo "THE CORRESPONDING VALUES OF THE SIMILAR JOBS IN PREVIOUS STATE ARE"
echo ${sum_similar_pre[@]}
###############
###TO CALCULATE THE SUM
sumsimilarpre=0
for i in ${sum_similar_pre[@]}
 do
        sumsimilarpre=`echo $sumsimilarpre + $i | bc`
done
echo "The current value of sum of similar job progress is " $sumsimilarpre

###############
##TO FIND THE EXACT PROGRESS VALUE BY
#SUBTRACTION total_progress3 form total_progress2

sum3=`echo $sumsimilarcurr - $sumsimilarpre | bc`
echo "THE PORGRESS IN GAP IS " $sum3



####TOTAL CURRENT PROGRESS IS ######

total_current_progress=`echo $sum1 + $total_progress + $sum3 | bc`
echo "TOTAL CURRENT PROGRESS IS" $total_current_progress
echo $total_current_progress > progress2.txt
 }

################################
#### FUNCTION TO CALCULATE PROGRESS FOR 30% RESOURCE ALLOCATION
progress_third () {
app_id_current=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $1}')
value_current=$(yarn application -list |
grep "root" | grep "RUNNING" | awk '{print $8 $9}'| cut -d '%' -f1 |
awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')

while [[ "${#app_id_current[@]}" != "${#value_current[@]}" ]]; do
        app_id_current=()
        value_current=()
        app_id=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $1}')
        value_initially=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $8 $9}'|
 cut -d '%' -f1 | awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')
done
```

```
echo "$app_id_current" > progress_third_only_app_id.txt
echo "$value_current" > progress_third_only_value.txt

app_id=$(sudo cat /home/ramesh/progress_second_only_app_id.txt)
value_initially=$(sudo cat /home/ramesh/progress_second_only_value.txt)

curr_id_only=($(paste <(echo "$app_id_current")))
echo "CURRENT APP ONLY ID"
echo "${curr_id_only[@]}"

curr_value_only=($(paste <(echo "$value_current")))
echo "CURRENT APP ONLY VALUE"
echo "${curr_value_only[@]}"

curr_id_value=$(paste <(echo "$app_id_current") <(echo "$value_current"))
echo "CURRENT APP ID AND CORRESPONDING VALUES"
echo "$curr_id_value"

##############
pre_id_only=($(paste <(echo "$app_id")))
echo "PREVIOUS APP ONLY ID"
echo "${pre_id_only[@]}"

pre_value_only=($(paste <(echo "$value_initially")))
echo "PREVIOUS APP ONLY VALUE"
echo "${pre_value_only[@]}"

###############


#different=$(diff <( echo "$app_id" ) <( echo "$app_id_current" ))
different=$(diff -ia --suppress-common-lines
<( printf "%s\n" "${app_id[@]}" ) <( printf "%s\n" "${app_id_current[@]}"))
echo "DIFFERENCE BETWEEN THE APPLICATION
IDS IN THE CURRENT STATE, WHETHER LOST OR ADDED ARE"
#echo ${different[@]}

for i in ${different[@]}
do
echo $i | grep "application_" | awk '{print $1}' >> app_id_changed.txt
done



fetch_file_data=$(sudo cat /home/ramesh/app_id_changed.txt)
echo ${fetch_file_data[@]}
```

```
sudo truncate -s 0 app_id_changed.txt


#####################################
#intersection_with_current
####################################
for item1 in ${app_id_current[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_current+=( "$item1" )
        fi
    done
done
echo "FOLLOWING APPS ARE NEWELY ADDED"
echo ${intersection_with_current[@]}
#############################################
#TO ADD THE VALUES OF THE NEWELY ADDED JOB
#############################################




for ((i=0; i < ${#curr_id_only[@]}; ++i))
   do
        for j in "${intersection_with_current[@]}"
          do
                if [[ "${curr_id_only[$i]}" == "$j" ]]
                then

                        index_arr_curr+=( "$i" )
                fi
        done
  done

echo "The list of the index for newely added jobs are"
echo ${index_arr_curr[@]}

########################

for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_arr_curr[@]}"
         do
```

```
                if [[ "$i" == "$j" ]]
                then
                        sum_newly_added+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding are"
echo ${sum_newly_added[@]}

##########################
sum1=0
for i in ${sum_newly_added[@]}
 do
        sum1=`echo $sum1 + $i | bc`
 done
echo "The total sum of the currently added job progress is" $sum1


#############################
#intersection_with_previous
###########################
for item1 in ${app_id[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_previous+=( "$item1" )
        fi
    done
done


echo "FOLLOWING APPS WERE IN PREVIOUS BUT NOT IN CURRENT"
echo ${intersection_with_previous[@]}

###################################
#  ADD THE VALUES SUBTRACTIONG FROM 100
#################################

for ((i=0; i < ${#pre_id_only[@]}; ++i))
   do
        for j in "${intersection_with_previous[@]}"
          do
                if [[ "${pre_id_only[$i]}" == "$j" ]]
                then
```

```
                        index_arr_pre+=( "$i" )
                fi
        done
  done

echo "The list of the index for
 previous jobs which are not in current job list are"
echo ${index_arr_pre[@]}

########################

for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
        for j in "${index_arr_pre[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_pre_added+=( "${pre_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding previous jobs are"
echo ${sum_pre_added[@]}

##########################
sum2=0
for i in ${sum_pre_added[@]}
 do
        sum2=`echo $sum2 + $i | bc`
 done
echo "The total sum of the previous job progress is" $sum2

###########################################
#tot_line=$(echo ${#sum_pre_added[@]} | bc)
#echo $tot_line

a=$((echo "${#sum_pre_added[@]}*100")|bc)
#a=$((echo "$tot_line*100")|bc)
#echo $a

total_progress=`echo $a - $sum2 | bc`
echo "The total done progress between the gap time was" $total_progress

######################################
#####FIRST TO FIND THE SIMILAR JOB IDS
for ((i=0; i < ${#curr_id_only[@]}; ++i))
 do
```

```
        for ((j=0; j < ${#pre_id_only[@]}; ++j))
           do
                   if [[ "${curr_id_only[$i]}" == "${pre_id_only[$j]}" ]]
                   then
                           similar_curr+=( "${curr_id_only[$i]}" )
                           index_similar_curr+=( "$i" )
                           index_similar_pre+=( "$j" )
                   fi
           done
done

echo "THE SIMILAR IDS IN PREVIOUS AND CURRENT STATE ARE"
echo ${similar_curr[@]}
echo "THE INDEX OF THE SIMILAR VALUES IN CURRENT STATE ARE"
echo ${index_similar_curr[@]}
echo "THE INDEX OF THE SIMILAR VALUES IN PREVIOUS STATE ARE"
echo ${index_similar_pre[@]}

##################################
#TO FIND THE CORRESPONDING VALUES
for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_similar_curr[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_similar_curr+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "THE CORRESPONDING VALUES OF THE SIMILAR JOBS IN CURRENT STATE ARE"
echo ${sum_similar_curr[@]}
#################################
##TO CALCULATE THE SUM
sumsimilarcurr=0
for i in ${sum_similar_curr[@]}
 do
        sumsimilarcurr=`echo $sumsimilarcurr + $i | bc`
done
echo "The current value of sum of similar job progress is " $sumsimilarcurr

##################################
##TO FIND THE CORRESPONDING VALUES OF THE PREVIOUS

for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
        for j in "${index_similar_pre[@]}"
```

```
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_similar_pre+=( "${pre_value_only[$i]}" )
                fi
        done
done
echo "THE CORRESPONDING VALUES OF THE SIMILAR JOBS IN PREVIOUS STATE ARE"
echo ${sum_similar_pre[@]}
############################
###TO CALCULATE THE SUM
sumsimilarpre=0
for i in ${sum_similar_pre[@]}
 do
        sumsimilarpre=`echo $sumsimilarpre + $i | bc`
done
echo "The current value of sum of similar job progress is " $sumsimilarpre

###########################
##TO FIND THE EXACT PROGRESS
#VALUE BY SUBTRACTION total_progress3 form total_progress2

sum3=`echo $sumsimilarcurr - $sumsimilarpre | bc`
echo "THE PORGRESS IN GAP IS " $sum3
##TOTAL CURRENT PROGRESS IS

total_current_progress=`echo $sum1 + $total_progress + $sum3 | bc`
echo "TOTAL CURRENT PROGRESS IS" $total_current_progress
echo $total_current_progress > progress3.txt
}
#### FUNCTION TO CALCULATE THE
#DIFFERENCE BETWEEN SECOND AND FIRST PROGRESS

diff_first_speed () {
        fetch_second_value=$(sudo cat /home/
ramesh/progress1.txt | awk '{print $1}')
        fetch_first_value=$(sudo cat /home/
ramesh/progress2.txt | awk '{print $1}')
        echo "Second progress value and first
progress value are" $fetch_first_value $fetch_second_value
        speed_first=`echo $fetch_first_value - $fetch_second_value | bc`
        }

#### FUNCTION TO CALCULATE THE DIFFERENCE BETWEEN THIRD AND SECOND
diff_second_speed () {
        fetch_third_value=$(sudo cat /home/
ramesh/progress3.txt | awk '{print $1}')
        fetch_second_value=$(sudo cat /home/
ramesh/progress2.txt | awk '{print $1}')
        echo "Third progress value and second
progress value are" $fetch_third_value $fetch_second_value
        speed_second=`echo $fetch_third_value - $fetch_second_value | bc`
        }
#### FUNCTION TO CALCULATE THE TOTAL CURRENT PROGRESS
progress_current () {

app_id_current=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $1}')
value_current=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $8 $9}'|
 cut -d '%' -f1 | awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')

while [[ "${#app_id_current[@]}" != "${#value_current[@]}" ]]; do
        app_id_current=()
        value_current=()
        app_id=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $1}')
        value_initially=$(yarn application -list |
 grep "root" | grep "RUNNING" | awk '{print $8 $9}'|
 cut -d '%' -f1 | awk -F'.' '{print $1}'| awk -F'[^0-9]*' '{print $1 $2}')
done

app_id=$(sudo cat /home/ramesh/progress_third_only_app_id.txt)
value_initially=$(sudo cat /home/ramesh/progress_third_only_value.txt)

echo "$app_id_current" > progress_third_only_app_id.txt
echo "$value_current" > progress_third_only_value.txt


curr_id_only=($(paste <(echo "$app_id_current")))
echo "CURRENT APP ONLY ID"
echo "${curr_id_only[@]}"

curr_value_only=($(paste <(echo "$value_current")))
echo "CURRENT APP ONLY VALUE"
echo "${curr_value_only[@]}"

curr_id_value=$(paste <(echo "$app_id_current") <(echo "$value_current"))
echo "CURRENT APP ID AND CORRESPONDING VALUES"
echo "$curr_id_value"

#############################################################
```

```
pre_id_only=($(paste <(echo "$app_id")))
echo "PREVIOUS APP ONLY ID"
echo "${pre_id_only[@]}"

pre_value_only=($(paste <(echo "$value_initially")))
echo "PREVIOUS APP ONLY VALUE"
echo "${pre_value_only[@]}"

##############################################################
#different=$(diff <( echo "$app_id" ) <( echo "$app_id_current" ))
different=$(diff -ia --suppress-common-lines
<( printf "%s\n" "${app_id[@]}" ) <( printf "%s\n" "${app_id_current[@]}"))
echo "DIFFERENCE BETWEEN THE APPLICATION IDS
IN THE CURRENT STATE, WHETHER LOST OR ADDED ARE"
#echo ${different[@]}

for i in ${different[@]}
do
echo $i | grep "application_" | awk '{print $1}' >> app_id_changed.txt
done



fetch_file_data=$(sudo cat /home/ramesh/app_id_changed.txt)
echo ${fetch_file_data[@]}

sudo truncate -s 0 app_id_changed.txt


#####################################
#intersection_with_current
####################################
for item1 in ${app_id_current[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_current+=( "$item1" )
        fi
    done
done
echo "FOLLOWING APPS ARE NEWELY ADDED"
echo ${intersection_with_current[@]}
#############################################
#TO ADD THE VALUES OF THE NEWELY ADDED JOB
```

```
#############################################
for ((i=0; i < ${#curr_id_only[@]}; ++i))
   do
        for j in "${intersection_with_current[@]}"
          do
                if [[ "${curr_id_only[$i]}" == "$j" ]]
                then

                        index_arr_curr+=( "$i" )
                fi
        done
  done

echo "The list of the index for newely added jobs are"
echo ${index_arr_curr[@]}

########################

for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_arr_curr[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_newly_added+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding are"
echo ${sum_newly_added[@]}

##########################
sum1=0
for i in ${sum_newly_added[@]}
 do
        sum1=`echo $sum1 + $i | bc`
 done
echo "The total sum of the currently added job progress is" $sum1


################################################
#intersection_with_previous
#################################################
for item1 in ${app_id[@]}
 do
    for item2 in ${fetch_file_data[@]}
      do
```

```
        if [[ "$item1" == "$item2" ]]
         then
            intersection_with_previous+=( "$item1" )
        fi
    done
done


echo "FOLLOWING APPS WERE IN PREVIOUS BUT NOT IN CURRENT"
echo ${intersection_with_previous[@]}

############################################################
# ADD THE VALUES SUBTRACTIONG FROM 100
###########################################################

for ((i=0; i < ${#pre_id_only[@]}; ++i))
   do
        for j in "${intersection_with_previous[@]}"
          do
                if [[ "${pre_id_only[$i]}" == "$j" ]]
                then

                        index_arr_pre+=( "$i" )
                fi
        done
  done

echo "The list of the index for previous
jobs which are not in current job list are"
echo ${index_arr_pre[@]}

########################

for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
        for j in "${index_arr_pre[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_pre_added+=( "${pre_value_only[$i]}" )
                fi
        done
done
echo "The array of the value corresponding previous jobs are"
echo ${sum_pre_added[@]}

################
```

```
sum2=0
for i in ${sum_pre_added[@]}
 do
        sum2=`echo $sum2 + $i | bc`
 done
echo "The total sum of the previous job progress is" $sum2

###############
#tot_line=$(echo ${#sum_pre_added[@]} | bc)
#echo $tot_line

a=$((echo "${#sum_pre_added[@]}*100")|bc)
#a=$((echo "$tot_line*100")|bc)
#echo $a

total_progress=`echo $a - $sum2 | bc`
echo "The total done progress between the gap time was" $total_progress

############################
#TO FIND OUT THE TOTAL PROGRESS OF
#THE JOB WHICH ARE IN BOTH STATE (PREVIOUS AND CURRENT)
##FOR THIS, CURRENT TOTAL PROGRESS
#AND PREVIOUS TOTAL PROGRESS WILL BE CALCULATED


#####FIRST TO FIND THE SIMILAR JOB IDS
for ((i=0; i < ${#curr_id_only[@]}; ++i))
 do
     for ((j=0; j < ${#pre_id_only[@]}; ++j))
        do
                if [[ "${curr_id_only[$i]}" == "${pre_id_only[$j]}" ]]
                then
                        similar_curr+=( "${curr_id_only[$i]}" )
                        index_similar_curr+=( "$i" )
                        index_similar_pre+=( "$j" )
                fi
        done
done

echo "THE SIMILAR IDS IN PREVIOUS AND CURRENT STATE ARE"
echo ${similar_curr[@]}
echo "THE INDEX OF THE SIMILAR VALUES IN CURRENT STATE ARE"
echo ${index_similar_curr[@]}
echo "THE INDEX OF THE SIMILAR VALUES IN PREVIOUS STATE ARE"
echo ${index_similar_pre[@]}
```

```
##########################################
#TO FIND THE CORRESPONDING VALUES
for ((i=0; i < ${#curr_value_only[@]}; ++i))
 do
        for j in "${index_similar_curr[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_similar_curr+=( "${curr_value_only[$i]}" )
                fi
        done
done
echo "THE CORRESPONDING VALUES OF THE SIMILAR JOBS IN CURRENT STATE ARE"
echo ${sum_similar_curr[@]}
#################################
##TO CALCULATE THE SUM
sumsimilarcurr=0
for i in ${sum_similar_curr[@]}
 do
        sumsimilarcurr=`echo $sumsimilarcurr + $i | bc`
done
echo "The current value of sum of similar job progress is " $sumsimilarcurr

######## TO FIND THE EXACT VALUE OF PROGRESS TO BE DONE AT THAT TIME
#a=$((echo "${#index_similar_curr[@]}*100")|bc)
#total_progress2=`echo $a - $sumsimilarcurr | bc`
#echo "THE EXACT PROGESS AT THAT TIME WAS" $total_progress2

############################
##TO FIND THE CORRESPONDING VALUES OF THE PREVIOUS

for ((i=0; i < ${#pre_value_only[@]}; ++i))
 do
        for j in "${index_similar_pre[@]}"
         do
                if [[ "$i" == "$j" ]]
                then
                        sum_similar_pre+=( "${pre_value_only[$i]}" )
                fi
        done
done
echo "THE CORRESPONDING VALUES OF THE SIMILAR JOBS IN PREVIOUS STATE ARE"
echo ${sum_similar_pre[@]}
##########################################
###TO CALCULATE THE SUM
sumsimilarpre=0
```

```
for i in ${sum_similar_pre[@]}
 do
        sumsimilarpre=`echo $sumsimilarpre + $i | bc`
done
echo "The current value of sum of similar job progress is " $sumsimilarpre
########TO FIND THE EXACT VALUE OF THE PROGRESS AT PREVIOUS STATE

sum3=`echo $sumsimilarcurr - $sumsimilarpre | bc`
echo "THE PORGRESS IN GAP IS " $sum3


#################################################################
#################TOTAL CURRENT PROGRESS IS ######################

total_current_progress=`echo $sum1 + $total_progress + $sum3 | bc`
echo "TOTAL CURRENT PROGRESS IS" $total_current_progress
echo $total_current_progress > progress3.txt
}

####FUNCTION THAT RESET THE ARRAY EVERY TIME LOOP EXECUTE
reset_array () {
        pre_id_only=()
        pre_value_only=()
        curr_id_only=()
        curr_value_only=()
        different=()
        fetch_file_data=()
        app_id_current=()
        intersection_with_current=()
        index_arr_curr=()
        sum_newly_added=()
        app_id=()
        intersection_with_previous=()
        index_arr_pre=()
        sum_pre_added=()
        similar_curr=()
        index_similar_curr=()
        index_similar_pre=()
        sum_similar_curr=()
        sum_similar_pre=()
        }



#### FUNCTION TO WRITE THOSE METRICS INTO FILE
write_file () {
                #### TO WRITE THE METRICS FORM THE CLUSTER INTO FILE
        var=$(paste -d, <(echo "$tot_mem") <(echo "$mem_used")
<(echo "$mem_unused") <(echo "$marp") <(echo "$marp_limit")
<(echo "$am_mem_used") <(echo "$am_vcore_used") <(echo "$tot_core")
<(echo "$core_used") <(echo "$core_unused") <(echo "$app_running")
<(echo "$app_pending") <(echo "$cont_running") <(echo "$cont_pending")
<(echo "$capacity_used") <(echo "$speed_first") <(echo "$speed_second"))
        echo "$var" >> output_dynamic."csv"
        }


while true; do

        file=/home/ramesh/hadoop-2.8.1/etc/hadoop/capacity-scheduler.xml
        for marp in $(sudo cat $file | awk -F" " 'NR==26 {print $2}'); do


        #### CONDITIONS

        fetch_metrics
        write_file
        if [[ "$(bc -l <<< "$marp == $minmarp")" == "1" && $app_running > 0 ]]
        then

        sleep 15
        progress_first
        reset_array




        increment
        sleep 15
        progress_second
        reset_array

        break
        elif [ "$(bc -l <<< "$marp == 0.20")" == "1" ]
        then
                marp_thirty
                sleep 15
                progress_third
                reset_array
                fetch_metrics
                write_file


                diff_first_speed
                echo "Speed First" $speed_first
                diff_second_speed
                echo "Speed Second" $speed_second


                fetch_metrics
                write_file
                break

        elif [[ "$(bc -l <<< "$marp > 0.20")"
        == "1" && "$(bc -l <<< "$speed_second > $speed_first")" == "1" ]]
        then
                increment
                sleep 15
                value_from_two_to_one=$(sudo cat /home/
                ramesh/progress2.txt | awk '{print $1}')
                echo "$value_from_two_to_one" > progress1.txt
                value_from_three_to_two=$(sudo cat /home/
                ramesh/progress3.txt | awk '{print $1}')
                echo "$value_from_three_to_two" > progress2.txt
                progress_current
                reset_array
                diff_first_speed
                echo "Speed First" $speed_first
                diff_second_speed
                echo "Speed Second" $speed_second



        elif [[ "$(bc -l <<< "$marp > 0.15")"
        == "1" && "$(bc -l <<< "$speed_second < $speed_first")" == "1" ]]
        then
                decrement
                sleep 15
                value_from_two_to_one=$(sudo cat /home/
                ramesh/progress2.txt | awk '{print $1}')
                echo "$value_from_two_to_one" > progress1.txt
                value_from_three_to_two=$(sudo cat /home/
                ramesh/progress3.txt | awk '{print $1}')
                echo "$value_from_three_to_two" > progress2.txt
                progress_current
                reset_array
                diff_first_speed
                echo "Speed First" $speed_first
                diff_second_speed
                echo "Speed Second" $speed_second
        #                       fetch_metrics
        #                              write_file
        #                              break


        elif [ "$(bc -l <<< "$speed_second == $speed_first")" == "1" ]
        then
                  sleep 15
                value_from_two_to_one=$(sudo cat /home/
                ramesh/progress2.txt | awk '{print $1}')
                echo "$value_from_two_to_one" > progress1.txt
                value_from_three_to_two=$(sudo cat /home/
                ramesh/progress3.txt | awk '{print $1}')
                echo "$value_from_three_to_two" > progress2.txt
                progress_current
                reset_array
                diff_first_speed
                echo "Speed First" $speed_first
                diff_second_speed
                echo "Speed Second" $speed_second

                break

        elif [[ "$(bc -l <<< "$app_running > 0")"
        == "1" && "$(bc -l <<< "$app_pending == 0")" == "1" ]]
        then

                sleep 15
               value_from_two_to_one=$(sudo cat /home/
                ramesh/progress2.txt | awk '{print $1}')
               echo "$value_from_two_to_one" > progress1.txt
               value_from_three_to_two=$(sudo cat /home/
                ramesh/progress3.txt | awk '{print $1}')
               echo "$value_from_three_to_two" > progress2.txt
               progress_current
                reset_array
               diff_first_speed
               echo "Speed First" $speed_first
               diff_second_speed
             echo "Speed Second" $speed_second

               break


        elif [ "$(bc -l <<< "$marp == 0.80")" == "1" ]
        then
                decrement
                sleep 15
```

```
                value_from_two_to_one=$(sudo cat /home/
                ramesh/progress2.txt | awk '{print $1}')
                echo "$value_from_two_to_one" > progress1.txt
                value_from_three_to_two=$(sudo cat /home/
                ramesh/progress3.txt | awk '{print $1}')
                echo "$value_from_three_to_two" > progress2.txt
                progress_current
                reset_array
                diff_first_speed
                echo "Speed First" $speed_first
                diff_second_speed
                echo "Speed Second" $speed_second

               break
        elif [[ "$(bc -l <<< "$app_running == 0")"
        == "1" && "$(bc -l <<< "$app_pending == 0")" == "1" ]]
        then
                echo "Set to Default"
                awk 'NR==26{$2=a}1' a=$minmarp $file > tmp && sudo mv -f tmp $file
                yarn rmadmin -refreshQueues
                break


        else
                sleep 15
                value_from_two_to_one=$(sudo cat /home/
                ramesh/progress2.txt | awk '{print $1}')
                echo "$value_from_two_to_one" > progress1.txt
                value_from_three_to_two=$(sudo cat /home/
                ramesh/progress3.txt | awk '{print $1}')
                echo "$value_from_three_to_two" > progress2.txt
                progress_current
                reset_array
                diff_first_speed
                echo "Speed First" $speed_first
                diff_second_speed
                echo "Speed Second" $speed_second

                                break

        fi

done
sleep 15
done
done
```

A paper titled **‘In the Quest of Trade-off between Job Parallelism and Throughput’** is currently in writing phase which will be submitted soon for some international conferences along with my supervisors.