ENHANCING PERFORMANCE OF TASK TRACKER FOR HADOOP IN HETEROGENEOUS ENVIRONMENT

A thesis

Submitted in partial fulfillment of the requirements for the award of the degree of

DOCTOR OF PHILOSOPHY

IN

COMPUTER APPLICATIONS

By

Gurwinder Singh

Registration Number: 41600065

Supervised by

Dr. Anil Sharma

Transforming Education Transforming India

LOVELY FACULTY OF TECHNOLOGY AND SCIENCES

LOVELY PROFESSIONAL UNIVERSITY

PUNJAB

April 2022

Declaration of Authorship

I, Gurwinder Singh, declare that this thesis titled, “ Enhancing Performance of Task Tracker for Hadoop in Heterogeneous Environment” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this project has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project is entirely my own work.
- I have acknowledged all main sources of help.
- Where the project is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Gurwinder Singh
Registration No.:41600065
Date:

Certificate

This is to certify that the thesis entitled "Enhancing Performance of Task Tracker for Hadoop in Heterogeneous Environment", which is being submitted by Mr. Gurwinder Singh for the award of the degree of Doctor of Philosophy in Computer Applications from the Lovely Faculty of Technology and Sciences, Lovely Professional University, Punjab, India, is entirely based on the work carried out by him under my supervision and guidance. The work reported, embodies the original work of the candidate and has not been submitted to any other university or institution for the award of any degree or diploma, according to the best of my knowledge.

Dr. Anil Sharma
Professor
School of Computer Applications
Lovely Professional University
Phagwara, Punjab, India
Date:

Abstract

Big Data is one of the most discussed term by academicians and the IT industry now days. The information is collected and produced at very fast speed which surpasses the limit. At present, more than two billion persons in world are connected with Internet, and more than five billion people own mobiles. We are living in an era where structured and unstructured data is produced, consumed and stored in enormous amount on frequent basis. Database transactions, social media, images, audios, videos etc. are the major sources responsible for generating big data in huge capacity and diversity. This usually consists of large volumes of complex and growing data sets with numerous self-regulating sources that are difficult to process with the conventional techniques of data management. Using big data mining, organizations are able to extract useful evidences from these large data sets. In spite of big data gains, there are numerous challenges also and among these challenges maintaining data privacy is the most important concern in big data-mining applications since processing large scale of sensitive data sets such as health record, banking transaction records needs to be maintained in such a way that the private data should not be revealed to any unauthorized person.

In this epoch of data surge, big data is one of the significant areas of research being widely pondered over by computer science research community, and Hadoop is the broadly used tool to store and process it. Hadoop is fabricated to work effectively for the clusters having homogeneous environment but when the cluster environment is heterogeneous then its performance decreases which result in various challenges surfacing in the areas like query execution time, data movement cost, selection of best Cluster and Racks for data placement, preserving privacy, load distribution: imbalance in input splits, computations, partition sizes and heterogeneous hardware, and scheduling. The epicenter of Hadoop is scheduling and all incoming jobs are multiplexed on existing resources by the schedulers. Enhancing the performance of schedulers in Hadoop is very vigorous.

MapReduce has emerged as a strong model for processing parallel and distributed data for huge datasets. Hadoop, an open source implementation of MapReduce, turns out to be de-facto platform which is appropriate for storage of data in distributed as well as local machines to analyze and process huge amount of information on commodity hardware. Hadoop fragments the input file into number of data blocks to allocate them to various DataNodes in cluster. Hadoop must provide effective scheduling to process these data blocks in efficient way. One of the issues that play vital role in efficient processing of MapReduce is Data Locality which is caused due to overhead of network. Data locality is equipped for moving the computation adjacent to the data where it dwells. It is a key resource in distributed environment which influences the tasks accomplishing time. The issues which troubles data locality are cluster and network load, resource sharing, cluster environment, size of data blocks, number of mappers and reducers.

Hiring different flavours of virtual machines for Hadoop virtual cluster hosted in heterogeneous physical servers in cloud data-centre leads to many challenges for MapReduce job and task schedulers. Heterogeneity in workloads and execution environment is inevitable while processing huge volume of data in cloud environment. Maximizing data local execution is one of the challenging tasks in cloud virtualized environment to improve job latency and throughput for a batch of jobs. So, research in data block placement is at increasing pace as it is crucial for other big data processing tools like Spark in cloud environment for effective and efficient job and task scheduling. Once data blocks are loaded into virtual machines, it is not generally expected to move them around the virtual cluster again unless there is a requirement to meet replication factor. Because, it consumes huge network bandwidth and takes significant time to move on virtual network. Therefore, it is important to make right decisions before loading large datasets into virtual machines. Moreover, heterogeneous performance of virtual machines causes varying latency for the same task due to disk IO contention by co-located virtual machines. To handle these situations, initially the performance of individual disk in physical machines is predict to choose the right disk to store data blocks. Based on the performance of individual disk in a physical server, the number of data blocks to store in each virtual machine is determined. Moreover, map task execution is done based on the heterogeneous performance of virtual machines. This increases the number of data local execution leading to reduction in map phase latency, thereby job latency and throughput is improved. The ideas are simulated and compared with existing schedulers such as classical fair scheduler, RWS-based scheduler, and HMJS. Results indicate that our proposed scheduler outperformed those existing schedulers for makespan.

Improving the performance of the MapReduce scheduler is a primary objective, especially in a heterogeneous virtual cloud environment. A map task is assigned with an input split(IS) which consists of one or more data blocks. When a map task is assigned to more than one data block, non-local execution is performed. In classical MapReduce scheduling schemes, data blocks are copied over the network to a node where the map task is running. This increases job latency and consumes more network bandwidth within and between racks in the cloud data-center. Considering this situation, a methodology "Improving Data Locality using Ant Colony Optimization (IDLACO)" is proposed to minimize the number of non-local executions and virtual network bandwidth consumption when IS are assigned to more than one data block. First IDLACO determines a list of an optimal number of data blocks for each map task of a job to perform a non-local execution reducing the job latency and virtual network consumption. Then, the target virtual machine to execute the map task is determined on the basis of its heterogeneous performance. Finally, if a set of data blocks is transferred to the same node for repeated job execution, it is decided to temporarily cache those data block in the target virtual machine. The performance of IDLACO is analysed and compared with fair scheduler and Holistic scheduler based on the parameters, such as the number of non-local executions, average map task latency, job latency, and amount of bandwidth consumed for a MapReduce job. Results show that our proposed IDLACO significantly outperforms the classical fair scheduler and Holistic scheduler.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor, Dr. Anil Sharma, for his supervision, advice, and guidance from the very first day of this research as well as giving me extraordinary experiences throughout the work. I am truly very fortunate to have the opportunity to work with him. I found this guidance to be extremely valuable.

I am grateful to the friends and fellow researchers, for their constructive criticism and suggestions.

I would like to show my gratitude to the entire family of Lovely Professional University, for providing me a suitable research atmosphere to carry out my work in proper time. I would like to thank the Division of Research and Development and School of Computer Applications, for all the support encouragement throughout the research work.

I am also very much grateful to my family for their moral support and care that they shown me during the period of this work. Last but not least, I thank God for sailing me through all the rough and tough times during this research work.

Gurwinder Singh
Date:

Contents

List of Figures

List of Tables

Abbreviations

ABC	**A**mount of **B**andwidth **C**onsumed.
ACO	**A**nt **C**olony **O**ptimization.
BASE	**B**enefit **A**ware **S**peculative **E**xecution.
CDC	**C**loud **D**ata **C**Center.
CL	**C**luster **L**ocal.
CPU	**C**entral **P**rocessing **U**nit.
CSP	**C**loud **S**ervice **P**roviders.
FIFO	**F**irst **I**n **F**irst **O**ut.
IDLACO	**I**mproved **D**ata **L**ocality using **A**nt **C**olony **O**ptimization.
HDD	**H**ard **D**isk **D**rive.
HDFS	**H**adoop **D**istributed **F**ile **S**ystem.
HPMR	**H**igh **P**erformance **M**ap**R**educe Engine.
IO	**I**nput **O**utput.
IDC	**I**nternational **D**ata **C**orporation.
IS	**I**nput **S**plit.
IT	**I**nformation **T**echnology.
MRAppMaster	**M**ap**R**educe **A**pplication **M**aster.
NL	**N**ode **L**ocal.
NNLE	**N**umber of **N**on- **L**ocal **E**xecutions.
PKINIT	**P**ublic **K**ey cryptography for **INIT**ial Authentication.
PKTAAP	**P**ublic **K**ey utilizing Tickets for **APP**lication servers.
PHFS	**P**redictive **H**ierarchal **F**ast **S**pread.

RF	**R**eplication **F**actor.
RL	**R**ack **L**ocal.
RWS	**R**oulette **W**heel **S**cheme.
VM	**V**irtual **M**achine.
XML	E**x**tensible **M**arkup **L**anguage.
YARN	**Y**et **A**nother **R**esource **N**egotiator.

Dedicated to my beloved family…

Chapter 1

Introduction

1.1 Big Data

Now a day's large amount of data is generating at a rapid fire speed and in vast and diverse formats which are non-manageable by traditional databases. Hence, the term 'Big Data' is coined which made it promising to store, process, access and analyze this enormous measure of data. 'Big Data' doesn't mean only large quantity of data, it also includes the pace at which data is being generated, transferred, organized and re-organized. It also backs diversified formats of data like text, audio, video, images, machine logs etc. Factors like volume, velocity and variety make it radically different and divergent from conventional data . Furthermore, the word 'Big Data' is somewhat fresh and novel in field of IT and business industry. In the recent literature various experts and researchers have used the term and highlighted its use in a huge amount of scientific data for visualization [1]. A Plethora of definitions of 'Big Data' are extant. Thus, Big Data is "the amount of data just beyond technology's capability to store, manage, and process efficiently" [2].'Big Data' is primarily characterized by three Vs: Volume, Variety and Velocity [3][4]. These terms were initially used by Gartner to define big data features. But some researchers do believe that Big data should not be categorized by only three Vs given by Gartner but one more V i.e. Value should also be added in its definition [5][6]. Study done by McKinsey [2], revealed that big data can enhance productivity, upsurge competitive advantage and upturn economic surplus for consumers. As per a white paper of International Data Corporation [7] data generation by year 2025 is all set to reach the level of 163 zettabytes as depicted in Figure 1.1.

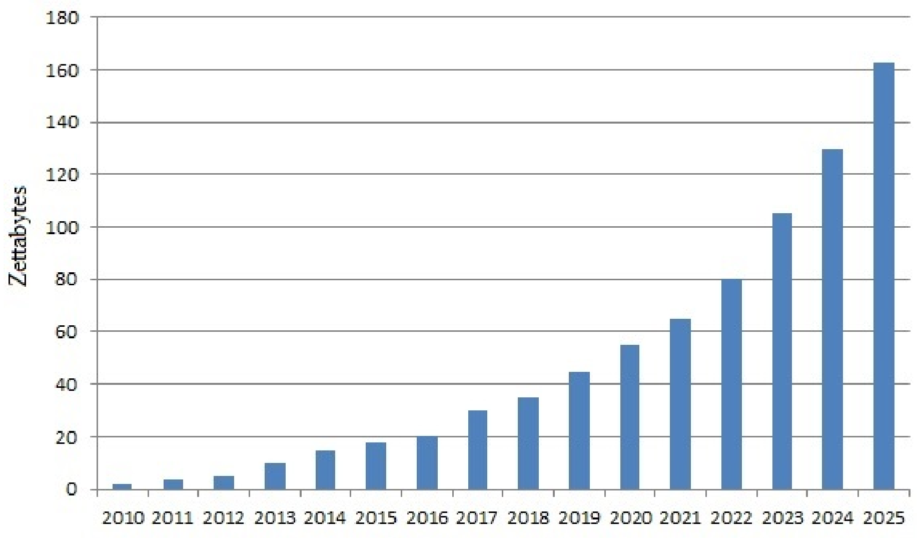

FIGURE 1.1: Projection big data 2010–2025, IDC's data age 2025 study [7]

A distributed database system is a group of self-governing machines coupled by participating networks that acts as single workstation [8]. To solve a complex problem requires the problem to be partitioned into sub problems where every sub problem is tackled by at least one of the computing node. These computing nodes can talk to one another via sending and receiving messages. The present scenario concerns about Big Data focuses on compute and data intensive tasks. A collection of huge data sets which are non-manageable for conventional tool are termed as Big Data [9]. Organizing Big Data is vital for business and big data analytics in terms of accessing data that arouses the requirement of such applications that can handle distributed processing of huge amount of data existing in different type of formats [8][9][10].

1.2 Hadoop

It is an open source framework of Apache Software foundation used for storage and processing of large and huge data sets (however for small datasets it is not recommended) with clusters of commodity hardware. Hadoop is written in Java and was originally conceived and designed by Dough Cutting in year 2005 [11]. Hadoop Ecosystem contains various tools such as HDFS, HBase, MapReduce, HCatalog, Pig, Mahout, Flume, Chukwa, Avro, Hive, Oozie, Zookeeper, and Kafka. Two central components of Hadoop are: HDFS and MapReduce [11].

Doug Cutting and Michael J. Cafarella, created Hadoop in context to be data intensive to support Nutch search engine project [12]. Hadoop is designed on the basis of master-slave architecture as shown in Figure 1.2. It offers easy solution for distributed and parallel computing with an ability of skipping the description related to communication recovery program [13]. The master JobTracker is responsible for management of resources of cluster, job scheduling, handling fault-tolerance and monitoring the progress. The TaskTracker module, present on each of the slave nodes, is accountable for throwing parallel tasks along with task status to the JobTracker. Responsibility of slave node here is to run as well as execute one or the other Map or Reduce tasks, and is bifurcated into static computing slots [14]. As a typical Hadoop cluster contains number of commodity computers therefore the jobs allocated to TaskTrackers need not be all the time Data or Rack local [15]. On deciding about the number of mappers and reducers, the user programs are executed by splitting the input file into default 64MB blocks and allocating these blocks among various slave nodes [15]. The physical distance of nodes and clusters may cause communication delays, resulting in longer waiting time for task's I/O and low utilization of CPU resources [15]. Hadoop offers, a replication policy for creating multiple copies of a block on different nodes plus racks, enhanced scalability and capability of installation on low-budget hardware

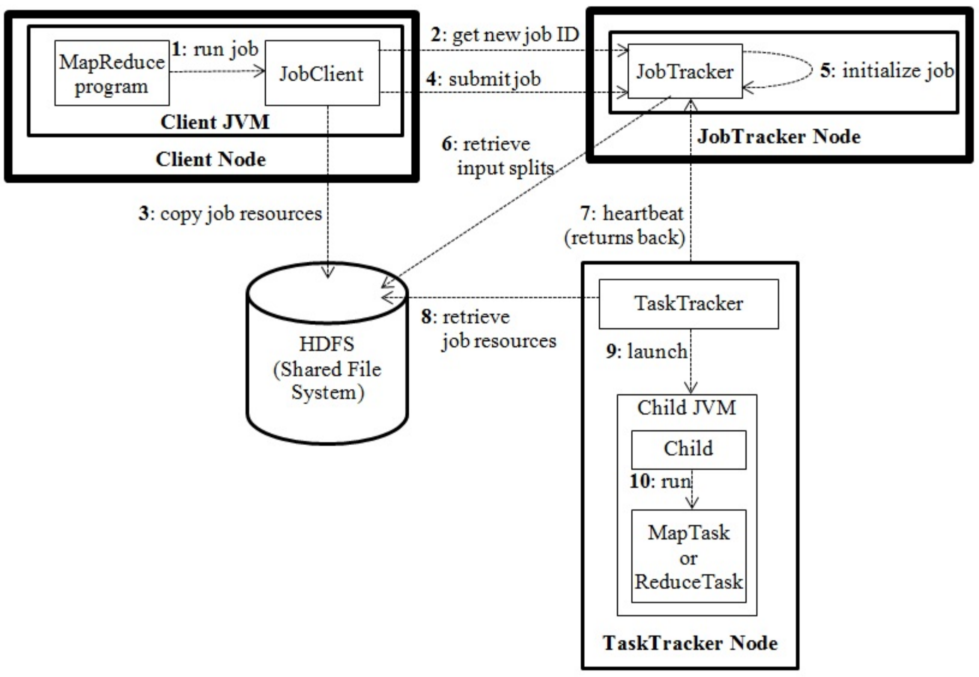

Figure 1.2: Hadoop Architecture

to deal fault tolerance [15]. It consists of modules Distributed file system (HDFS) and MapReduce model for storing and processing of data.

1.2.1 HDFS

The responsibility of HDFS is to store and preserve the data. It is a specifically intended file system utilized for storage of gigantic data sets on clusters of commodity hardware with streaming access pattern [16]. In HDFS the default block size is of 64 MB (can assign also 128MB or 512MB) [17]. The reason behind block size of minimum 64MB is that if block size is 4KB as in case of UNIX then HDFS needs to maintain more METADATA for small size blocks i.e. why it is used only for huge data sets [18]. HDFS follows master slave architecture as shown in Figure 1.3. Name Node, Secondary Name Node, DataNode, JobTracker, and TaskTracker are five services of HDFS [19]. All services operate internally at the background. All Master Services can communicate with each other and all the Slave Services can converse to each other [20].

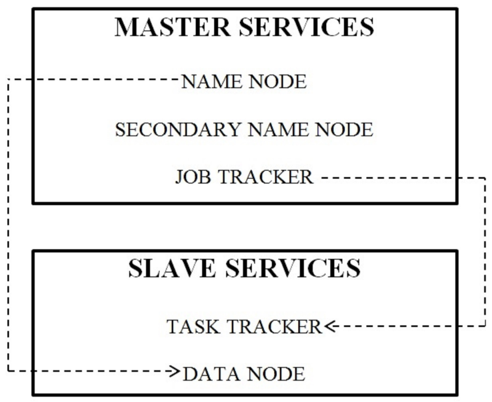

FIGURE 1.3: HDFS Architecture

By default, the replication factor for a block in HDFS is three. HDFS default replication policy only ensures two things [21]:

1. A DataNode cannot have more than one replica of any block.
2. A rack cannot have more than two copies of same block.

All DataNodes send ‘heartbeat message’ and block report to Name Node for every short period of time to say that certain clients have stored some blocks in local DataNode and are still alive, processing and working properly. If any of the DataNode is not giving proper heartbeat in-time then NameNode may think that DataNode has become dead and is removed from meta-data and then some other DataNode is chosen to store the copy of file to maintain the default replication property of HDFS. DataNode which was declared dead may start storing the data freshly [20].

Being motivated by GFS, Hadoop Distributed File System (HDFS) is used for storage of huge data (terabytes or even petabytes) and files on several computers [15]. By replicating data on geographically diverse nodes and different servers it attains reliability. These nodes dialogues to: rebalance scattered data, create and transport replicas, and preserve high data replication rate. HDFS contains: NameNode and DataNode where the NameNode acts as master in order to manage namespace and the DataNode is slave node used to store blocks of data nearby and remote locations following distributed policy to perform read/write requests [22].

1.2.2 Mapreduce

After storing data the next thing is processing the data which is performed using MapReduce. It consists of two major components: JobTracker and TaskTracker. JobTracker act as master and TaskTracker acts as workers [23]. JobTracker does not know which data is placed on which DataNode because there is no one-to-one communication between JobTracker and DataNode. JobTracker requests NameNode which in turn goes through its metadata to get the locations [23][24]. Client requests JobTracker to accept jobs which are in form of MapReduce programs comprising mapper and reducer methods, and set of input/output files. Location of these input files is retrieved from the metadata of NameNode by JobTracker. TaskTrackers are assigned tasks by JobTracker, applying suitable scheduling algorithm. The TaskTrackers apply the designated operations encoded in reducer, on the data elements spilled from mapper and generates the outcomes in the form of key-value pairs to write it on to HDFS [23].

MapReduce [25] is one of the main programming models that supports parallel and distributed processing of a huge datasets. It is extremely scalable and efficient platform because it allows use of number of commodity machines for distributed computing and furthermore it provides application programmers a linear execution of logic stated with mappers and reducers. The lower level of parallelization particulars are taken care at runtime. Important characteristics of MapReduce includes scaling, fault tolerant and it can be easily applied to data mining, machine learning and technical simulations. Among Hadoop, Sphere, Mars etc. Hadoop is commonly and extensively used MapReduce platform [26][27][13].

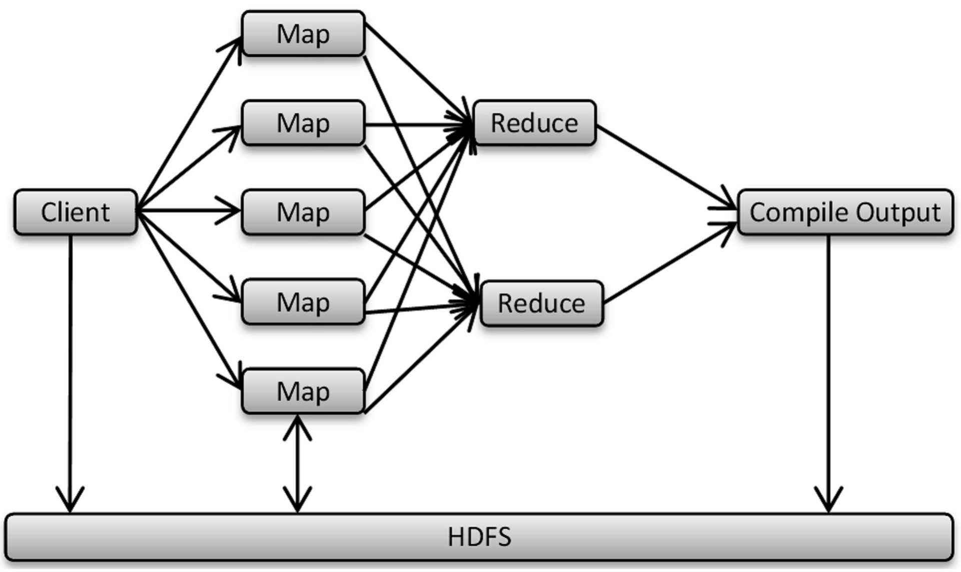

Figure 1.4: Map Reduce Structure

For storage of data Hadoop consists of Hadoop Distributed file system (HDFS) and processing of data is done via MapReduce model in HDFS following master-slave architecture as depicted in Figure 1.4 [28][29]. The JobTracker is master node and the TaskTrackers are slave nodes. Job is split into number of map and reduce tasks. The input of a job in Hadoop is divided into fragments of same size called input splits. Map tasks results are deposited to local disk and these intermediate results are input for reduce tasks. Further, the output of reduce task is generated and stored in HDFS [28]. Scheduling the reduce tasks in Hadoop comes up with several problems related to congestion because task scheduling process in Hadoop is based on pulling the data from source node. TaskTracker is responsible for sending a heartbeat message periodically and carting map-reduce tasks by issuing requests. JobTracker is responsible for ensuring each task to run on the designated TaskTracker which holds the required input splits. Thus this is the reason for MapReduce being data local at the time of map tasks scheduling. One of the negatives of Hadoop is that any reduce task which is still not running is scheduled to any of the demanding TaskTracker irrespective of location of TaskTracker in network [30][19][31].

1.3 Background

Exploding data growth [32] due to web and IoT based applications [33] has been affecting businesses and government sectors for effective decision making. Hadoop MapReduce [34] is one of the efficient batch processing tools for processing large scale dataset using cluster of unreliable computers. Nowadays, Hadoop MapReduce is offered as a service [35] on a cluster of virtual machines from cloud for pay-per-use basis. This includes significant challenges to be addressed to improve MapReduce job latency and throughput. In production environment, a batch of MapReduce jobs is executed repeatedly on the same dataset to extract information at different times. However, heterogeneity in execution environment [36] is a barrier to improving performance of MapReduce job scheduler further. After Hadoop 2.0 [37], users can configure the resource requirement (container size) of map and reduce tasks of each job in the batch. These will cause vast resource under-utilization due to unmatched container size of map/reduce tasks to fit in heterogeneous virtual machines. Moreover, hiring different flavours of virtual machines hosted in heterogeneous physical servers in cloud data-centre, as shown in Figure 1.5, leads to significant challenges [38], [39] for MapReduce job and task scheduling. There were some efforts to improve MapReduce job latency and throughput for a batch of heterogeneous MapReduce jobs while scheduling in heterogeneous environment. Data block placement in Hadoop Distributed File System (HDFS), and scheduling map/reduce tasks play a vital role in improving the performance of MapReduce scheduler. In general, data blocks are stored in HDFS with replication considering rack awareness in physical cluster, as shown in Figure 1.6. Consider three racks each with four computers and a file with three blocks. From Figure 1.6, it is easy to understand that rack-awareness is strictly held rather than holding equal number of blocks in each machine. On the other hand, virtual machines in the virtual cluster (Figure 1.5) are hosted in racks across cloud data-centre [40]. Rack awareness is not provided to store replicated data

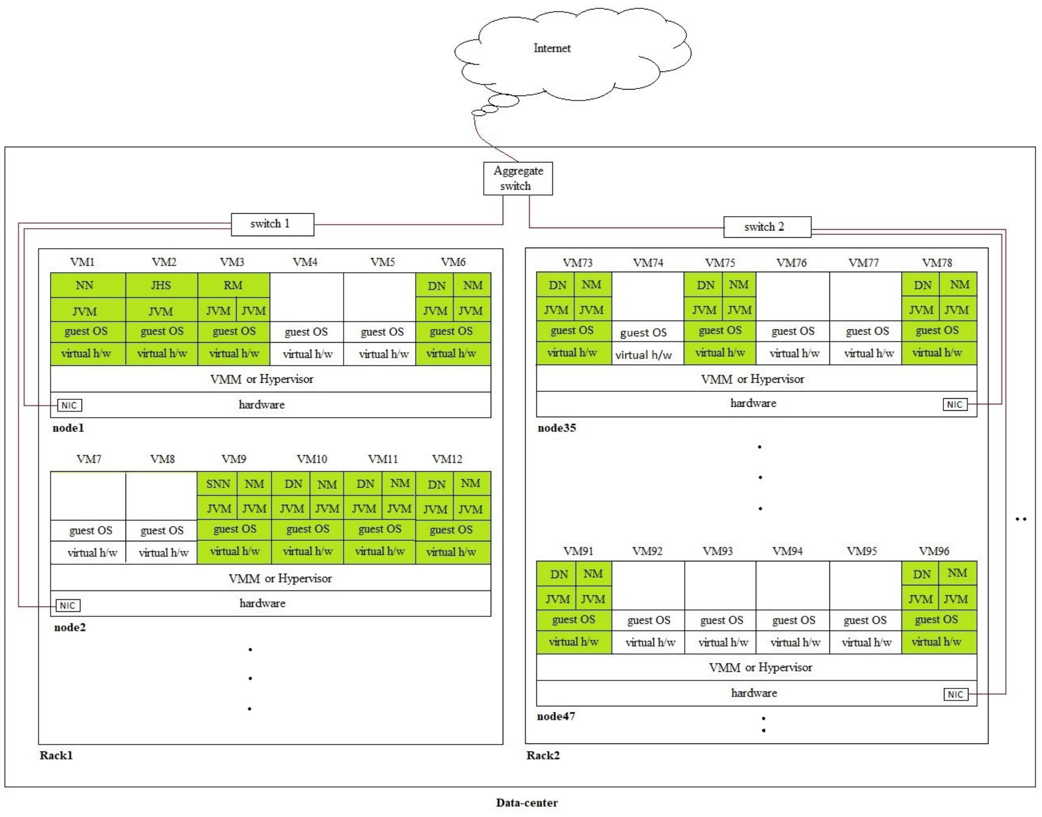

FIGURE 1.5: Hadoop Virtual Cluster

blocks to achieve fault tolerance in virtual cluster [34]. So, data blocks are stored evenly in virtual machines across virtual cluster. Moreover, once data blocks are placed into virtual machines, it is not generally expected to move them around in cluster again, unless there is a requirement to meet replication factor. Therefore, it is important to make right decisions before loading large datasets into virtual machines.

Secondly, opensource MapReduce job and task schedulers [41], [42] are mostly based on resource availability in the virtual cluster. As servers in cloud datacentre are highly heterogeneous in capacity and performance, virtual machines also exhibit the heterogeneous performance [40]. If many virtual machines are hosted in a physical server, sometimes their storage may be allocated to the same hard disk drive even though there are many hard disk drives available, as shown in Figure 1.7. Generally, in a server computer, CPU and memory are space-shared while disk

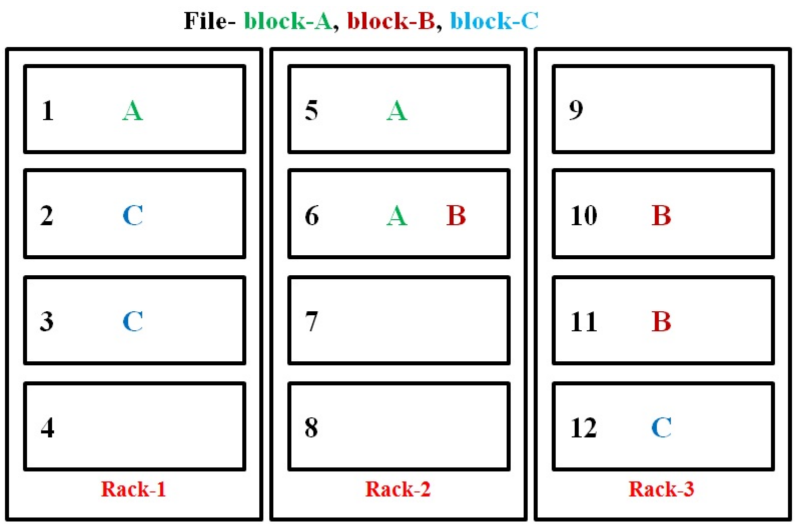

FIGURE 1.6: Rack-awareness

and network bandwidth are time-shared. This leads to varying latency for map and reduce tasks of a MapReduce job due to disk IO contention/race. It highly affects data local execution as disk IO is mostly busy [43]. To perform non-local execution, data blocks must be brought from disks, that consumes more time. In addition, the nature and resource requirement of map/reduce tasks of different jobs are highly heterogeneous. Therefore, it is necessary to exploit the dynamic performance of virtual machines based on the task requirement of different jobs achieving data local execution.

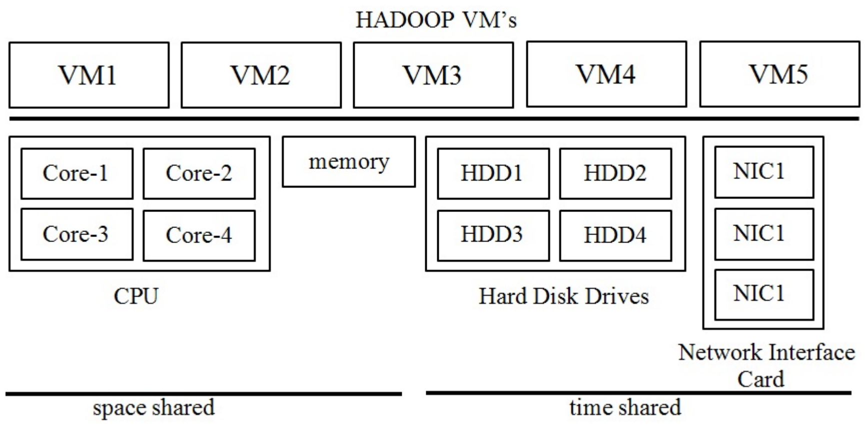

FIGURE 1.7: Resource Sharing [39]

To handle these situations, performance of individual disks in physical machines must be measured, because disk IO rate becomes a bottleneck when many Hadoop/non-Hadoop virtual machines get allocated storage in the same disk [40].

In general, MapReduce jobs tend be more disk-intensive, which needs constant disk bandwidth to bring input data blocks for map tasks execution. However, each virtual machine hosted in that physical server tries to share the disk bandwidth. This specifically leads to an increase in map task latency as map tasks read data blocks for preliminary processing. Therefore, data blocks must be stored based on the performance of disks, so that map tasks get input data blocks seamlessly. Moreover, as virtual machines in the hired virtual cluster could be of different flavour, the number of blocks processed per unit time significantly varies. So, it is important to understand the capacity and processing performance of different virtual machines before loading data blocks.

To achieve this, we initially find the performance of individual disk in a physical machine to choose the right disk to store data blocks. Based on the performance of individual disk in a physical server, the number of data blocks to store in each virtual machine is determined. MapReduce jobs in the batch can be CPU-intensive, network-intensive, memory-intensive, and IO-intensive. When they are scheduled without understanding its nature, for example, CPU-intensive job may be scheduled onto virtual machine which faces CPU bottleneck leading to increase in task latency, which in turn increases job latency. Therefore, understanding the performance of each virtual machine for map tasks of different jobs could help schedule map/reduce tasks in right virtual machine. Because, executing map tasks demand data locality to the maximum without moving them around in cluster while reduce tasks require to minimize bandwidth consumption. This can improve MapReduce job latency and throughput when MapReduce jobs are submitted as a batch.

1.4 Heterogeneity in Mapreduce for Cloud

As VMs for Hadoop virtual cluster are hosted across racks in a cloud data centre (CDC), it introduces various heterogeneities at different levels in a virtualized environment: hardware heterogeneity, VM heterogeneity, performance heterogeneity, and workload heterogeneity, as shown in Figure 1.8.

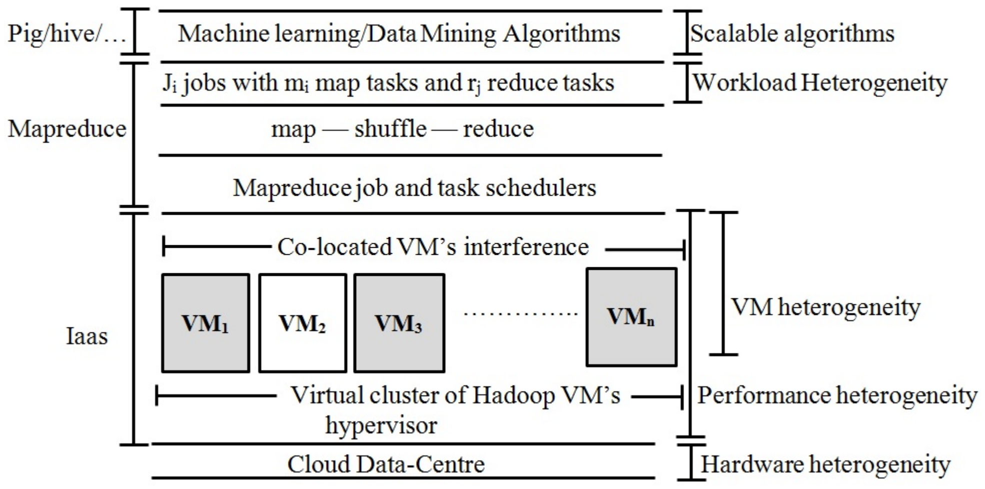
Pig/hive/...
Machine learning/Data Mining Algorithms
Scalable algorithms
J_i jobs with m_i map tasks and r_j reduce tasks
Workload Heterogeneity
Mapreduce
map — shuffle — reduce
Mapreduce job and task schedulers
Co-located VM's interference
Iaas
VM_1
VM_2
VM_3
VM_n
VM heterogeneity
Virtual cluster of Hadoop VM's
hypervisor
Performance heterogeneity
Cloud Data-Centre
Hardware heterogeneity

FIGURE 1.8: Heterogeneity at different levels [39]

1. A CDC containing physical servers of different configuration (processor type, memory size, etc.) and capacity is called hardware heterogeneity.

2. VMs in the virtual cluster could belong to different flavours, such as small, medium, and large, and be hosted in heterogeneous physical machines. It is called VM heterogeneity.

3. Hardware heterogeneity, VM heterogeneity, and co-locating VMs interference together cause heterogeneous performance of same map/reduce task running in a VM. It is called performance heterogeneity, which makes VM performance unpredictable at the infrastructure level. This causes various problems for data-intensive applications. For example, a low-performing VM might receive a greater number of data blocks to process, while a high-performing VM might receive very less number. This increases the map task

latency; thereby, increasing the MapReduce job latency. Thus, it is essential to consider the heterogeneous performance of VMs to improve MapReduce scheduler performance.

4. Varying number of tasks and its resource demand denote workload heterogeneity. In MapReduce, map and reduce tasks can be configured with different size of containers (resource size). More specifically, the number of blocks processed by map task of different jobs can also be customized. Even though it minimizes the number of map tasks, it can increase the number of non-local executions [32] and local bandwidth consumption at the MapReduce level.

1.5 Data Locality in Mapreduce

Data locality is an influential metric of Hadoop which makes considerable impact on system performance. Elements that play vital role in data locality include number of replicas, cluster size, and job execution time and stage. Creating more number of replicas results in improved data locality but it employs more storage space. If size of cluster is huge then it ventures reduced data locality. When job is in initial stage, the input data is on nodes and there are more number of tasks that are not mapped then there is high probability of data locality, consequently at time of job end the probability of data locality is low. Default scheduling algorithm of Hadoop, First in first out (FIFO), supports concept of data locality. The master JobTracker node when receives a heartbeat message from slave node, then initially attempts to look for a map task in the queue for a job whose input data is with that node. If searching of data is successful it results in node level locality and task is launched on particular node, but if node level locality not achieved then JobTracker efforts for rack locality as shown Figure 1.9. Further if again flops then a task is picked randomly which provides rackoff locality. FIFO is in support of data locality but it does have shortcoming of performing scheduling task to task irrespective of its impact on other tasks [27][28]. In view of balancing the load, data is disseminated by Hadoop to numerous nodes on the basis of approachability of disk space.

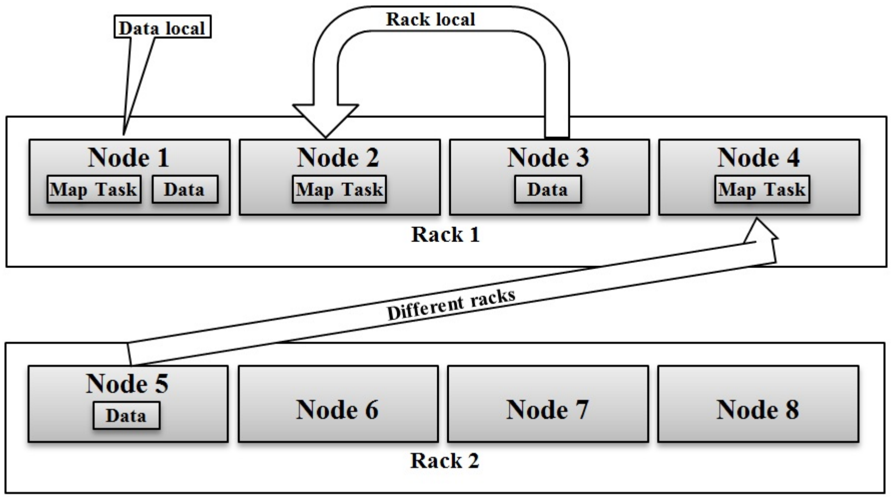

Figure 1.9: Types of data local execution during MapReduce job execution

The default policy for data locality is concrete and effective for homogeneous environment when nodes are indistinguishable i.e. workload of all nodes is equal which signifies that there is no requirement of shifting data from one to another node. When cluster environment is heterogeneous, local data can be processed fast by a node with high-performance than a node with low performance. Once the high-performance node completes local disks data processing, it is given the data residing in a distant sluggish node. Moving unprocessed data from a sluggish node to a high-performance node increases overhead if volume of data is enormous [29][30].

There are different types of data locality [44], as shown in Figure 1.10: Node local (NL), rack local (RL), and cluster local (CL) execution. Consider a cluster with two racks (rack1 and rack2), each with two physical servers (node1...node4). There is a set of data blocks (b1...b9) loaded in these physical servers. Assume each map task is assigned with an IS that refers to two data blocks.

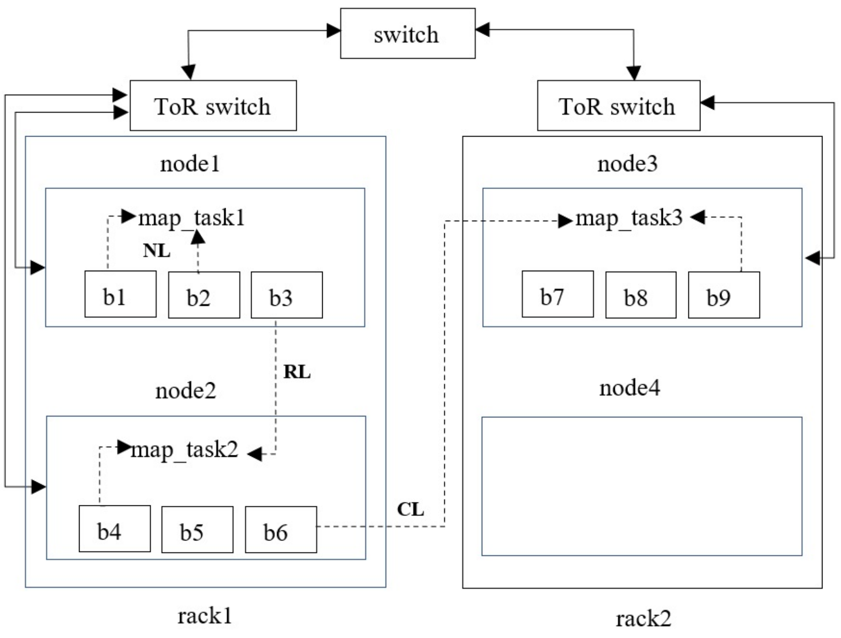

FIGURE 1.10: Types of data local execution

1. NL is executing a map task where the required data block resides. Thus, map_task1 processes b1 and b2 locally in node1. This does not transfer data outside the server.

2. If a data block is copied from one server to another server in the same rack, it is called RL execution. Thus, map_task2 in node2 processes b4 and b3, where b3 is retrieved from node1. The latency of RL is considerably higher than that of NL since it consumes real network bandwidth. If there is more than one virtual machine hosted in a single server, there could be virtual network bandwidth consumption between virtual machines hosted in the same server.

3. If a data block is moved off the rack across the data center, it is called CL execution. Blocks that are copied must cross three switches (top of the rack (ToR) switches and a central switch) to reach the target node. As shown in Figure 1.10, IS for map_task3 includes b9 and b6, where b6 is copied over network to node where map_task3 is hosted. It increases job latency and

consumes more physical network bandwidth. So, closer racks are preferred for non-local execution in this case.

1.6 Data blocks and Input splits

Typically, a MapReduce job consists of a set of map and reduces tasks. A map task is assigned with an input split (IS), which points to one or more physical data blocks. It is worth noting that data block and IS are not the same. A data block is a physical entity in Hadoop distributed file system (HDFS) while IS is a logical entity in the MapReduce during a job execution. IS points to one or more data blocks processed by a map task. As shown in Figure. 1.11, consider a file of size 256 MB with a data block size of 64 MB.

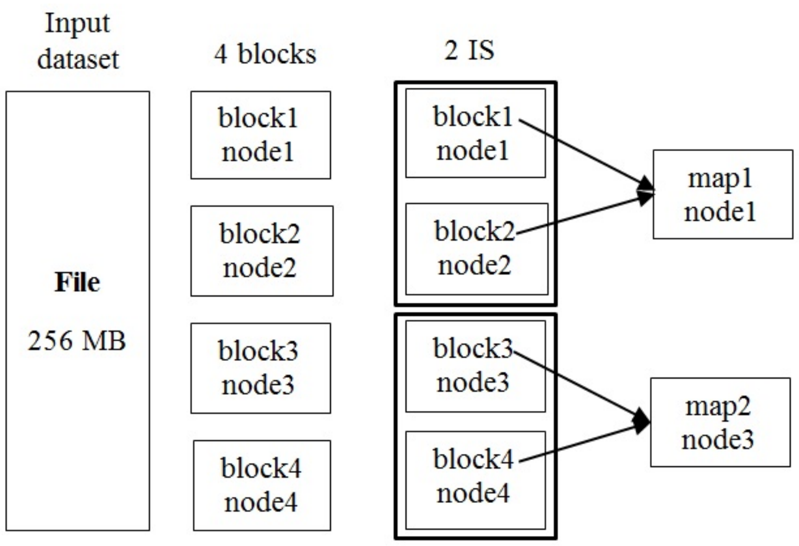

FIGURE 1.11: HDFS data blocks vs Mapreduce IS

This results in four data blocks in HDFS with the desired number of replications. If each IS is configured to point two data blocks, only two map tasks are launched. These two data blocks are logically linked and processed by each map task. Thus, the MapReduce scheduler forms a logical link to execute all data blocks sequentially for a map task. As shown in Figure. 1.11, consider the map task (map1) launched in node1 where the first data block resides. Now, block2 is copied from node2 over the virtual network to node1 to finish the first map task execution. Similarly, if an IS is configured to contain five data blocks, a map task is launched in the node where first data block resides. The other four data blocks are copied from their respective node to the node where the map task is launched. When the number of data blocks in an IS for a map task increases, the number of non-local executions (NNLE) increases, consuming more virtual network bandwidth. Besides, data blocks are stored on the basis of topological (rack) awareness, which is feasible only in the physical cluster. It is not yet known to be implemented for a virtual cluster in a cloud environment. Thus, VMs in the virtual cluster for MapReduce could be in the same rack or distributed across racks in the CDC. If all virtual machines in the virtual cluster are hosted in a single physical machine, there will not be any non-local executions. When VMs are hosted in different racks across the CDC, network bandwidth consumption is highly critical since it must transfer data through hierarchical switch connections. Sometimes, non-local execution is performed when slave nodes do not have any free slots to process data blocks locally. Thus, data blocks are copied over the local network, which incurs communication costs.

1.7 Motivation of the study

Heterogeneity in workloads and execution environment of Hadoop mapreduce is unavoidable while processing huge amount of data in the cloud virtualized environment. Therefore it affects the performance of hadoop mapreduce schedulers. This motivated us to design scheduling algorithm's, for heterogeneous virtualized environment, to increase the number of data local execution for improving job latency, throughput, number of non-local executions, average map task latency, and the amount of bandwidth consumed for a MapReduce job. Problems with the existing works(research gaps) are:

1. **Heterogeneous performance**: Just understanding the performance of a node from the near past may not improve task latency and resource utilization. For example, the map phase requires more of disk IO and CPU, while reduce phase requires network IO and CPU. When performance for a node is calculated based on CPU and Disk IO, reduce task might face bottleneck due to network congestion. Performance of a VM also varies dynamically due to the interference of co-located VMs. Also, independent jobs go for non-local execution due to static scheduling decision. Moreover:
   (a) All these works suffer from computational load imbalance as data locality is mandatory to minimize the number of non-local execution.
   (b) Data locality is significantly affected while concentrating on the performance of a node to place tasks.
   (c) Dynamically tuning container configurations minimizes the latency but at the cost of resource under-utilization.
2. **Heterogeneous workloads**: While placing heterogenous tasks in heterogeneous VMs, a large portion of virtual cluster resource is wasted. So, finding the right combination of tasks to schedule in each VM is a possible option for task scheduling.

1.8 Objectives of the study

In order to explore and enhance the working of Hadoop in heterogeneous multi-node clusters, this research aims to achieve following objectives:

1. To design a novel scheduling algorithm to improve the Data Locality in Hadoop's heterogeneous multi node cluster.
2. To improve the amount of data processed in Hadoop's heterogeneous multi node cluster.

3. To integrate the designed algorithm with Hadoop.
4. Testing of results to analyse the performance of algorithm.

1.9 Thesis Contribution

The research objectives are achieved by categorizing them into two works:

Work 1(Chapter 3):

1. To improve the amount of data processed in Hadoop's heterogeneous multinode cluster.
2. To integrate the designed algorithm with Hadoop.
3. Testing of results to analyse the performance of algorithm.

Work 2(Chapter 4):

1. To design a novel scheduling algorithm to improve the Data Locality in Hadoop's heterogeneous multi node cluster.
2. To integrate the designed algorithm with Hadoop.
3. Testing of results to analyse the performance of algorithm.

Focus of thesis is to to design scheduling algorithm's to improve data locality and improving the amount of data processed in heterogeneous multi node cluster of hadoop. The contribution of thesis can be categorized as:

1. A review of various existing scheduling algorithms in hadoop along with their key features and challenges.

2. A review and comparison of Scheduling Algorithms that are aware of Data locality in Hadoop with their shortcomings.
3. Prediction based data block placement algorithm to improve amount of data processed.
    (a) Predicting disk IO bandwidth of every disk in each physical machine before loading data blocks.
    (b) Scheduling map tasks of different jobs based on the heterogeneous performance each virtual machine.
4. An ACO based map task scheduler algorithm to improve data locality.
    (a) To calculate heterogeneous performance of virtual machines.
    (b) To find a list of data blocks for each IS that can minimize the number of non-local executions and bandwidth consumption using ACO.
    (c) To cache data blocks that are frequent in the target virtual machine.

1.10 Thesis Outline

The thesis is organized in five chapters. A brief outline of the chapters is given below:

Chapter 1 introduces to the domains of Big Data, Hadoop, Data Locality along with background and motivation behind this work. It also highlights objectives of study and contribution of thesis.

Chapter 2 presents the allied research work done by different researchers in this domain. Literature review has been classified into sub-parts. Initially, the existing default algorithms were compared with other popular schedulers in Hadoop along with their key features and challenges . Secondly, features and drawbacks of various scheduling algorithms related to data locality and then review of data block placement in hadoop to improve amount of data processing.

Chapter 3 proposes a method to predict IO bandwidth of every disk and migrating the data blocks. The aim of proposed method is to improve amount of data processed in heterogeneous hadoop clusters. The results along with experiments conducted are discussed in this chapter.

Chapter 4 proposes a method to minimize the number of non-local executions and virtual network bandwidth to minimize global transfer cost using ACO. The basic idea of this proposed method is to improve data locality in heterogeneous hadoop clusters. The results along with experiments and comparison to existing methods are discussed in this chapter.

Chapter 5 conclude the thesis underlining the key conclusions of the current research and author's significant contribution in the thesis. The scope for future research in this area is also notified.

Chapter 2

Review Of Literature

2.1 Literature Review

2.1.1 Scheduling Algorithms in MapReduce

Several regions of attempt have concerns with enormous information and more or less traditional business applications have confronted enormous information for quite a while, such as aircraft reservation frameworks, and more modern business applications to exploit massive information are under development for e.g. information sharing hubs, groups of databases). Big data problem can be split into two individual issues: Big data objects and big data collections [1]. Scientists in computer field as well as biologists are struggling with more and more huge data sets these days [45]. Various problems that surface and required to be surmounted while processing big data includes timeliness, scalability, privacy, error handling, visualization and heterogeneous data and the same must be taken into consideration for highly effective and accomplished processing of big data. This chapter also seeks to compare some of the Hadoop's components like Hive, HBase, Cassandra, MongoDB and Redis of Hadoop [46]. Public-key based extensions: Public

Key cryptography for INITial Authentication in Kerberos (PKINIT), Public Key utilizing Tickets for APPlication servers (PKTAPP) and Public Key Cryptography for CROSS-Realm Authentication in Kerberos (PKCROSS) in [47] provides extra support to public-key cryptography in Kerberos framework at various stages which results in enhanced and improved security and as well as scalability.

Comparison of performance of Hadoop clusters in homogeneous and heterogeneous environments was made and a new load balancing framework for MapReduce was for chalked out much more efficacious performance better in both environments. Furthermore, instead of bifurcating the tasks for participating nodes equally, as in conventional framework of MapReduce, MARLA ("MApReduce with adaptive Load balancing for heterogeneous and Load imbalAnced clusters") forms a large number of tasks conceived from input divisions, more noteworthy in number than the summation of its nodes. Thus, this load balancing technique permits the contributing nodes to themselves appeal the tasks as and when previous task is completed [48]. Load balancing scheme in [49] has taken advantage of Cloud Storage's grey prediction theory to predict the load rate for various replicas at some particular instance for a node to perform different operations efficiently. Yet another forsighted and pioneering dynamic strategy envisaged to balance the load in HDFS is by placing blocks on DataNodes considering metrics bandwidth of network and fault tolerance [50].

Information dispersal algorithm in [51] along with data placement strategy is used for encoding the files to the blocks and for storing the blocks in cluster to improve storage efficiency to enhance availability of data and for better network load both the storage load of DataNode and usage of network were combined. Data placement solution in [52] improves data transfer time in Hadoop by calculating the bandwidth among the DataNode and client periodically, and exhausting the DataNode having maximum bandwidth placing data blocks. By combining Disk Utilization and Service Blocking Rate Model's new improved technique in [53] has been proposed to balance the load in HDFS as compared to load balancing method used in existing HDFS. Workload imbalance of unbalanced node can be balanced by minimizing the data transfer time by computing the capacity of each node and balancing the data stored in each node dynamically for heterogeneous Hadoop environment [54]. Load balancing can be improved by heightening the execution of reduce task as referred to [55] by accomplishing all reduce tasks at same time based on performance of nodes by utilizing the historical information and assigning the tasks to the nodes as per performance i.e. the input to nodes with poor execution is diminished. In [56] the authors introduced a new node called BalanceNode with which DataNodes having heavy-load and light-load can be compared, to enable the light loaded nodes to share a portion of load from heavy loaded nodes, consequently to minimize the blocks movement cost. Based on system information a load balancing method envisaged by [57] undertakes total available CPU cores and the memory size at disposal to identify the performance of each Data Node and implements the algorithm with sufficient and insufficient memory environment.

Triple-H hybrid design for heterogeneous storage architecture given by [58] hides the disk access cost for Read/Write operations, cutting down I/O bottlenecks, ensures the trustworthiness by using SSD-based staging for HDFS and efficient use of different types of devices on High Performance Computing Clusters. Multiple partitioning technique[59], resolves the problem of load imbalance at MapReduce stage, instigated by Hadoop's default partition algorithm, by refining the tasks and balancing the reducer input in the map phase to improve job scheduling and resource utilization. To deal with problem of maintaining privacy while executing the sensitive and critical data on unsecured cluster [60] introduces a dynamic scheduling algorithm to balance the load and transfer data among Hadoop racks on the basis of log files without revealing the private information. Anonymization algorithm for probabilistic inference attack and similarity attack by [61] provides better and effective privacy and data utility on the basis of resilient and clustering. Data encryption scheme for HDFS given by [62] for ARIA and Advanced

Encryption Standard algorithms together on Hadoop provides splitting of blocks and data processing component for variable-length blocks to implement encryption and decryption in efficient way for various applications, such as sorting, word counting, k-Means, and hierarchical clustering.

Dynamic Information as a Service architecture for Hadoop cluster by [63] using Scheduling, Monitoring, Virtual Machine Management, and Virtual Machine Migration modules highly useful in providing load balancing which in turn enhances data locality, resource scheduling. For maximum utilization of the resources which are held by idle slots [29] held out BASE (Benefit Aware Speculative Execution) as resource stealing scheduling algorithm which improves execution time of job and reduces the speculative tasks which are not advantageous in MapReduce by stealing the resources from idle slots. Longest Approximate Time to End scheduling in [64] overcomes the weaknesses of both progress-rate-based and threshold-based scheduling by detecting straggler tasks early and improves the response time of Hadoop by factor of 2 on a cluster of 200 machines on Amazon's Elastic Compute Cloud. Deadline Constraint algorithm envisaged in [65] for scheduling confirms execution of jobs for which deadlines can be encountered through a cost model for executing the job, but it does not pay any head to features like estimating filter ratio and run time for MapReduce task, distribution of data. Dynamic Priority Multiqueue algorithm [66] renders the tasks near to finish the job on a priority to adorn the response time for Hadoop jobs in MapReduce phase.

2.1.2 Data Locality in HDFS

Google [8] has been productively using the MapReduce programming model for diverse purposes and this achievement can be endorsed for numerous causes:

i. Even the programmers who have no exposure to distributed and parallel systems can use MapReduce model as it conceals the subtle elements of parallelization, fault tolerance, locality optimization, and load balancing.

ii. Extensive problems like generating data for Google's Web search service, sorting, machine learning, data mining, and lots of other systems, can be expressed easily as MapReduce computations.

iii. Made it possible for MapReduce to scale to huge clusters consist of enormous number of machines by efficiently utilizing the resources and is appropriate for computational problems come across at Google.

Degraded performance of Hadoop due to heterogeneity inspires to propose a data placement strategy [28] for placing data across nodes, to improve data locality, in manner that every node has stable load of data processing. Furthermore this scheme smartly performs load balancing to attain enriched data-processing on the data stored in each node. The performance evaluation on actual data-intensive applications Grep and WordCount reveals that data rebalancing among participating nodes earlier than data-intensive operation consequently improves performance of heterogeneous clusters in MapReduce.

A new task scheduler [30] introduced specifically for scheduling reduce tasks to enhance its data locality to effectively increasing the processing of requesting nodes. The technique confirms run the reduce tasks right TaskTracker so that network bandwidth can be used better. This proposed scheduler can minimize local data shuffling by 11-80% as shown in experiments.Dynamic task scheduler [67] used for predicting performance of number of MapReduce jobs and dealing with designation of resources to each job at runtime to accomplish the target without squandering resources. This scheduler foresees the expected finishing time for individual MapReduce job by exploiting fact that job in MapReduce is comprised of several tasks (include mappers and reducers) to accomplish, in addition the number of tasks are deliberated ahead of time amid the phase of job initialization at the time of input split, and observing performance of job at runtime. Considering both individually submitted and yet not finished jobs, the scheduler observes the time taken by previously finished tasks to know average task length for envisaging the job finishing time. This estimation allows the scheduler to adapt required task slots for each job to be allocated at runtime. Experiments conducted to study the influence of data locality and the problems allied with usage of memory in MapReduce phase of Hadoop reveals that reading of data locally is quicker (around 1.5-2 seconds instead of 3 seconds) than reading from distant nodes in HDFS which attributes to distress data transfer on network, and large inputs leads to failure of one slave node which in turn affects the other slave node resulting in degrading performance in terms to run task and serving data to distant TaskTracker.

Prefetching and Pre-shuffling schemes [68], implemented in High Performance MapReduce Engine (HPMR), proved to enhance the performance of shared environment of MapReduce computation, as a solution to problem that Hadoop-On-Demand (HOD) upsurges the use of resources on the cost of performance when number of users are in race for network and hardware resources which significantly deteriorates the performance, in contrast to dedicated environment for MapReduce. Prefetching scheme enhances data locality using intra-block prefetching and inter-block prefetching. Two issues of intra-block prefetching: essential to synchronize the computation and catch suitable prefetching rate, tackled by introducing concept of bi-directional processing bar to synchronize computation along with assessing the efficacy of technique, look for suitable prefetching rate to maximize performance by reducing I/O overhead and eliminating duplicate read operations. Inter-block prefetching, pre-fetch the required replica of block to the local rack by perused it from the node with minimum loaded replica's so as not to limit the effect of performance as whole. Pre-shuffling drops the shuffling's required for intermediary outputs by observing the participating node data or input splits at map phase to envisage which reducer the pairs of key-value are segregated. Data locality is main concern of prefetching scheme while pre-shuffling is responsible for moderating the overhead of shuffling overhead in reduce phase.

In [69] authors designed a parallel task scheduler which can control the allotted capacity of resources to users via regulating their time spent and capable of effective preemption and work conservation. This dynamic priority technique provides effective decisions regarding prioritization of jobs and tool for users for improving and altering the capacity to cope with their job requirements. Due to lightweight design its plus side includes improved data locality, enhanced scalability and overhead of virtualization. Whenever required the users can also scale back the consumption of resources, but the cost of this is too expensive. The major drawback of this approach is not strictly imposing the properties of isolation and no proper mechanism of dealing with long running tasks. A fair scheduler [70] is designed, to elucidate the clash among data locality and fairness, by proposing a delay scheduling technique, for a cluster of 600 nodes, at Facebook. The jobs are scheduled according to fairness and need to wait slightly to let the other jobs to launch tasks. This algorithm claims to attain almost optimum data locality and fairness for variety of assignments and upturns the throughput. If fairness is imposed strictly it results in loss of locality in two ways: sticky slots and head-of-line scheduling, although 100% locality can be attained by slightly compromising fairness. Whenever the number of jobs is changed, resources are reallocated by killing the executing tasks to entertain new job and wait to let the running task finish for better fair sharing. This delay scheduling method has been executed in Hadoop Fair Scheduler and improves the response times by 5x for light jobs while and doubles the throughput for heavy assignments.

A mathematical model was proposed in [71] to calculate the cost involved in assigning tasks for MapReduce framework of Hadoop. Outcomes of this model include: Optimal assignment of tasks cannot be achieved unless P = NP, discards the supposition that more replicas into system always results in better load balancing, algorithm for overall Hadoop task assignment by offering maximum flow and improved threshold procedures using additional constant that bank on merely number of servers and cost function for data locality. To improve data locality in homogeneous environment [72] introduces NKS (next-k-node) scheduling strategy for map task by analyzing probability of all map tasks to schedule highest probability task. Low probability tasks can be reserved for nodes that satisfies node locality. The problem with default policy is that it processes some of the map tasks irrespective of node locality by choosing the task which is traversed first, irrespective of whether the nodes having input data for task's may issue requests later on. Experimental results of NKS strategy shows: 77% network load reduction, 78% decrease in those map tasks that are processed without node locality and enriched performance of MapReduce. Distributed adaptive data replication algorithm [73] aims to increase data locality with an adaptive replication method having minimum overhead, via automatically creating more replicas for popular datasets and least replications for unpopular datasets. The method promises minimum network traffic, flexibly adapting changing file access methods and put a budgetary limit on additional memory space to be occupied by replicas of data. By using probabilistic sampling along with aging algorithm individually for respective node this technique resolve issues of creating lots of replicas for every file and where to store these replicas. It increases data locality by 7x as compared to FIFO scheduler and reduces turnaround time by 19% and job slowdown 25%.

The algorithm in [74] proposes a method to reduce access latency by foreseeing the forthcoming file usages. In order to increase locality Predictive Hierarchal Fast Spread (PHFS) algorithm pre-replicates data in hierarchal order and attempts to upsurge locality in accesses by calculating user's dynamic adaptability for replicas and with assumption that users working in on the similar environment will demand high probability files. As compared to common fast spread, PHFS results in improved access latency. An algorithm to provide optimal data locality by scheduling multiple tasks simultaneously by emphasizing merely on locality at node level i.e. placing data and compute on same node is proposed in [75]. Scheduling the tasks one-by-one without caring about its influence of one task on other tasks makes default scheduling methods non-optimal. Therefore Linear Sum Assignment Problem has been integrated with proposed method to consider all the available multiple tasks and idle slots at once. Considering the state when tasks to be scheduled are not more than available idle slots, experiments performed on different parameters: number of tasks, number of nodes, slots per node, idle slots ratio and replication factor that affects the impact of data locality reflects enhancement in goodness of data locality by 14%. Authors mainly focus on determining optimality of default algorithms in Hadoop as compared to Linear Sum Assignment Problem.

To address the overheads allied to data locality in Hadoop [76] proposed a replication scheme for data on the basis mechanism access count prediction to enhance data locality and minimizing data transfer time resulting in reduction of overall processing time. This technique uses Lagrange's interpolation to envisage the succeeding access count of the data files, consequently to decide about whether to create a new replica or to use dynamically the loaded block of data in place of cache in order to optimize replication factor. The experimental results illustrates that mapping phase completion time shrinks by 9.6%, map tasks with node locality grows by 6.1%, decrease of map tasks by 45.6% in rack locality and 56.5% in rack-off locality. To interpret problem of data locality in MapReduce from perspective of network [77] introduced a joint scheduler for scheduling and routing with an aim of utilizing the resources and network by pre-processing the routing information of few tasks, and not waiting for some idle machine to demand it. This scheduler can work efficiently with any workload within the determined capacity region. It considerably increases the throughput by more than 30% and minimizes delay for different workloads.

Pause-resume preemption [78] algorithm, which first time considers both Map and Reduce functions to improve execution time and data locality. It safeguards schedulers to choose among kill and wait processes. Furthermore, the preemption increases the overhead but it has improved map tasks locality by nearly 5%. This algorithm performs better for the circumstances where the share changes continuously because of appearance of fresh jobs. Based on prefetching the resources, [79] gives a novel scheduling policy to improve data locality by evaluating left over time to finish some task. It pre-fetches the resources for a remote map tasks with some overhead for disk space and network to provide better data locality. The outcomes with experimental setup of 4 computers, 1-JobTracker, 3-TaskTrackers and Hadoop0.20.2 on dataset WordCount, the algorithm illustrates enriched data locality of map tasks by nearly 15%, and a little bit reduction in response time of a job.

To increase overall working of MapReduce along with data locality and task completion time in heterogeneous cluster environments [80] offers techniques to contribute following:

i. Every node implements dynamic data partitioning regardless of various nodes in network.

ii. Particular physical machine is allocated reducer on the basis of virtual machine's readiness and size of partition.

iii. Offers priority to those specific reducers that do not obtain proportional data from particular machine.

By considering speed of every virtual machine the reducers with high workloads are allocated to those with fast processing speed.

A new task scheduling method [81] proposed for MapReduce framework in Hadoop which emphasizes to increase both locality of cache and locality of data along with minimizing the data transfer cost for task operation. The relationship among tasks and resources is given by a selection matrix in addition with bipartite graph according to locality of data. Experiments conducted in environment consisting one NameNode and 3-racks, Ubuntu 14.04.1, and Hadoop 2.3.0 with every rack comprises two DataNodes results in efficient improvement in data locality and cache locality.

A novel dynamic IaaS architecture [63] for Hadoop cluster designed by adding the features: monitoring, scheduling, Virtual machine management and Virtual machine migration as solutions to scheduling resources and data locality. The monitoring unit gathers load of Virtual machines and physical hosts for fabricating and fixing data locality and resource scheduling. Resource scheduling based on load-feedback succeeds in solid scaling for virtual clusters by varying the count of virtual machines. These techniques results in overall better system and efficient load balance.

A multi-objective model [82] proposed with an intend to build up the connection between assignment of resources and scheduling of jobs for streamlining MapReduce task scheduling in the cloud considering minimizing cost and time completion models. It offers enhanced throughput using low latency and reduced runtime of complex jobs in a parallel environment. Furthermore, this model attains a prediction of MapReduce job with high probability on cloud using the multi-objective scheme to confirm noble trade-off conclusions.

Using algorithm [83] execution time variation of Map tasks' in MapReduce framework, as a result of data skew and intensified by interference of virtual machines, can be handled. Randomly dividing tasks among subsets, proposed method result in reducing virtual machine cost and further reduced execution time by 46.7% when contrasted with some past methodologies. Experiments were performed in homogeneous cluster environment with assumption that standard deviation of task and mean are identical for each node. Consequently, heterogeneous groups were not considered because of need of evaluating the normal distribution of assignments for every arrangement independently.

2.1.3 Handling Non-Local Executions

Task scheduling in MapReduce prefers data blocks to be processed where they reside. However, it is not possible in many cases. For instance, when slave nodes do not have any free slots it cannot accept any more map tasks to process data blocks locally. So, data blocks are transferred over network, which incurs communication cost. To investigate more about non-local executions, this section elaborately discusses the past works that focus on improving the number of data local executions, thereby minimizing network bandwidth, map task latency, and job latency.

Big data challenges and some of the big data frameworks are surveyed in [84] to show their importance in real-world applications. Authors did a comparative study and have presented an experimental evaluation based on batch and iterative workloads for machine learning, graph processing, and stream processing applications. Dongjoo Choi et al. [85] proposed a data locality classifier considering the location of all data blocks that constitute IS. After the classification, map tasks are scheduled sequentially based on the number of non-local executions, which is denoted as priority. Authors have claimed that the proposed algorithm improved total processing time and data copying frequency up to 25% and 28% respectively when compared to the classical MapReduce scheduler. A novel data placement technique [86] is proposed to maximize the MapReduce scheduler performance in virtualized cloud environment. Authors devised a data block placement model as NP hard problem to minimize the unexpected global data transfer cost by modelling replica balanced distribution tree. Results were compared based on data locality and overall data transfer cost.

Data blocks placement and intermediate shuffle phase data transfer are discussed in [87]. Authors also considered cost, deadline, and energy for scheduling and provisioning the frequently executed MapReduce job in cloud. They proposed two heuristic algorithms for scheduling tasks and allocate resources for MapReduce jobs in cloud and compared its performance with classical MapReduce schedulers.

Ankita Atrey et al. [88] discussed data block placement in geo-distributed cloud using spectral cluster on hypergraph. They initially calculated spectral approximation for finding approximate low-rank solution from hypergraph. Then, authors devised an algorithm (spectradist) to extract a list of data blocks to place across clouds in distributed environment. Typically, hybrid cloud scales geographically to huge network, which incurs huge bandwidth consumption cost when data blocks are moved across clusters. A similar work is done in [89] to solve data placement problem in geo-distributed cloud environment for data-intensive applications. Authors devised an algorithm "Virtual Data Agent" to minimize the transmission time of data blocks across data-centre. They mapped data blocks to virtual data agent, which in turn maps them to data-centre. The proposed algorithm minimized 5%-20% data transmission overhead between the data-centre.

A scheduling aware data prefetching is introduced by Chunlin Li et al. [90] for hybrid cloud data transfer to minimize non-local map tasks by using bandwidth of idle network. Authors determine the popular data blocks to cache for the repeated job execution. The proposed algorithm was compared with capacity and fair scheduler to show its effectiveness. Replication helps not only mitigate the data loss due to node failure or corruption and also minimizes the network bandwidth consumption when more non-local execution happens. In generl, replication factor is statically set. But, a dynamic prefetching-aware bock replication is proposed in [91] by N. Mansouri et al. Authors determine which blocks of the dataset to replicate at present by using previous log files. However, high replication factor occupies more storage space resulting to store a smaller number of original data blocks. To handle this, a fuzzy inference system is used considering parameters such as number of accesses, cost of replica, last time data block accessed, and availability of data blocks. Authors claimed that response time of tasks improved up to 35% compared to other classical algorithms.

To minimize data block movement for non-local map task execution, Hemant

Kumar et al. [92] proposed data partitioning and placement aware computation scheduling scheme (DPPACS) algorithm based on data blocks availability. Authors clustered data blocks based on inter-dependency of task execution by building a graph for data block and map tasks running in geo-distributed cloud data-centre. Therefore, only data blocks required by map tasks are copied to certain cluster. This way authors minimized the number of non-local executions. Resource allocation for data-intensive tasks is always a difficult task as it is uncertain how the resource usage scales up or down. In cloud virtual environment, data transfer is critical as many users share the virtual network. Authors in [93] devised a scheduling strategy for distributing data-intensive virtual machines based on the characteristics of job running in the virtual machines. Similarly, to handle real-world skewed data distribution, DALM (Dependency-Aware Locality for MapReduce) was presented in [94] by Xiaoqiang Ma et al. Even though replicating blocks multiple-times could minimize the number of non-local execution, this idea enormously consumes disk storage. Copying data blocks frequently around the cluster cause block thrashing, which is not desired in a multi-tenant environment. Moreover, heterogeneous performance of nodes in virtual environment affect performance of scheduler. To address this, Li Chunlin et al. [95] proposed a dynamic multi-objective optimized replica placement and migration strategies for cloud environment. Authors focus on improving network utilization, response time, and balancing data node storage. Despite replication of data blocks improve the distributed system performance, replica placement policy with budget and deadline constraint affects energy of cloud data-centre in [96]. Similarly, Rihab Derouiche et al. [97] proposed a data placement strategy based on formal concept analysis to minimize data block movement, overall network bandwidth consumption, and more specifically energy consumed by servers. Authors considered various parameters such as input dataset, communication levels (switches, routers), and resources used in the cluster for analysis. The aim of the authors was to group datasets and tasks into servers that are closer thereby minimizing network bandwidth consumption.

A centralized mapping strategy [44] is proposed to minimize the inter-rack communication cost by cutting down the NNLE in a virtualized and heterogeneous environment. Authors logically divided map and reduce tasks of different jobs and form groups to improve data local executions and minimize communication cost. Based on the number of data local executions of different jobs, map tasks are grouped and scheduled. Based on the communication cost, reduce tasks are grouped and scheduled.

A hybrid scheduling algorithm, HybSMRP, is proposed in [98] to improve data local execution and job latency. Authors proposed two techniques to achieve their objectives: dynamic priority and localization ID. Dynamic priority helps to determine which tasks from which jobs should be assigned to the available resource node. Localization ID is assigned to each node in the cluster to get fair amount of data local executions. Chunlin Li et al. [99] proposed a replica-aware task scheduling and cache mechanism to improve job latency and minimize unnecessary replications. Initially, non-local executions for respective blocks and frequent failed tasks are traced from the logs to identify where repeated executions happening in multi-cloud heterogeneous environment. To minimize makespan and improve resource utilization for a batch of MapReduce jobs in heterogeneous environment, a novel task scheduler is proposed in [100]. It includes two policies HaSTE, HaSTE-A for YARN distributed system. The scheduler assigns resources to the tasks whenever resources become available based on tasks urgency and fitness, especially, for iterative jobs. To improve resource utilization and job latency, DRMJS is proposed in [39] to exploit heterogeneous performance in a virtualized environment. DRMJS calculates the performance score for map and reduce tasks separately. Based on the performance score, map and reduce tasks of different jobs are scheduled. There are various classical MapReduce scheduling strategies (FIFO, Capacity, Fair) discussed in [101]. FIFO schedules jobs and allocates entire cluster resources in sequential order as they arrive. Capacity scheduler shares the cluster resources in different proportions based on the requirement of each job. Fair scheduler shares resources equally among all the jobs that are currently running. These schedulers do not consider heterogeneity and frequent non-local executions.

Chi-Ting Chen et al. categorizes a batch of MapReduce jobs into two groups (CPU-intensive and IO-intensive) using "dynamic grouping integrated neighbouring search strategy" [102] to improve resource utilization and number of data-local executions in heterogeneous computing environment. There are four phases in this proposed method. Phase 1 classifies the MapReduce jobs into two groups. A ratio table is created in Phase 2 for both Task Tracker in MapReduce and Data Node in HDFS. Phase 3 groups a set of data blocks and map tasks. In phase 4, neighbouring approach is used to schedule tasks that consume CPU and IO separately. Data locality and resource utilization aware scheduler is proposed in [103] to save energy cost in heterogeneous cluster. Authors proposed a framework that contains three modules: constructing task list, scheduling, and updating task list. Fuzzy logic is used to calculate the availability of slots in each node based on processor, RAM, and bandwidth availability for allocating tasks. Based on this scheme, data-local and rack-local executions are preferred. Once tasks are scheduled, the task list is updated for upcoming schedule using fuzzy logic. To improve the makespan and resource utilization, a heuristic method is proposed in [104] to estimate the MapReduce job latency. Firstly, log analysis is performed to profile the jobs already executed several times and understand the variables that affect the job latency. Then, a machine learning algorithm is used to estimate the execution time, which is used to calculate the makespan for a batch of jobs.

Improving data local executions also improves profit of service providers. Authors employed dynamic programming and ChainMap/ChainReduce in [105] to minimize data transmission time during MapReduce workflow execution. The proposed approach largely relies on data locality to minimize the job latency with frequent replications of data blocks on demand. A holistic scheduler was designed by Mohamed Handaoui et al. in [106] to improve resource utilization and job latency. It consists of three components: resource utilization prediction, determining data local executions, minimizing interferences from co-locating workloads. Authors have demonstrated the proof of concept with the help of constraint programming, genetic algorithm, and local search-based algorithms.

2.1.4 Data Block Placement

Due to the prevalent use of cloud-based data processing service, managing virtual resources offered to the users is becoming difficult to improve MapReduce job latency and throughput. Therefore, big data processing with tools offered by cloud is increasingly becoming a research hotspot. Different block placement and MapReduce job/tasks scheduling play a vital role in improving job latency, and throughput. Over a decade, there have been many data block placement algorithms [107] [108] proposed to improve throughput and novel scheduling algorithms [8] to exploit dynamic performance to improve job latency in cloud environment. This section discusses some significant previous works on data placement and MapReduce job scheduling, focussing on heterogeneity.

Distributed file system such as Google File System [107] and Hadoop Distributed File System [109] divide input dataset into fixed size chunk, called blocks. These blocks are stored based on rack-awareness in case of physical cluster, which leads to different number of blocks in each physical machine in the cluster. But, in cloud virtual cluster environment, there is no rack-awareness and number of blocks in each virtual machine is same regardless of the performance of virtual machine. An efficient modified version hash algorithm [110] is introduced for block placement to minimize energy consumption and improve throughput in virtual cluster environment. It is done in terms of CPU-intensive, IO-intensive, and interactive jobs to justify the claim made by the author. It is also important to consider geo-distributed data-centres while placing data blocks. A 2-stage block placement and job scheduling strategy [111] is discussed by Shao-Wei Liu et al. for efficient workflow execution. The objective is to place the related blocks in the same data-centre by identifying data dependency among tasks in workflow build time. So, the map tasks of a job are not scheduled to many data-centres. Results show that significant data transfer reduction is achieved in virtual network. Similarly, if huge dataset is stored in multiple data-center, running workflow involves large amount of data transfer among data-centres during shuffle phase. Therefore, a k-means based data block placement strategy [112] for scientific workflows is proposed by Dong Yuan et al. The downside of this approach is that algorithm does not maintain the information on load of each virtual machine in the virtual cluster deployed in different data-centres. Despite it could minimize the amount of data transfer during shuffle phase, latency of jobs is a big concern. Roulette wheel scheme-based data block placement and heuristic based MapReduce job scheduler are proposed [43] to improve number of data local execution for map tasks, job latency, and throughput. In this work, authors have considered computing capacity of virtual machines but not the heterogeneous performance virtual machines. More specifically, disk IO performance largely affects map phase latency even though CPU and memory are available for map tasks.

Heterogeneity in different level of cloud environment is considered before data block distribution in virtual cluster by using a framework, called MRA++ [113]. This method uses some sample map tasks to gather information on capacity and performance of individual node in the cluster. If a node is performing up to the mark than other nodes, then it does not attract tasks that are compute-intensive. Thus, straggler is avoided. Balancing data load among the nodes in Hadoop cluster is very difficult to determine in heterogeneous environment as performance of individual node highly varies. Therefore, performance is quantified [114] to place data blocks, such that, latency of job and throughput are improved. A novel collaboration-based scheduling strategy is proposed by Deng K et al. [115] for placing data sets and scheduling tasks based on the dependency that exists among them to improve MapReduce scheduler performance. This approach also tried to balance the load in each node and maintain the dependency for datasets loaded in different data-centre during execution. Zheng et al. [116] developed an intelligent data placement method to improve data local tasks thereby improving the performance of MapReduce scheduler of data-intensive applications. Genetic algorithm was used by authors to devise a 3-stage data placement strategy to minimize data transmission time across data-center. A novel data placement technique was proposed by Vrushali Ubarhande et al. [108] to minimize makespan in heterogeneous cloud environment. Computing performance is determined for each virtual node using some heuristics and data blocks placed accordingly. Authors claim that data locality and makespan improved improved, compared to classical methods. Chia-Wei Lee et al. [54] also proposed a similar work calculating the processing capacity of virtual machines in heterogeneous cloud environment to improve MapReduce scheduler performance. Workload type and processing capacity are identified to load data blocks accordingly. 23.5% of performance is improved when compared to classical MapReduce schedulers. To balance optimal load across virtual cluster, a topology aware heuristic algorithm was developed [86] focussing to minimize non-local execution for map tasks and minimize the global data access during shuffle phase. Authors claim that computation cost was minimized up to 32.2% when compared to classical MapReduce schedulers.

In production environment, a batch of jobs is periodically executed to extract insight from huge data in physical/virtual cluster in different time. Nature of jobs would not change mostly and reveal more information about the workload behaviour. Therefore, understanding them is very much important. Zujie Ren et al. [117] devised a workload generator, called Ankus, to perform performance evaluation and debugging in distributed environment for big data processing. They designed a scheduling algorithm, Fair4S, that gives importance for small jobs in the batch. They performed MapReduce workload analysis using the trace collected from Taobao, an online E-commerce enterprise to identify the size of jobs in the batch. In order to adapt various dynamic parameters in big data applications to improve energy efficiency, workload analysis [118] was performed to select the optimal configuration and system parameters. They used micro-benchmark and real-world applications to demonstrate the idea proposed. It is a study conducted to experiment with various processing elements along with system and Hadoop configuration parameters. These parameters highly emphasize the performance of MapReduce scheduler performance. Identifying right combination of these parameters is a challenging task which cannot be done at the time of execution. Therefore, Metric Important Analysis (ensemble learning) is done by Zhibin Yu et al. [119] using MIA-based Kiviat Plot (MKP) and Benchmark Similarity Matrix (BSM). This produces more insight than traditional based dendrogram to understand job behaviour by using three different benchmarks: iBench, CloudRank-D and SZTS.

2.2 Outcome of Literature Review

- **Scheduling Algorithms in Hadoop**

By default Hadoop attained three configurable scheduler policies: First Come First Serve (FCFS), Hadoop Fair Scheduler (HFS), and Capacity Scheduler policy. FCFS scheduler processes the jobs in accordance with their submission. Its major demerit is low resource utilization. It does not pave the way for reasonable sharing among users and furthermore deficit in response time for processing of minor jobs. Capacity Scheduler was envisioned and propounded by Yahoo to render the partaking of cluster possible among organizations. Therefore, to achieve unbiased sharing of cluster a minimum guaranteed capacity of the queues was set. Further Facebook planned HFS to moderately segment the cluster among different applications and users. Consequently, it ensures an evenhanded segmentation of the capacity of cluster over the time [11][21][46].

Drawback in default scheduling algorithms of MapReduce is that these algorithms assume that the nodes and environment are homogeneous. These scheduling algorithms randomly select the DataNodes for processing and storage. The period of time for MapReduce jobs fluctuate from seconds to days [61].

Table 2.1: Scheduling Policies comparison

Sr. No	**Scheduling Policy**	**Key Features**	**Challenges**
1	First Come First Serve (FCFS) Scheduler [11][21][46][65]	Resources are allocated as per arrival. Jobs are executed in the similar manner in which they are submitted. High Throughput.	Data Locality and starvation is reduced. Resources are not fairly distributed which leads to low resource utilization. Non pre-emptive. Suitable to clusters having Homogeneous Environment.
2	Capacity scheduler by Yahoo [11][21][46][65]	It brings about fare distribution of resources among different users with minimum assured capacity. High resource utilization and more rapidly response time.	Maximum capacity is need to set, to limit the number of users who can access the resources. It does not make sure the fairness and stability for pending jobs in queue. Performance decreases in Heterogeneous Environment.
3	Hadoop Fair Scheduler (HFS) by Facebook [11][21][46][65]	Individual task is confirmed with a rational portion of the resource. Provide reasonable share of the cluster capacity over time.	Does not consider the job weight for individual which results in unbalanced performance of nodes. Restriction on number of jobs to be placed in pool. Performance decreases in Heterogeneous Environment.
4	Longest Approximate Time to End (LATE) [64]	Ponders heterogeneity of cluster. Flourishes in refining the data locality.	Slightly negotiates fairness and reliability. Static scheduler.
5	Delay scheduler [66]	Ponders data locality issue. For execution of complex calculations no overhead is required.	If majority of tasks are considerably more than an average job then it is not effective. Not suitable for Heterogeneous environment. No resource sharing. Static scheduler.
6	Deadline Constraint Scheduler [65][120]	Originate least count criteria for map and reduce task. Specifies the deadlines to improve system utilization.	Not considered aspects runtime estimation for map and reduce task, filter ratio approximation, distribution of data and execution of more than one MapReduce tasks. Identical nodes are required which leads to more cost. Constraint about deadline for individual job is to be stated by user.

Based on up to examined literature, authors comprehend that there are a glut of challenges which are being encountered by Hadoop. The major challenges in Hadoop are Query execution time, Data movement cost, Selection of best cluster and racks for data placement, Preserving privacy, Overall load distribution to handle: Imbalance in input splits, computations, partition sizes and heterogeneous hardware, and Scheduling. The average interval of map and reduce tasks is different for each task and job depending upon the available number of TaskTrackers. The JobTracker assigns TaskTracker a task which is nearby locality to the TaskTracker. Each application is taken as job in MapReduce framework and a lot of map and reduce tasks constitute a job. A variety of factors like number of running jobs, wait time, response time, and run time plays an essential role while inducing the load in MapReduce. Hadoop scheduler makes use of Queue data structure to assigns the tasks [23]. Table 2.1 shows comparison on features of default and some other scheduling algorithms.

- **Data Locality in HDFS**

The Table 2.2, illustrates the Key features and drawbacks of scheduling algorithms that are aware of Data locality in Hadoop.

- **Handling Non-Local Executions**

Data block thrashing is the result of copying around the cluster, which cannot be minimized unless IS size set to low. As number of blocks in an IS increases, the non-local executions also increase. Even though non-local execution can be minimized with the help of too many replications, it is not suitable for huge dataset as it enormously wastes storage capacity. Despite the past works focussed to improve scheduler performance in terms of number of non-local executions, network bandwidth consumption, and job latency, there are few other factors listed below that can be considered to maximize the scheduler performance, by minimizing the

NNLE and ABC during data block movement across virtual networks, while facing too many non-local executions.

1. Heterogeneous performance virtual machines.
2. Caching data blocks that are frequent to copy over network.

- **Data Block Placement**

Existing works largely depend on finding the processing performance of virtual machine to place data blocks to improve data local execution. However, they do not consider performance degradation when many virtual machines are hosted in a single disk drive in a server. Moreover, performance of virtual machines fluctuate due to co-located virtual machines resource consumption interference. Therefore, the following works are proposed to increase number of data local execution, thereby improving MapReduce job latency and throughput for a batch of workloads.

1. Despite there are works to improve data locality in heterogeneous virtual environment, it is important to consider the disk IO performance before loading blocks. A simple linear regression algorithm is used to predict the performance of disk IO over time.
2. In addition, virtual machine performance varies due to co-located virtual machine interference. Therefore, there is a need to model dynamic performance of a virtual machine for launching map tasks.

Table 2.2: Comparison of Scheduling Algorithms

Sr. No	**Algorithm**	**Key features**	**Drawbacks**
1	Data placement strategy [28]	Considers Heterogeneity of nodes. It offers Enhanced Data Locality and improved Load balancing.	Suffers from data redundancy problem while allocating data in cluster. The mechanism for distribution of data is static.
2	Prefetching and Pre-shuffling schemes [68]	No wastage of system resources. Predicts the straggler. Reduces execution time in shared environment.	Performance reduces in when with complex load.
3	Parallel Task scheduler [69]	Regulates the time spent. Effective pre-emption. Work conservation. Improves Data Locality, scalability, and reduces virtualization overhead	It is expensive. The properties of isolation not strictly implemented. No appropriate tool for long running tasks.
4	Delay Scheduler [70]	Nearly optimum data locality and fairness for variety of assignments. Increases the throughput.	Does not allow resource sharing. It is suitable only for homogeneous environment. Not suitable when more number of tasks are heavy than average size of task.
5	NKS (next k-node) scheduling strategy [72]	It boosts Data locality. Capably manage and reduces network load.	Only suitable for homogeneous environment.
6	Distributed adaptive data replication algorithm [73]	Increase data locality. Minimum overhead of replicas and network traffic. Reduces turnaround time.	No progress is shown for the output-bound tasks by dynamic replication.
7	Predictive hierarchical fast spread (PHFS)[74]	Reduces access latency. Upturn the data locality. Efficient for such tasks where clients works on a particular framework.	Not suitable when clients requests are random.
8	Linear Sum Assignment Problem lsap-sched [75]	Provide optimal data locality at low cost. Able to Schedule multiple tasks simultaneously.	Performs well when resources at disposal are more.
9	Adaptive Data Replication Scheme [76]	Enhance data locality and minimizes data transfer time. Reduction of over-all processing time, rack locality and rack-off locality. Optimized node locality, replication factor.	Node locality drops as the size of data blocks is increased.
10	Joint Scheduler [77]	Advance routing information of some tasks results in better network load balancing. Improves throughput and delay performance.	Not able to include scheduling at job level.
11	Pauseresume preemption algorithm [78]	Improves execution time and data locality. Allows pre-emption of Map and Reduce functions.	Pre-emption increases the overhead.
12	Job scheduling algorithm based on data locality [79]	Improve data locality. Pre-fetches the resources for a remote map tasks.	Increases the overhead for disk space and network.
13	Load Aware Virtual Machine Mapper [80]	Divides input data dynamically to improve data locality, total job completion time and the Reduce-phase completion time.	Cannot decide on count of reducers to be used in MapReduce phase which results in more cost.
14	Map task scheduling method [81]	Progresses both data and cache locality. Minimizes the data transfer cost for task operation.	No consideration given to clusters current load which results in load imbalance.
15	Multi-objective algorithm [82]	Improve the workflow by minimizing cost and time. Efficient resource usage. Offer better throughput with minimum delay.	Increased number of tasks results in longer finishing time for job.
16	Locality and Interference aware scheduler [83]	Enhances data locality. Tackles the time variation when executing map tasks' in MapReduce phase. Execution time improves and cost of virtual machines reduced.	Suitable only for homogeneous clusters.
17	Data locality based scheduler [121]	Nodes allocated data blocks on basis of processing capacity. It is suitable for heterogeneous environment. Minimizes average job execution time and improves data locality.	Suitable for only small clusters.

Chapter 3

PBDBP: Prediction Based Data Block Placement

3.1 Introduction

Heterogeneous disk performance due to VMs hosted in the same disk affects data local execution and stores equal number of data blocks in each VM regardless of its performance. Therefore, heterogeneous performance in terms of disk IO is focussed in this chapter to improve data local execution, thereby improving MapReduce job latency and throughput for a batch of MapReduce jobs, in-order to improve amount of data processed per second. The significant contributions of this chapter are:

1. Predicting disk IO bandwidth of every disk in each physical machine before loading data blocks.
2. Scheduling map tasks of different jobs based on the heterogeneous performance of each VM.

The rest of the chapter is organized as follows. Section 3.3 provides some background on Hadoop MapReduce and the motivation that helps workout this problem. Subsequently, proposed methodologies are justified and explained in Section

3.4. Section 3.5 discusses the results and analysis of proposed methodologies focusing data local execution to improve MapReduce job latency and throughput, resulting in improved amount of data processed per second. Finally, a summary is drawn in Section 3.6.

3.2 Problem Definition

Improving the performance of Hadoop MapReduce in terms of amount of data processed by placing HDFS data blocks based on the workloads behavior in heterogeneous virtual environment to improve latency and throughput.

3.3 Background and Motivation

This section outline's the background concepts on the Hadoop big data analytics framework and motivations that inspired to work on this problem.

3.3.1 Hadoop MapReduce

When data size exceeds the storage capacity of a single machine, it is divided and distributed to multiple machines in the cluster. Hadoop [34] is a framework that includes a stack of distributed processing tools, as shown in Figure 3.1.

Hadoop Distributed File Systems (HDFS) helps to manage data across multiple computers in the cluster using streaming data access pattern. MapReduce is a distributed programming tool to process data stored in multiple machines and combine the result. HDFS and MapReduce can make use of commodity servers that are low in cost to store and process huge data. Early problem with Hadoop framework was, if MapReduce was deployed in a cluster, no other distributed processing can be installed. Therefore, tool like Yet Another Resource Negotiator

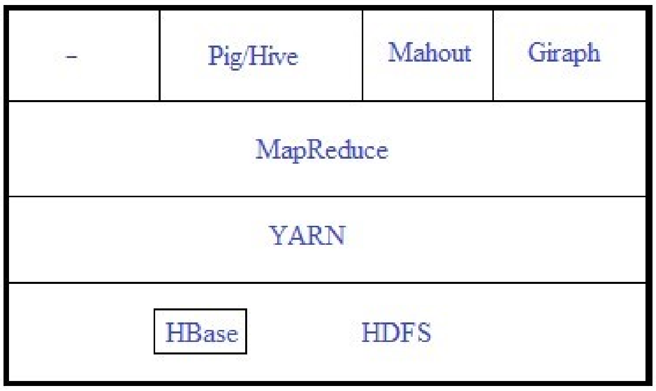

FIGURE 3.1: Hadoop Framework

(YARN) [39] is used to manage cluster resources and share them among multiple frameworks. Upon installing YARN, other distributed processing framework like Spark, Storm, etc. can be installed for various purposes. Once big data is uploaded onto HDFS, MapReduce jobs can be launched. A MapReduce job consists of two phases, as shown in Figure 3.2: map and reduce.

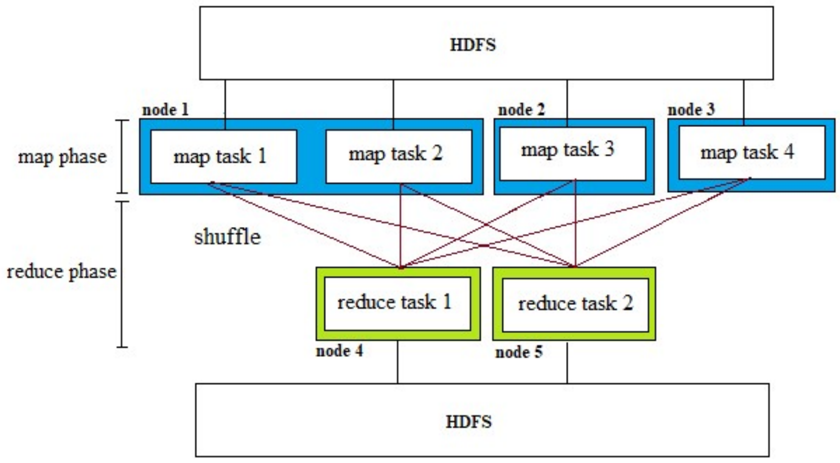

FIGURE 3.2: MapReduce Phases

A map phase executes a set of map tasks. Each map task processes a data block and produces an arbitrary number of intermediate outputs. Reduce phase executes a set of reduce tasks. Reduce phase can be started along with map phase but reduce function in reduce phase can be executed only after map phase is finished. Each reduce task collects its portion of input from each map task and consolidates them. Then, each reduce task goes through a sequence of steps (merge, sort, group) before reduce function gets executed. Reduce function gets a list of input from group function and writes output onto HDFS. In this execution sequence, moving output of all map tasks to each reduce task is called shuffle, in reduce phase.

3.3.2 Motivation

YARN comprises two major services: Resource Manager and Node Manager. Resource manager shares the pooled cluster resources (CPU, memory, storage) among different frameworks (like Hadoop, Spark). Consider two nodes, as shown in Figure 3.3, in which Node Manager (NM) manages local resources in the server.

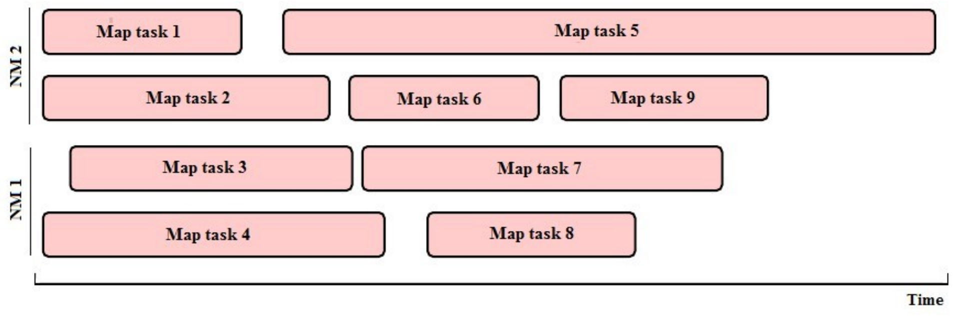

FIGURE 3.3: Map task completion

In map phase, many map tasks of a MapReduce job are executed in these two NMs. But, the latency of all map tasks need not necessarily be the same. For instance, latency of Map task 5 is three times higher than Map task 1. One of the reasons is heterogeneous performance of VM due to disk bandwidth sharing. So, Map task 5 must wait for a long time to bring its data block to process, which causes map phase to be longer despite all other map tasks are completed. To avoid this, non-local execution must be performed. However, map task must still wait for data blocks to read from disk for performing non-local execution. Therefore, understanding the dynamic consumption of disk IO bandwidth is very important before loading large data set into virtual cluster to improve map task latency, thereby improving throughput for a batch of workloads.

Another important thing is to understand the reduce task completion in reduce phase. As shown in Figure 3.4(a), consider four map task to be executed. Once all map tasks are completed, copy phase (also called as shuffle) starts to move map tasks output to node where reduce tasks are running. After shuffle process is executed for each reduce task separately, sort function is executed to facilitate reduce function for smooth running. At the time of second reduce task running reduce function (Figure 3.4(b)), other reduce tasks are still in copy phase. Similarly, third and fourth reduce tasks still run slowly with copy phase (Figure 3.4(c)) while the first two reduce tasks run into completion. Finally, all reduce tasks finish its execution (Figure 3.4(d)). The longest latency of reduce task leads to huge job latency. This is because, either reduce tasks run in node where the resource requirement of reduce tasks is not met, or dynamic performance of VMs causes reduce tasks to run slower. Therefore, map tasks should be executed where bandwidth is not a constraint, so that map tasks produce map outputs seamlessly to node where reduce tasks are running. This helps all reduce tasks to finish its execution almost in equal time.

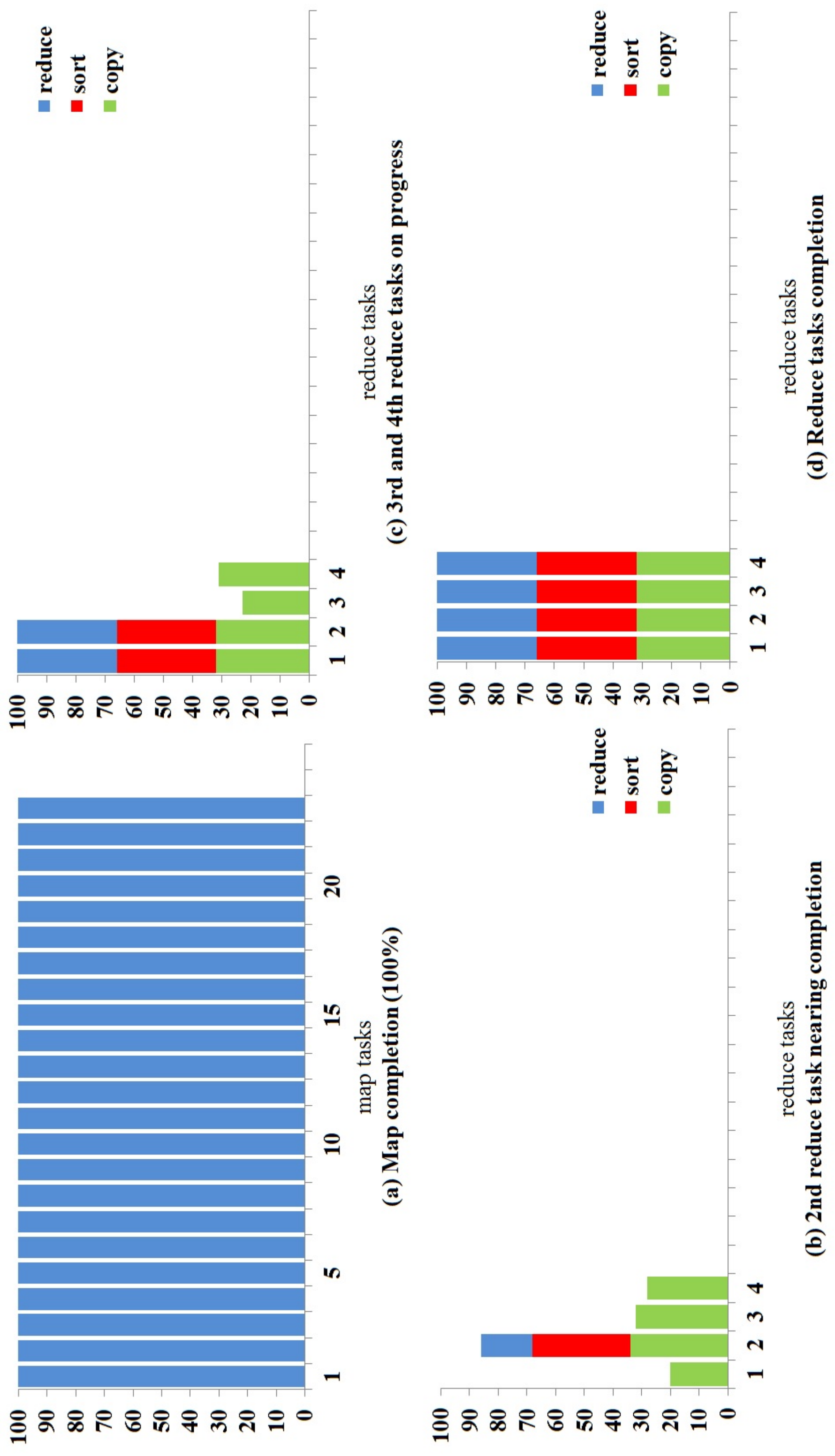

Figure 3.4: Reduce task completion

3.4 Proposed Methodology

3.4.1 Predicting disk IO performance to place data blocks using regression

Hadoop is offered as a service on a cluster of VMs, hosted in different physical servers in cloud data-centre, as shown in Figure 1.5. There are many racks (Rack1, Rack2, etc.) in cloud data-centre each with different physical machines lodged (node1, node2, etc.). Each physical machine (node) hosts many VMs (VM1, VM2, etc.). These VMs can be of Hadoop/non-Hadoop VMs. Hadoop VMs are highlighted with shade and clustered virtually. HDFS services such as NameNode (NN), Secondary NameNode (SNN), and Data Node (DN) run in separate nodes. Similarly, YARN components such as Resource Manager (RM), and Node Manager (NM) run in different nodes. These Hadoop VMs in the virtual cluster are spread across cluster wherever physical resources are available. These physical servers can be of different capacity and performance. Therefore, heterogeneity in physical environment is unavoidable. Thus, it causes various heterogeneity [39] for the same task in different time. Sometimes, VMs hosted in a node may be allocated in same hard disk drive, despite there being many disks attached. Even though CPU and memory are available for launching map tasks, due to contention created by co-located VMs in the same hard disk drive, it takes time to bring data blocks into memory for map tasks. This affects overall job latency and the number of local tasks. Therefore, it is important to consider hard disk drive performance before loading large data blocks. So the disk IO bandwidth rate is predicted over time to decide the target disk for loading data blocks. The detailed steps are given in Algorithm 1, as shown in Figure 3.5. Mostly Hadoop is hosted in a cluster of VMs for easy provisioning/deprovisioning, which adds the complexity of capturing the virtual resource usage. However, Hadoop VM resource consumption is observed by calculating the percentage of usage at point of time (Hadoop resource consumption/overall resource consumption). As it is periodic, so it is predicted using regression. The number of disk IO access (read+write) is directly proportional to the amount disk IO bandwidth has consumed most of the time in cloud environment, because applications running in the VMs exhibit a similar behaviour of disk access. As a result, the simple linear regression algorithm, which is a supervised learning model that aids in numeric prediction, is applied. It takes an input variable (independent variable) and produces an approximate/estimated output (dependent variable), as shown in Eq. 3.1.

$$predicted\_disk\_IO_i^k = mx + c \tag{3.1}$$

Here, i, k, and x indicate physical machine, disk number, and input value respectively. Therefore, $predicted\_disk\_IO_i^k$ indicates k^{th} disk in i^{th} physical machine. In our case, we used "number of disk IO access (read+write)" as input variable recorded every five seconds. While reading/writing happens in disk, some amount of data is transferred back and forth by consuming disk bandwidth. Even though one cannot predict the behaviour of applications running inside non-Hadoop VMs, using bandwidth consumption over time, but one can approximate the amount of disk bandwidth consumed. So, here "bandwidth consumption rate in %" is used as output variable. This is calculated by averaging the disk IO consumption rate of every second up to five seconds. By recording thousands of such samples, a linear model is built using regression algorithm.

In general, VMs are not migrated very frequently. Therefore, the predicted IO rate can be useful maximum time. Prediction is done for each disk in the machine and for all machines in the data-centre. Despite many Hadoop VMs are hosted in the same hard disk drive, data blocks are stored in drive which gives high performance. By default, three copies of a data block are stored across virtual cluster. What if three copies are stored in three different VMs hosted in same physical machine? The aim of multiple copies of same block is to achieve fault-tolerance. But there is no way to ensure rack awareness to achieve fault tolerance in virtual cluster. Therefore, after predicting the IO performance of individual disk, it is guaranteed that no copies of a block are stored in VMs hosted in same physical machine. Moreover, if disk IO performance is predicted to be high, then the number of blocks stored in that VM is high, as given in Eq. 3.2. In general, the number of data blocks stored in each VM is equal. But, in the proposed approach, the number of data blocks stored in a disk for a VM is directly proportional to the performance of disk IO in that machine. If performance is 100%, then exactly equal number of data blocks is stored in that VM.

$$\begin{aligned} &\forall_{i,j,k} \quad (No.\_of\_blocks\_stored)_{i,j}^{k} \propto \\ &(predicted\_disk\_IO)_{i}^{k} * equal\_share\_of\_blocks \end{aligned} \tag{3.2}$$

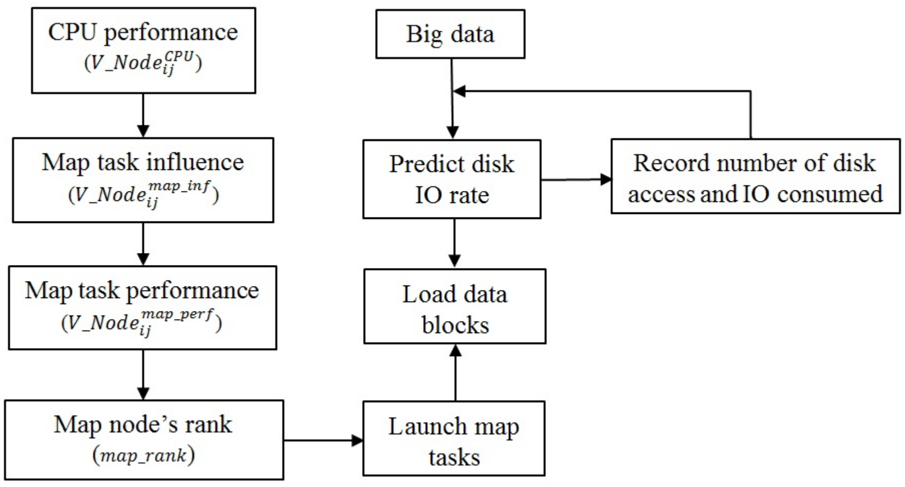

FIGURE 3.5: Prediction-based data block placement

3.4.2 Scheduling map tasks based on dynamic performance of VMs

Typically, Hadoop VMs are co-located with non-Hadoop VMs, which affect the performance of Hadoop VMs by sharing the IO resources of underlying physical machine. Even though performance of disk is high, there could be a chance that CPU and memory are tightly shared by tenants. Therefore, it is also important to observe that data local execution is minimized due to unavailability of resources. So, the performance of each Hadoop VM's is dynamically monitored, as shown in

Figure 3.5 and given in Algorithm 1. If performance of a VM is not as desired, then frequently accessed data blocks from that VM are migrated to other VM that gives high performance. Thus, there is an opportunity to increase the number of local executions, thereby improving the latency and makespan for a batch of MapReduce jobs. To calculate the VM performance hosted in physical machines, we need to find the physical machine that has high CPU frequency. CPU performance of j^{th} virtual machine (V_Node) in i^{th} physical machine (P_Node) ($V\_Node_{ij}^{CPU}$) is calculated by finding the physical machine having maximum CPU frequency (CPU_freq) among all the physical machines in which Hadoop virtual machines have been hosted, as given in Eq. 3.3.

$$V\_Node_{ij}^{CPU} = \frac{V\_Node_{ij}^{CPU\_freq}}{max(\forall P\_Node_{i}^{CPU\_freq})} \tag{3.3}$$

Eq. 3.4 calculates the Disk IO performance, ($V\_Node_{ij}^{DiskIO}$), of j^{th} V_Node in i^{th} P_Node considering current disk bandwidth rate, ($V\_Node_{ij}^{curr\_disk\_band}$), of j^{th} V_Node in i^{th} P_Node over the disk bandwidth, ($P\_Node_{ij}^{Disk\_band}$), of k^{th} disk in i^{th} P_Node.

$$\forall i, j \quad V\_Node_{ij}^{DiskIO} = \forall k, \frac{\sum V\_Node_{ij}^{curr\_disk\_band}}{P\_Node_{ik}^{Disk\_band}} \tag{3.4}$$

Map/reduce tasks have different resource requirements. Map tasks demand more of CPU and storage accesses while reduce tasks need CPU and network bandwidth. Therefore, to launch map tasks in VMs, it should have seamless disk bandwidth while the job begins and seamless network bandwidth while moving map outputs to reduce nodes where reduce tasks are running. To find the virtual node which is suitable for running map tasks, the influence of j^{th} V_Node in ith P_Node for map ($V\_Node_{ij}^{map\_inf}$) is calculate by considering the latency of last z map/reduce tasks executed in j^{th} V_Node, using Eq. 3.5.

$$\forall_j, V\_Node_{ij}^{map\_inf} = min\left(\forall_z, \frac{map\_latency_{jz}}{\sum_{m=1}^{z} map\_latency_{jm}}\right) \tag{3.5}$$

Algorithm 1: Loading data blocks based on the predicted disk IO performance

```
1  Notations Used
2    $V\_Node-$ Virtual machine
3    $V\_Node_{ij}-$ $j^{th}$ virtual machine hosted in $i^{th}$ physical machine
4    $V\_Node_{ij}^{CPU}-$ CPU frequency of $j^{th}$ virtual machine hosted in $i^{th}$ physical
     machine
5    $V\_Node_{ij}^{DiskIO}-$ Disk IO usage of $i^{th}$ physical machine where $j^{th}$ virtual machine
     hosted
6    $V\_Node_{ij}^{map\_inf}-$ map task influence of $j^{th}$ virtual machine hosted in $i^{th}$ physical
     machine
7    $V\_Node_{ij}^{map\_perf}-$ map task performance of $j^{th}$ virtual machine hosted in $i^{th}$
     physical machine
8  Data loading stage
9  Input: Disk IO usage
10 Output: Predicted Disk IO performance
11 while true do
12     Independent variable= number disk IO access (read+write)
13     Dependent variable = average (disk IO rate in every second up to five seconds)
14     Predict disk rate using Eq. 3.1
15     Determine number of data blocks to load using Eq. 3.2
16     Redistribute blocks if too much variation after data blocks loaded
17 end
18 Launching map tasks stage
19 Input: $V\_Node$ parameters (virtual CPU, virtual disk, and virtual network),
     system-level parameters (Disk)
20 Output: $V\_Node$ performance in terms of map tasks
21 while true do
22     Calculate the CPU performance $V\_Node_{ij}^{CPU}$ using (Eq. 3.3)
23     Calculate the disk IO rate $V\_Node_{ij}^{DiskIO}$ using (Eq. 3.4)
24     Calculate the map task influence $V\_Node_{ij}^{map}$ using (Eq. 3.5)
25     Calculate the map task performance $V\_Node_{ij}^{map\_perf}$ using Eq. 3.6
26     Find the rank list of map nodes by sorting $V\_Node_{ij}^{map\_perf}$ in descending order
         to find map_rank using (Eq. 3.7)
27 end
```

Typically, overall performance of a V_Node is calculated regardless of tasks type. Using Eq. 3.6, the map task performance ($V\_Node_{ij}^{map\_perf}$) in each V_Node based on CPU frequency and Disk IO bandwidth of respective V_Node hosted in each P_Node is determined.

$$\begin{aligned}\forall_{i,j}, V\_Node_{ij}^{map\_perf} &= V\_Node_{ij}^{CPU} \times \\ (1 - V\_Node_{ij}^{DiskIO}) &\times (1 - V\_Node_{ij}^{map\_inf})\end{aligned} \tag{3.6}$$

Finally, virtual machines are sorted, using Eq. 3.7, based on its performance to launch map tasks, else reduce tasks could be launched in place map tasks.

$$map\_rank = sort(V\_Node_{ij}^{map\_perf}) \tag{3.7}$$

This *map_rank* list is used to schedule the map tasks of nth job J_n as given in following procedure.

```
Launch maptask based on performance of virtual machine

Notations used:
n_map − number of map tasks of a job $J_n$
$map\_task_{np}$− $p^{th}$ map task of $n^{th}$ job
$\forall_p, map\_task_{np} = 0$
$C\_map_n = 0$− number of completed map tasks of job $J_n$

Input:   map tasks of a job, map_rank of virtual machines
Output:  assign map tasks to the right $V\_Node_{ij}$

while $C\_map_n < n\_map$ do
    Pick up a map task (p) from task list
    if $map\_task_{np}$ == 0 then
        Choose top 20% VMs from $V\_Node_{ij}^{map\_perf}$ rank list
        while until 20% nodes do
            if containers possible for $map\_task_{np}$ && local execution then
                $map\_task_{np}$=1
                Launch map task, $C\_map_n$++
                break
    else if $map\_task_{np}$ == 0 then
        Choose rest 80% VMs from $V\_Node_{ij}^{map\_perf}$ rank list
        while until 80% nodes do
            if containers possible for $map\_task_{np}$ && local execution then
                $map\_task_{np}$=1
                Launch map task, $C\_map_n$++
                break
    else if $map\_task_{np}$ == 0 then
        perform non-local execution
        $map\_task_{np}$=1
        Launch map task, $C\_map_n$++
    else
        add $map\_task_{np}$ into task queue
```

Here map_task$_{\mathrm{np}}$ is p$^{\mathrm{th}}$ map task of n$^{\mathrm{th}}$ job and it's initial value is 0 as no map tasks are scheduled yet. In the beginning, uppermost 20% of map nodes from the *map_rank* list is preferred for scheduling the map tasks. If containers are possible for map task and local execution, then map task is launched. Otherwise if there are not sufficient resources available to form container and then remaining 80% of the map nodes in the *map_rank* list is considered to schedule the map tasks. But performance of these 80% of nodes' for map task would be less when compared with topmost 20% of nodes in the *map_rank* list.If local execution is not possible, then non-local execution is performed. If at any moment there is no possibility for a map task, it is added back to the task queue. When *map_rank* for virtual machines is calculated, map tasks are scheduled by MRAppMaster. Figure 3.6 depicts a flowchart for steps of PBDBP.

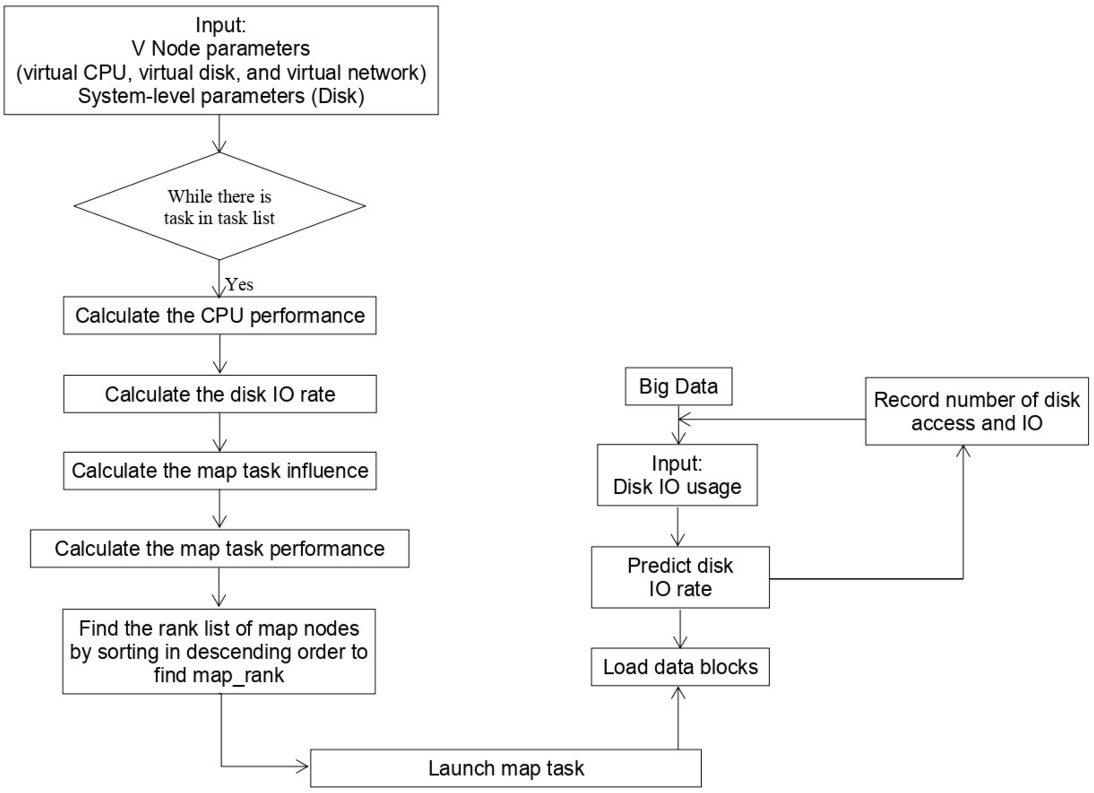
Input:
V Node parameters
(virtual CPU, virtual disk, and virtual network)
System-level parameters (Disk)
While there is task in task list
Yes
Calculate the CPU performance
Calculate the disk IO rate
Calculate the map task influence
Calculate the map task performance
Find the rank list of map nodes by sorting in descending order to find map_rank
Launch map task
Big Data
Record number of disk access and IO
Input:
Disk IO usage
Predict disk IO rate
Load data blocks

Figure 3.6: Flowchart for PBDBP

3.5 Performance Evaluation

3.5.1 Experimental Setup

Proposed methodology ideas are simulated in the Ubuntu server with 12-core CPU (hyper-threaded), 64 GB memory, storage 4 x 1 TB HDD and disk bandwidth rate 100 MB maximum. Proposed ideas are compared with classical scheduler [122], and [43] based on the quality of service parameters such as number of non-local execution, MapReduce job latency, and throughput, using Hadoop 2.7.0. Assumptions about workload size and virtual machine's configuration parameters as discussed below. Physical servers and VMs in cloud data-centre are assumed to be highly heterogeneous. Moreover, these VMs can be of different flavours (very small (1 vCPU, 2 GB memory), small (2 vCPU, 4 GB memory), medium (4 vCPU, 4 GB memory), large (8 vCPU, 16 GB memory), extra-large (12 vCPU, 24 GB memory)). Hundred VMs (20 VMs in each flavour) are considered and deployed across racks in cloud data-centre. The disk IO interference by co-located VMs is simulated in random. Workloads considered are wordcount, wordmean, wordmedian, and kmean to process datasets of size 128 GB, 64 GB, 256 GB, and 192 GB respectively, altogether 640 GB in total. Wordcount job finds the frequency of word occurrence in a file. Average length of words is calculated by wordmean job. Median length of words in a file is calculated by wordmedian job. Kmean job finds clusters from the given input data file. The work environment is static and tasks are independent. For all datasets, input block size is 128 MB and the replication factor is 3. The number of map/reduce tasks and its resource requirements is given in Table 3.1. The number of map and reduce tasks are given in Table 3.1.

One map task is assigned for a block, such that the number of map tasks for each job is 1000, 500, 2000, and 1500 respectively. Map task latency and reduce task

TABLE 3.1: Resource requirements of each job

MapReduce Job	No. of map tasks	No. of reduce tasks	vCPU		Memory		Map task latency	Reduce task latency
			map	reduce	map	reduce		
wordcount	1000	20	1	1	2	1	21	39
wordmean	500	15	1	2	1	1	18	33
wordmedian	2000	15	1	2	1.5	2	15	30
kmean	1500	10	2	2	1.5	2.5	21	60

latency are also fixed and given in the table. These latencies are approximated and taken from our lab experiments.

3.5.2 Results and Analysis

To predict the performance of disks IO rate, the trace of disks IO access recorded every five seconds is used . Consider three physical machines (nodes) each with one hard disk drive. Assume first node hosting five VMs, second node hosting three VMs, and third node hosting one VM. IO intensive applications like web applications are run in VM to record the disk IO access behaviour, as shown in Figure 3.7.

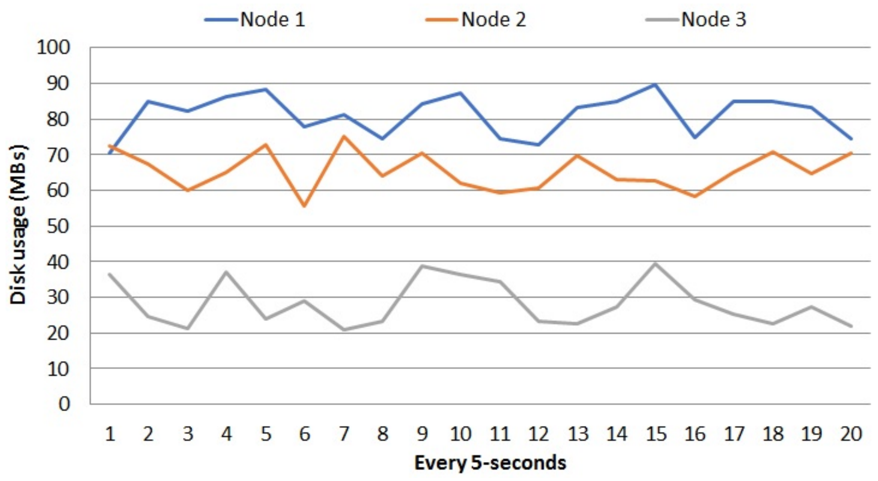

FIGURE 3.7: IO access pattern in different hard disk drives

Disk access of Node 1 fluctuates between 70 MB/s and 89 MB/s because of hosting five VMs. Each VM races each other holding disk access such that disk IO is busy always delivering data to running tasks. If many map tasks are launched in Node 1, then bringing data blocks into memory takes more time. Moreover, data blocks might be in different sector/disk in hard disk drive. Similarly, in Node 2, disk access ranges between 55 MB/s and 74 MB/s as three virtual machines are hosted in this machine. Node 3 disk access varies between 20 MB/s and 39 MB/s. In general, classical scheduler places data blocks across virtual machines evenly, but the proposed method places data blocks based on the performance of hard drives. Figure 3.7 shows the record disk IO access for every 5 seconds. However, it is recorded over different time spans and used for predicting disk access in the future. Simple linear regression is used for predicting disk IO access pattern based on the observed data. As given in Table 3.2, Node 1 is predicted to use 80 MB/s in the future. Therefore, 80% of data is to be store in Node 1 from its equal share. Similarly, Node 2 and Node 3 are predicted to use disk IO access to be 65 MB/s and 30 MB/s.

TABLE 3.2: Disk IO rate in different machines

No. of observations	**Predicted Bandwidth Used (MB/s)**		
	Node 1	**Node 2**	**Node 3**
1152	80	65	30

Secondly, heterogeneous performance of VMs causes varying latency of same task in different VMs. As shown in Figure 3.3, if a map task execution lasts for a long time, then latency of map phase is extended. As batch of MapReduce jobs is executed periodically, data blocks could be shifted to VM with higher performance. Therefore, the number of map tasks completed per unit time by high performing VM is higher than the VM with large capacity but varying performance. So, VMs are dynamically monitored for varying performance and data blocks are moved from one VM to another VM. The proposed approach is compared with existing schedulers: Fair scheduler [122], Rowlett Wheel Scheme (RWS) based job scheduler and Heuristics based MapReduce Job Scheduler (HMJS) [43] based on the configuration mentioned in Experimental setup. The number of non-local executions by these schedulers with different workloads is given in Table 3.3. Figure 3.8

shows the number of non-local execution achieved with different schedulers for a batch of four different workloads.

TABLE 3.3: Number of data non-local executions

Job	**Fair scheduler**	**RWS**	**HMJS**	**Prediction-based**
wordcount	29	28	5	8
wordmean	12	41	21	2
wordmedian	73	49	38	31
k-mean	112	69	18	10

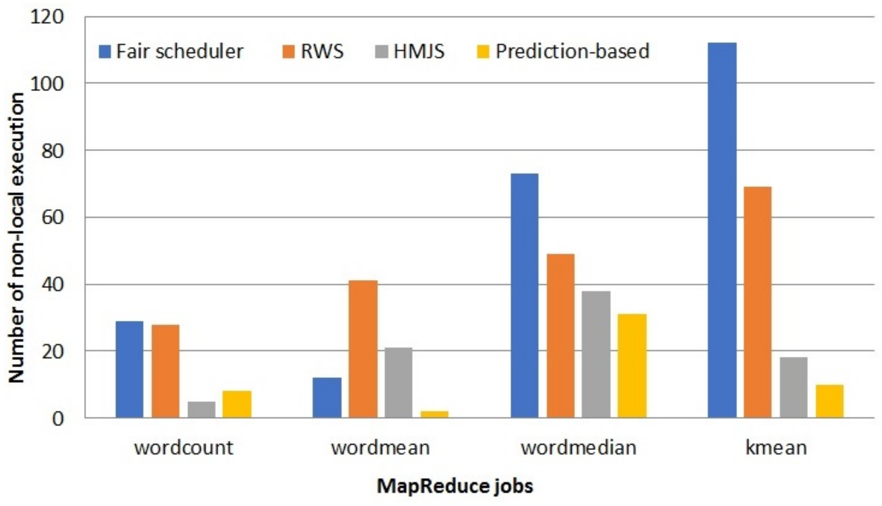

FIGURE 3.8: Number of non-local execution with different schedulers for different jobs

Prediction-based scheduler outperformed classical fair scheduler by 76% in average considering all four workloads. Similarly, 72% improvement in minimizing number of non-local execution in average is recorded when compared to RWS-based scheduler while using our proposed approach. Minimizing the number of non-local execution leads to reduction in job latency, as shown in Table 3.4 and Figure 3.9.

TABLE 3.4: Job latency(in seconds)

Job	**Fair scheduler**	**RWS**	**HMJS**	**Prediction-based**
wordcount	272	219	189	161
wordmean	219	223	172	148
wordmedian	573	380	332	221
k-mean	780	690	492	279

It is observed that job latency is minimized up to 49% in average while using our proposed method over classical fair scheduler. This is because, fair scheduler

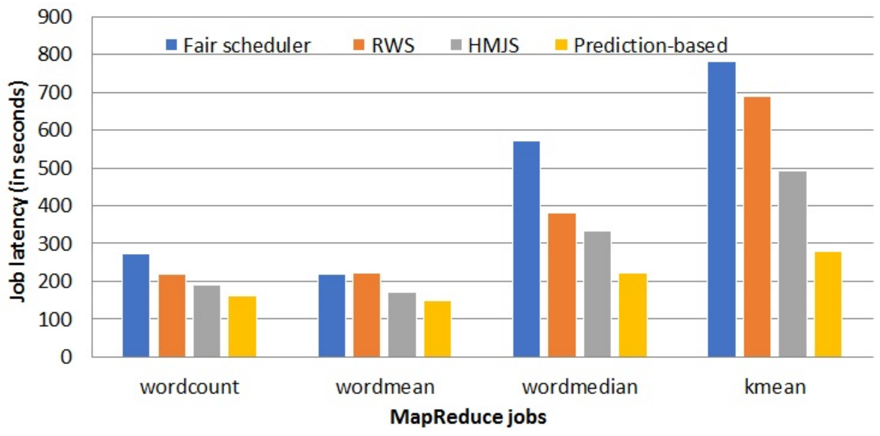

FIGURE 3.9: MapReduce job latency with different schedulers

partitions the resources for all the jobs, so number of local executions is very less resulting to increase in job latency (not the map task latency). Similarly, in average, 40% improvement is observed while using prediction-based approach when compared to RWS-based scheduler. It is because, RWS-based scheduler focuses on computing capacity of a VM while prediction-based approach considers the disk IO performance for making map phase latency to minimize.

Despite HMJS considering the heterogeneous performance to minimize job latency, prediction-based scheduler considers the number of disks available in each physical machine and its performance over time. Therefore, prediction-based scheduler outperforms HMJS by 26% only. Finally, to emphasize the effectiveness of the proposed method, makespan is also compared , as in Table 3.5, of each scheduler for a batch of jobs. Prediction-based scheduler improves makespan up to 66% when compared to traditional fair scheduler, as shown in Figure 3.10.

TABLE 3.5: Makespan of schedulers(in seconds)

Algorithm	**Makespan (in seconds)**	**Improvement (%age) over**
Fair scheduler	1189	66.6947
RWS	829	52.2316
HMJS	489	19.0184
Prediction-based	396	N.A

Similarly, it outperformed RWS-based approach by 52%. It is important to note that improvement of the proposed approach is 19% when compared to HMJS.

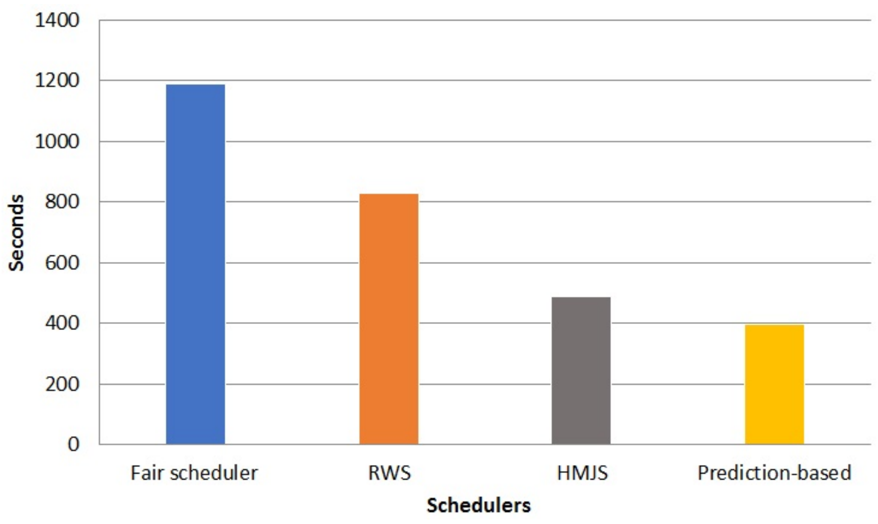

FIGURE 3.10: Makespan for different schedulers

HMJS used bin packing based approach that randomly fits the best combination of containers in each VM. In contrast, the proposed method takes different approach to load data blocks in advance to minimize map phase latency. When bin packing is focussed, data local execution is compromised to get the best combination of containers. Therefore, the number of non-local execution is steadily increasing while working with bin packing based approach in HMJS. In the proposed approach, this limitation is overcome by loading data blocks in the very beginning based on the performance of disk in physical machines where VMs have been hosted. Moreover, disk IO performance could be limited over time due to co-located virtual machines interference. So, the change of disk IO performance is observed periodically and the data blocks are moved accordingly based on the disk IO persistence. This significantly caused improvement in makespan when compared to HMJS.

3.6 Summary

Without cloud service, it is impossible to use big data processing frameworks as it is expensive to set up on-premise. Hadoop is one of the efficient processing tools for crunching a large volume of data. It is also offered as cloud service on a cluster of VMs hosted in a cluster of physical servers in a cloud data-center. In this case, heterogeneity in physical servers and VM performance are unavoidable. Therefore, the main focus is at disk IO performance due to many VMs hosted in a same hard disk drive. It highly affects map task execution in map phase due to delay in bringing data blocks from disk into memory. This motivated us to investigate the disk performance when VMs are hosted in same disk. To distribute data blocks based on the performance of disk IO, simple linear regression algorithm is used to predict disk IO performance and distribute data blocks accordingly. Therefore, data blocks for map tasks are brought into memory quickly. Moreover, varying performance of VM due to co-located virtual machine's interference affects MapReduce job latency. Therefore, heterogeneous performance is dynamically exploited, and if any high variation in performance for long time persists, then data blocks are redistributed in such a way that, data local execution is improved. At any time, the number of data blocks in a VM are stored based on the performance of that VM. The ideas are finally simulated based on Hadoop 2.7.0 and compared with classical fair scheduler, RWS-based scheduler, and HMJS. Results indicated that the proposed scheduler outperformed by 66%, 52%, and 19% when compared to those existing schedulers for makespan.

Chapter 4

IDLACO: An ACO based Map Task Scheduler

4.1 Introduction

Collecting big data is becoming more common in academia, industry, and research sectors. Hadoop [107] is one of the efficient big data processing tools for making decisions out of big data. Nowdays, Hadoop framework and relevant applications are offered as a service [36] on demand by various cloud service providers(CSP) over the internet, on-premise IT infrastructure for Hadoop MapReduce is not affordable for short-term users. CSP deliver MapReduce service to end users on different schemes hosted in virtual machine (VM).

- Private Hadoop MapReduce (pay per VM)
    - Purchase VMs from CSP and setup MapReduce manually.
    - Purchase MapReduce as a service on a cluster of VMs.
- Sharing MapReduce service with more than one user (pay per job basis)

MapReduce scheduler performance is highly affected if there is a greater number of non-local executions, especially in a virtualized environment. While offering

MapReduce as a service, virtual network bandwidth availability is always not guaranteed in a multitenant environment since it is shared. Besides, more virtual network bandwidth is consumed in the shuffle phase during the MapReduce job execution. It is a critical situation when the map phase of one job occurs along with the shuffle phase of another job. To overcome these situations and improve the MapReduce performance, in this chapter IDLACO is proposed to minimize the NNLE, thereby, minimizing job latency. Firstly, it minimizes the overall bandwidth consumption during job execution by finding a list of data blocks for each map task of a job to copy across the virtual cluster. Secondly, the target VM is determined based on its performance to copy the data blocks to perform a non-local execution. Since VM performance is heterogeneous, it is essential to dynamically determine the target VM. Finally, if a set of data blocks is copied frequently for repeated job execution, it is temporarily cached to avoid consuming bandwidth during job execution.

In summary, the proposed works in this chapter are listed below.

1. To calculate heterogeneous performance of virtual machines.
2. To find a list of data blocks for each IS that can minimize the number of non-local executions and bandwidth consumption using ACO.
3. To cache data blocks that are frequent in the target virtual machine.

Rest of the chapter is organized as follows. Section 4.2 includes problem definition and section 4.3 provides background and motivation behind this study. All the proposed methods are modelled and discussed in detail in Section 4.4 while Section 4.5 includes results and analysis of proposed methodology by comparing with fair scheduler and Holistic scheduler. Finally, summary is given in Section 4.6.

4.2 Problem Definition

Improving hadoop mapreduce performance by minimizing global virtual network bandwidth consumption to handle non-local execution to improve job latency by increasing data locality.

4.3 Background and Motivation

MapReduce task scheduling prefers data blocks to be processed where they reside. However, non-local executions are unavoidable when IS is assigned with more than one blocks, and required resources are not available in a node. To investigate more about data locality, non-local executions in a heterogeneous virtualized environment, some of the previous works are referred based on the parameters H, V, DL, IS, NLe, JL, and B, as tabulated in Table 4.1.

Table 4.1: Literature survey

PW	**H**	**V**	**DL**	**IS**	**NLe**	**JL**	**B**
[86]		✓	✓			✓	✓
[85]			✓	✓	✓	✓	✓
[90]	✓		✓		✓	✓	✓
[44]	✓	✓	✓		✓	✓	
[98]			✓			✓	
[99]	✓		✓		✓	✓	
[100], [102], [105], [106]	✓		✓			✓	
[39]	✓	✓				✓	
[103]	✓					✓	
[104]	✓					✓	
IDLACO	✓	✓	✓	✓	✓	✓	✓

PW- Previous Works, H- Heterogeneity ,
V- Virtualized, DL- Data Locality, IS- Input Split,
NLe- Non-Local execution, JL-Job Latency, B-Bandwidth

As tabulated in Table 4.1, most of the previous works try to target job latency by improving data locality in heterogeneous environment. In proposed method "IDLACO", all the parameters are considered to improve MapReduce scheduler performance. Fair scheduler and Holistic scheduler with genetic algorithm are used for comparing the results of proposed method.

4.4 Proposed Methodology

MapReduce scheduler performance is affected if there are a greater NNLE, especially in a virtualized environment. Thus, a methodology IDLACO is proposed, as presented in Algorithm 2, to minimize the NNLE and and amount of bandwidth consumed (ABC) during data block movement across virtual networks. IDLACO consists of the following steps:

1. Calculating the heterogeneous performance of VMs.
2. Modeling the NNLE and ABC.
3. Finding a set of data blocks for each IS using Ant Colony Optimization (ACO).

4.4.1 Calculating Heterogeneous Performance of VMs

Multiple users share the virtual machines commonly in the CDC. VMs are typically placed across racks in a CDC based on the resource availability. Thus, the performance of each virtual machine is based on the resource consumption of neighboring VMs. DRMJS [39] can be used to model the heterogeneous performance of VMs hosted in a physical server. DRMJS calculates performance of a VM based on map and reduced tasks separately. This study, as shown in Figure 4.1, considers only the performance and suitability of map tasks of a job along with the node attraction of data block to perform a non-local execution instead of using the map task performance in each VM. Along with the heterogeneous performance model proposed, few more components are introduced to maximize the map task performance.

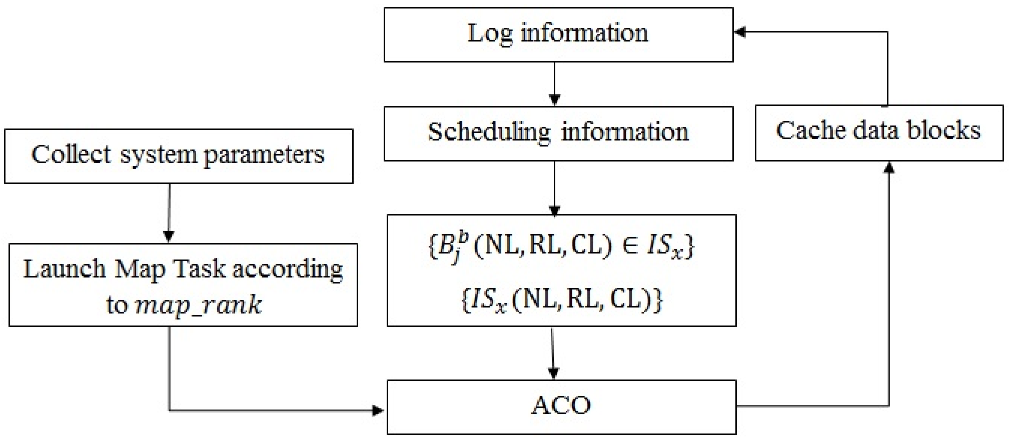
Log information
Scheduling information
Cache data blocks
Collect system parameters
Launch Map Task according to map_rank
$\{B_j^b(\text{NL}, \text{RL}, \text{CL}) \in IS_x\}$
$\{IS_x(\text{NL}, \text{RL}, \text{CL})\}$
ACO

FIGURE 4.1: IDLACO

First, as given in Algorithm 2, the performance of vCPU of a virtual machine is calculate. As given in Eq. 4.1, CPU performance of j^{th} virtual machine (V_Node) in i^{th} physical machine (P_Node) ($V\_Node_{ij}^{CPU}$) is calculated by finding the physical machine with a maximum CPU frequency (CPU_freq) among all physical machines where Hadoop VMs have been hosted. Besides, the performance of V_Node is based on the number of cores allocated from the total number of P_Node cores ($P\_Node_i^c$). Thus, the performance factor of V_Node in terms of the number of cores ($V\_Node_{ij}^c$) allocated to it is introduced.

$$V\_Node_{ij}^{CPU} = \frac{V\_Node_{ij}^{CPU\_freq}}{max(\forall i, P\_Node_i^{CPU\_freq})} \times \frac{V\_Node_{ij}^c}{P\_Node_i^c} \tag{4.1}$$

Since map tasks require huge disk IO interaction, the disk IO performance of j^{th} V_Node in i^{th} P_Node ($V\_Node_{ij}^{DiskIO}$) is calculate based on the current disk bandwidth rate of j^{th} V_Node in i^{th} P_Node ($V\_Node_{ij}^{curr\_disk\_band}$) over the disk bandwidth of k^{th} disk in i^{th} P_Node ($P\_Node_{ik}^{Disk\_band}$) (Eq. 4.2). Besides, it is essential to determine the capability of executing more number data local map tasks performed in a specific V_Node. To specify the disk IO performance based on the data block size, one more component, data locality (DL_i^j), is added in this equation to emphasize that a V_Node can run more data local map tasks over time. It indicates the number of data local executions in j^{th} V_Node hosted in i^{th}

P_Node.

$$\forall i,j \quad V\_Node_{ij}^{DiskIO} = \forall k, \frac{\sum V\_Node_{ij}^{curr\_disk\_band}}{P\_Node_{ik}^{Disk\_band}} \times DL_i^j \tag{4.2}$$

MapReduce tasks of a job have different resource requirements since users can configure it explicitly. A map task requires more CPU and storage accesses, while a reduced task needs CPU and network bandwidth. To launch map tasks in VMs, it should have seamless disk bandwidth while starting the job and seamless network bandwidth while moving map outputs to reduce nodes where reduce tasks are running. To find the virtual node suitable for running map tasks, the influence of jth V_Node in i^{th} P_Node for map ($V\_Node_{ij}^{map\_inf}$) is calculate by considering the latency of the last z number of map and reduced tasks executed in j^{th} V_Node, using Eq. 4.3. If the number of map tasks considered in a V_Node occurs differently, there is no point in considering them. However, as MapReduce jobs in a production environment are periodically executed, it is essential to consider the map tasks of the same job execution in the recent past.

$$\forall_{i,j}, V\_Node_{ij}^{map\_inf} = min\left(\forall_z, \frac{map\_latency_{jz}}{\sum_{m=1}^{z} map\_latency_{jm}}\right) \tag{4.3}$$

Using Eq. 4.4, the map task performance ($V\_Node_{ij}^{map\_perf}$) in each V_Node is obtained based on the CPU performance and Disk IO bandwidth of respective V_Node hosted in each P_Node.

$$\begin{aligned}\forall_{i,j}, V\_Node_{ij}^{map\_perf} &= V\_Node_{ij}^{CPU} \times \\ (1 - V\_Node_{ij}^{DiskIO}) &\times (1 - V\_Node_{ij}^{map\_inf})\end{aligned} \tag{4.4}$$

Finally, virtual machines are sorted, with help of merge sort, to prepare a rank list, using Eq. 4.5, based on its performance to launch map tasks, else reduce tasks could be launched in place map tasks.

$$map\_rank = sort(V\_Node_{ij}^{map\_perf}) \tag{4.5}$$

4.4.2 Modelling NNLE and ABC

The number of data blocks for an IS is configured before launching MapReduce jobs. When the number of data blocks in an IS increases, the NNLE increases. The number of data blocks to transfer over a network for a non-local execution can be intra-rack or inter-rack communication. If blocks are copied between the racks, it introduces more network traffic for other applications. Thus, aim should be to minimize the NNLE considering the racks in data center, which minimizes the ABC as well.

As discussed in Section 4.1, there are three types of data local execution: , NL, RL, and CL. If IS comprises s data blocks for a map task to process, then we must track the s data block from the previous execution log (Figure 4.1) of previous schedule. Using such information, it can decided whether a data block should be moved for non-local execution or to cache in the target VM. Block information is maintained as a triplet $B_j^b(NL, RL, CL) \in IS_x$. Each block b of a data-set that belongs to an IS x and resides in j^{th} VM contains information on how a block is executed (NL, RL, and CL). Consider an instance $B_2^5(NL, RL, CL) \in IS_4$. It means that the 5^{th} block that belongs to 4^{th} IS has been executed as NL in 2^{nd} VM. Here, the number of IS is equivalent to the number of map tasks. For instance, if there is 1 GB input data and the block size is 64 MB, then the number of blocks is 16. If an IS is configured to contain the data block of size 128 MB, it can comfortably include two physical data blocks. This forms 8 IS, which results in eight map tasks. Similarly, each IS information is represented as a triplet $IS_x(NL, RL, CL)$, indicating the information of x^{th} IS. If an IS contains n data blocks, then the number of data blocks executed as NL/RL/CL is denoted in the IS triplet. Consider $IS_4(1, 2, 1)$. It means that the 4^{th} IS contains four data blocks, where one data block is executed within the node, two data blocks are executed within the rack as non-local executions, and one data block is executed across racks as a non-local execution. Thus, the 4^{th} IS accounts for one local and three non-local executions. The residence of each block is denoted using the first triplet $B_j^b(NL, RL, CL) \in IS_x$ to calculate the virtual network traffic between the servers in a rack and between the racks in a cluster. Algorithm 2 elaborately explains this sequence.

Notations used in algorithm:

Input dataset (ds) – in GB

Block size (bs) – 64/128 MB

Number of data blocks (b) – ds/bs

Input Split (IS) size – >= 1 bs

Number of IS (y) – 1...x...y

Number of blocks in an IS (s) – 1...z...s

Type of locality – NL/RL/CL

V_Node- Virtual machine

P_Node- Physical machine

$V\_Node_{ij}-$ j^{th} - V_Node hosted in i^{th} P_Node

$V\_Node_{ij}^{CPU}-$ CPU frequency of j^{th} V_Node hosted in i^{th} P_Node

$V\_Node_{ij}^{DiskIO}-$ Disk IO usage of i^{th} P_Node where j^{th} V_Node hosted

$V\_Node_{ij}^{map\_inf}-$ map task influence of j^{th} V_Node hosted in i^{th} P_Node

$V\_Node_{ij}^{map\_perf}-$ map task performance of j^{th} V_Node hosted in i^{th} P_Node

$B_j^b(NL, RL, CL) \in IS_x$ – Data locality information for each b that belongs to IS

$B_j^b(NL, RL, CL)$ – IS information

REP_b – Replication of each block

$BC_{(j,k)bw}^d$ – Bandwidth consumption

From the log files, one can find the number of NL, RL, and CL for each map task of a job in the past. The, initial plans for map and reduce tasks are prepared by the scheduler. The number of non-local executions in each IS and the overall NNLE can be calculated using Eq. 4.6. This information is used, to find the VMs,

Algorithm 2: Improving Data Locality using Ant Colony Optimization (IDLACO)

```
1  Load input dataset
2  Find the number of data blocks stored in HDFS
3  Set IS size
4  Launching map tasks stage
5  Input: $V\_Node$ parameters (virtual CPU, virtual disk, and virtual network),
     system-level parameters (Disk)
6  Output: $V\_Node$ performance in terms of map tasks
7  while true do
8      Calculate the CPU performance $V\_Node_{ij}^{CPU}$
9      Calculate the disk IO rate $V\_Node_{ij}^{DiskIO}$
10     Calculate the map task influence $V\_Node_{ij}^{map}$
11     Calculate the map task performance $V\_Node_{ij}^{map\_perf}$
12     Find the rank list of map nodes by sorting $V\_Node_{ij}^{map\_perf}$ in descending order
         to find $map\_rank$
13 end
14 Find a set of blocks for each IS using ACO
15 Get information form scheduler
16 Control parameter initialization
17 $\tau_{x,z}$ – Pheromone matrix initialization
18 while (true) do
19     RWS calculation ($\rho_{ij}$)
20     Path construction ($Path_{x,z}$)
21     Ants generation
22     Mapping ants with path
23     Evaluate objective function for the candidate solution constructed
24     Calculate the NNLE and ABC for each block including replicated blocks
25         $REP_b(B_j^b(NL, RL, CL) \in IS_x, IS_x(NL, RL, CL))$
26         Min (NNLE) and Min (ABC)
27     Local pheromone update
28     Global pheromone update
29 end
30 Get a set of data blocks for each IS that minimizes the NNLE and bandwidth
     consumed.
31 Get the list of target $V\_Nodes$ based on the performance.
32 Schedule tasks.
33 Record the schedule into log for future use.
```

which attract more data blocks and the amount of data consumed across a virtual cluster. To represent the network bandwidth relationship among VMs, consider a graph (G) with a set of vertices (V) and a set of edges (E). In a connected graph G, V denotes VMs, and E denotes the bandwidth connection between VMs. Bandwidth consumed between the vertices and reading time of a data block from the busy VM denote the cost of a block in IS. Bandwidth consumption data structure ($BC^{d}_{(j,k)bw}$) is used to record the number of data blocks (d) transferred and the amount of bandwidth (bw) consumed (in MBs) between VMs (j and k). For instance, $BC^{4}_{(2,3)300}$ indicates the bandwidth consumption between VMs 2 and 3. There are four data blocks transferred and 300 MB of bandwidth consumed between VMs 2 and 3. Using this data structure, an analysis on the ABC can be performed using Eq. 4.7. The number of data blocks transferred between each VM can also be found. Besides, the number of replications for each block plays a significant role in minimizing non-local executions. $REP_b(B^b_j(NL, RL, CL) \in IS_x, IS_x(NL, RL, CL))$ denotes the information of three replications of block b. The downside is, if the replication factor is high, finding the right copy of the respective blocks is not easy and takes time to find the solution. Thus, it is essential to decide the right replication factor for data blocks. In general, default replication factor (three) is considered.

$$NNLE = \sum_{x=1}^{y}(RL_x + CL_x) \tag{4.6}$$

$$ABC = \sum_{j=1}^{n}\sum_{k=1}^{n} BC^{d}_{(j,k)bw} \tag{4.7}$$

4.4.3 Finding a set of data blocks for each IS using ACO

In the previous section, the number of data local executions and the ABC are modelled when a MapReduce job is periodically executed. With log information, we can infer the blocks that are frequently executed non-local in different VMs. To decide dynamically, scheduling information from the MapReduce job scheduler is obtained. With this information, we can find a set of data blocks to transfer over a virtual network, which can minimize NNLE and ABC. It is ensured that the resulting data block set for each IS constitutes minimal NNLE and overall bandwidth consumption. To achieve this, ACO algorithm is employed for finding a set of data blocks for each IS that can optimize the NNLE and overall bandwidth consumed. The reason for using ACO is to deal with discrete solution space.

Proposed model is mapped with the ACO algorithm, such that parameters in ACO will determine an optimal result. IDLACO ultimately finds a set of data blocks that belong to different IS and minimizes NNLE and overall bandwidth consumed. The block selection problem is mapped with the optimization problem and give a short glimpse of the ACO algorithm and the parameters used in ACO. Ants foraging behavior can be mapped with optimization problem as a tree structure with more than one level to find an optimal solution. As shown in Figure 4.2, there are y IS for a MapReduce job.

Each IS consists of s data blocks. Each level is a decision variable (IS) in an objective function, and nodes (data block) in each level denote possible solutions from search space. Each block in IS contains information whether it is NL/RL/CL and the amount of data to transfer. Moreover, each block has r replications, which complicate further the selection of the right blocks to move. Thus, based on this

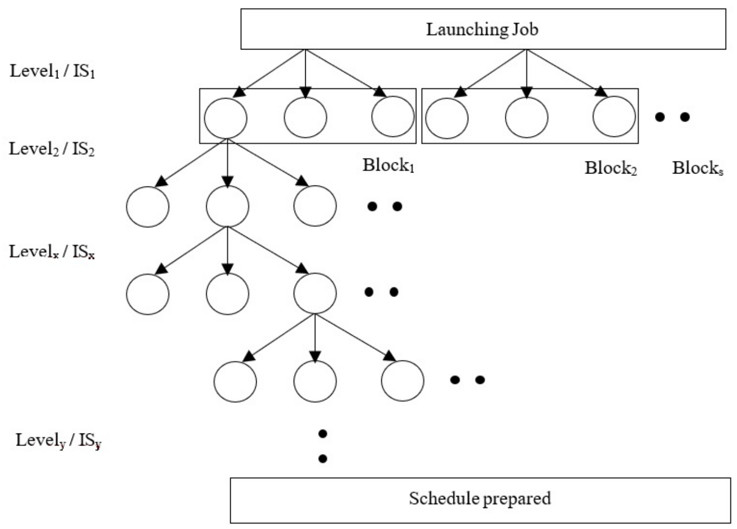

FIGURE 4.2: Finding a set of data blocks for IS using ACO

information, different data blocks are used to find a set of blocks to perform a non-local execution. For example, if the replication factor is 3 for each data block, s data blocks are chosen in each level. Once all decision variables obtain a set of data blocks, one can evaluate an objective function and update the solution iteratively until the optimal solution or specified iteration limit is reached. Algorithm 2 briefly describes the ACO algorithm for this problem.

To discuss the approach more elaborately the steps of the ACO algorithm are used to map this problem with ACO and include objective function. There are problem-specific parameters, such as IS and the number of data blocks in an IS. Besides, there are algorithm-specific parameters. Some of the problems and algorithm-specific parameters are mapped as presented in Table 4.2.

Table 4.2: Parameter mapping

Algorithm specific parameters	**Notation**	**Problem specific parameters**
Decision variable (level)	y	IS
Nodes in each level	s	Blocks in IS

Pheromone is initialized between every level for each edge in the tree. This value is not a problem specific parameter. Thus, a random value is assigned for the pheromone matrix in each level and each edge in the tree. Then, based on roulette wheel scheme (RWS) selection, a path from source to destination is selected. In the later iterations, this selection is affected by the amount of pheromone accumulated in the path. Eq. 4.8 is used to calculate the density of pheromone in each level, where $Path_{x,z}$ is a path matrix and $\tau_{x,z}$ is a pheromone matrix for each path in the tree.

$$p_{x,z} = \frac{\tau_{x,z}}{\sum_{z=1}^{s} \tau_{x,z}} \quad x = 1...y, z = 1...s \tag{4.8}$$

Using this pheromone matrix ($p_{x,z}$), a path is constructed for ants to get chosen in random. However, instead of creating a random path, we construct a path using

RWS based path, using Eq. 4.9. The path for each level is constructed using probability matrix. If there are four paths in a level, we construct a range for each path. After constructing a path between each level, several ants are randomly generated at each level. Here, an artificial ant is a random number generated between 0 and 1. The number of ants is equal to the number of candidate solutions. These ants are mapped with each path at the respective levels.

$$Path_{x,z} = \begin{bmatrix} 0 & p_{x,z} \\ p_{x,z} & p_{x,z} + p_{x,z+1} \\ p_{x,z} + p_{x,z+1} & p_{x,z} + p_{x,z+1} + p_{x,z+2} \\ . & . \\ . & . \\ . & p_{x,z} + p_{x,z+1} + \ldots + p_{x,s} \end{bmatrix} \tag{4.9}$$

Finally, one ant from each level is selected. If the number of ants is greater, the optimal solution may be reached quicker. However, it takes more computation time. Once a (node) block in each level is chosen, it is then combined to obtain the result. Each node contains two data structures which have already been explained: $IS_x(NL, RL, CL)$ and $B_j^b(NL, RL, CL) \in IS_x$. IS_x denotes NNLE initially decided by the task scheduler. If there are RL/CL executions mentioned in IS, B_j^b is explored to find the right block number (b) residing in three VMs (j) according to the replication factor. From these three VMs, the amount of bandwidth currently available and size of the data block are noted along with the total number of blocks to be processed in all target VMs. This is done in every level, resulting in combination of data blocks in each IS. These values are then passed to the objective functions Eqs. 4.6 and 4.7.

The combination that gives the optimal value for NNLE and overall bandwidth consumed is selected from the best path. Subsequently, pheromone from the best path is updated with the new probability value in the path matrix. Finally, the respective local and global paths are updated with the pheromone to increase the chance of the current best path to have chosen in the upcoming iterations. Once the algorithm produces a set of data blocks for each IS, they are scheduled for execution based on the performance of target VMs, as presented in the *Launching map tasks* stage part of Algorithm 2. Information on those data blocks was noted for repeated execution of the same job. Over time, if those data blocks are frequent to the same target VMs, they are cached in the target VM until the storage gets exhausted. If this information is not useful later times, it is discarded from the scheduler, which will remove the cached blocks from the target VMs. Figure 4.3 depicts a flowchart for steps of IDLACO.

4.5 Performance Evaluation

4.5.1 Experimental Setup

MapReduce task scheduler is simulated (Hadoop 2.7.0) to evaluate the proposed methodology on Ubuntu server with 12-core CPU (hyper-threaded), 64 GB memory, storage 4 x 1 TB HDD, and disk bandwidth rate 100 MB maximum. IDLACO is compared with classical fair scheduler and Holistic scheduler [106] based on parameters, such as the NNLE, average map task latency, job latency, and the ABC (within and between racks). These parameters are analyzed for various configurations of the number of blocks and IS. The configuration of the simulator with a 1 TB input file and 128 MB of HDFS block size is assumed . Word count job is used to compare the performance of IDLACO with other schedulers for those parameters. Besides, we assumed a CDC with physical servers and VMs to be highly heterogeneous. The work environment is static and tasks are independent. Here ten racks are considered each with ten physical machines that belong to different types, as given in Table 4.3, to launch VMs of different flavours, as given in Table 4.4. Altogether, we consider 100 physical machines of different types and, 100 VMs (20 VMs in each flavour) are launched across racks in the cluster. Each VM

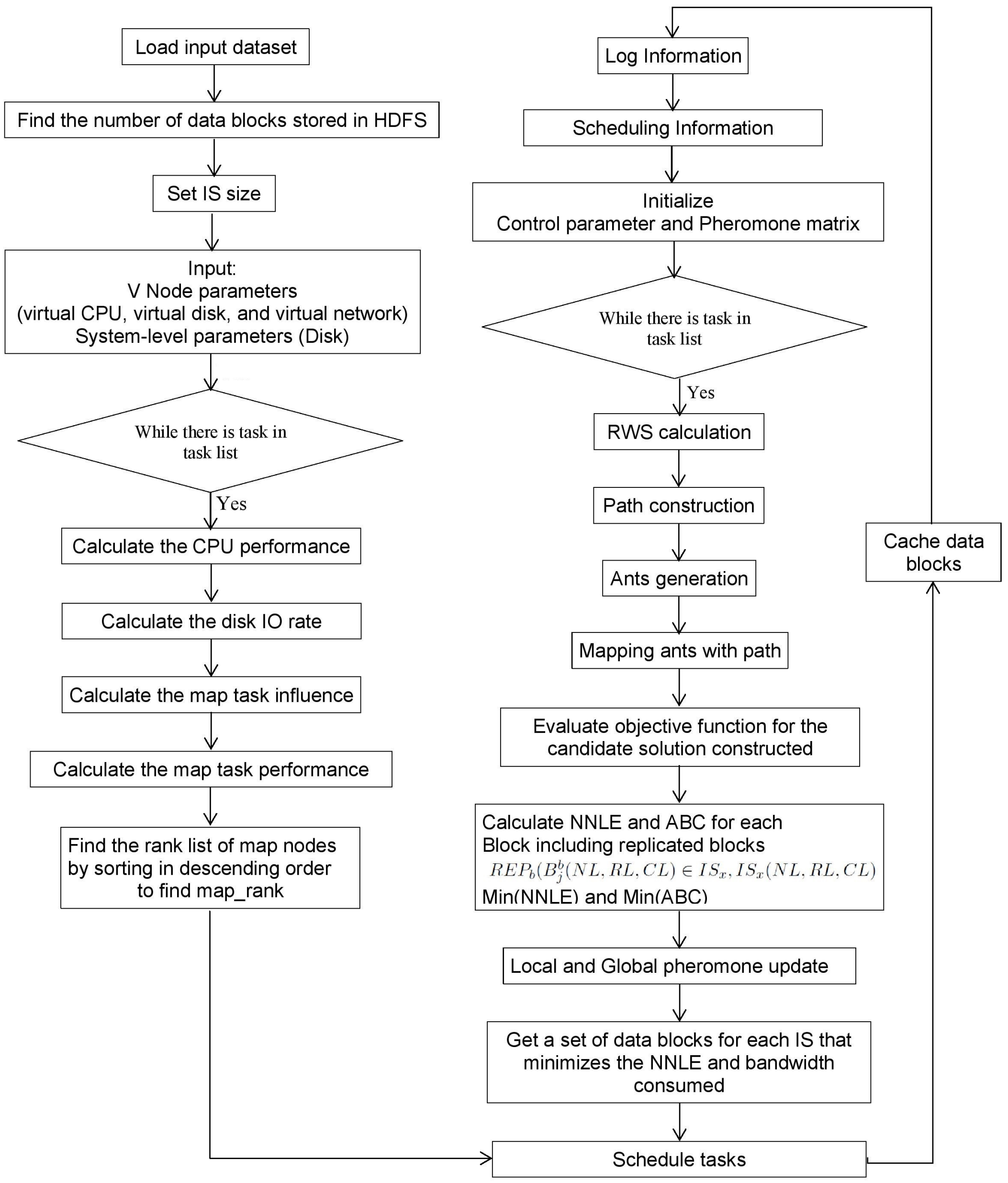
Load input dataset
Find the number of data blocks stored in HDFS
Set IS size
Input:
V Node parameters
(virtual CPU, virtual disk, and virtual network)
System-level parameters (Disk)
While there is task in task list
Yes
Calculate the CPU performance
Calculate the disk IO rate
Calculate the map task influence
Calculate the map task performance
Find the rank list of map nodes by sorting in descending order to find map_rank
Log Information
Scheduling Information
Initialize
Control parameter and Pheromone matrix
While there is task in task list
Yes
RWS calculation
Path construction
Ants generation
Mapping ants with path
Evaluate objective function for the candidate solution constructed
Calculate NNLE and ABC for each Block including replicated blocks
$REP_b(B_j^b(NL, RL, CL) \in IS_x, IS_x(NL, RL, CL)$
Min(NNLE) and Min(ABC)
Local and Global pheromone update
Get a set of data blocks for each IS that minimizes the NNLE and bandwidth consumed
Schedule tasks
Cache data blocks

FIGURE 4.3: Flowchart for IDLACO

is a Hadoop node. In total, 98 slave nodes, one resource manager, and one name node are configured. The schedulers are experimented for different combinations of replication factor (RF) and the number of blocks (s) in an IS, as given in Table 4.5.

Table 4.3: Physical machines configuration

PMs Types	**Configuration of physical machines**
PM_Type_1	Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz 6 cores, 32 GB memory, 1 TB HDD
PM_Type_2	Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz 28 cores, 132 GB memory, 3 TB HDD
PM_Type_3	Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz 4 cores, 8 GB memory, 1 TB HDD
PM_Type_4	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 4 cores, 8 GB memory, 1 TB HDD

Table 4.4: VM Types

VM Type	**vCPU**	**Memory (GB)**
Very small	1	2
Small	2	4
Medium	4	4
Large	8	16
Extra large	12	24

Table 4.5: RF and s combinations

RF	3	3	3	3	3	6	6
s	1	2	3	4	8	4	8

4.5.2 Results and Analysis

In Hadoop 2.x version, resource manager, a component of YARN (cluster resource manager), looks after the scheduling of jobs from different distributed processing tools, such as MapReduce, and Spark. Once a MapReduce job is scheduled by resource manager, application master (MRAppMaster), a component of MapReduce 2.x, schedules the map and reduce tasks. Each MapReduce job gets an MRAppMaster to manage their map and reduce tasks running independently. MRAppMaster collects log information and data block locations from the namenode, a component in HDFS, for scheduling map tasks across virtual cluster. Log information is used to understand the repeated pattern of data blocks copied to different VMs. As shown in Figure 4.1, MapReduce job log is maintained to identify a set of data blocks frequently copied to different VMs. Initially, MRAppMaster schedules map tasks by following locality principle. Then, using the proposed methodology, the NNLE and overall ABC across clusters are significantly minimized.

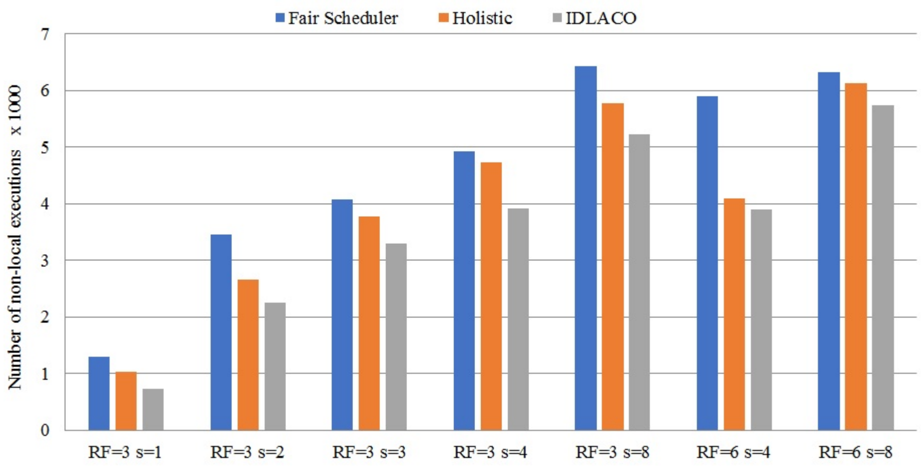

Figure 4.4: NNLE during execution

Table 4.6: Comparison of NNLE during execution

Configuration parameters	**Number of non-local executions**			**Improvement in IDLACO (%age) over**	
	Fair Scheduler	**Holistic Scheduler**	**IDLACO**	**Fair Scheduler**	**Holistic Scheduler**
RF=3 s=1	1290	1023	732	43.26	28.45
RF=3 s=2	3456	2659	2245	35.04	15.57
RF=3 s=3	4082	3769	3290	19.40	12.71
RF=3 s=4	4929	4721	3921	20.45	16.95
RF=3 s=8	6429	5782	5230	18.65	9.55
RF=6 s=4	5902	4100	3890	34.09	5.12
RF=6 s=8	6320	6129	5743	9.13	6.30
Average NNLE				**25.72**	**13.52**

Figure 4.4 shows the NNLE performed with fair scheduler, Holistic scheduler, and IDLACO. It is observed that IDLACO considerably has shown improvement up to 25.2% and 13.5% on average over fair scheduler and Holistic scheduler (as shown in Table 4.6). The objective of IDLACO is to minimize the NNLE; thereby, minimizing ABC across racks in the virtual cluster. Initially ACO is used to find the set of data blocks to copy on the virtual network. When s is increased for different RF, the NNLE increases. This is because a map task is executed in the node, where the first data block of IS is stored. Thus, other blocks in the

IS must be copied to the node where the map task is processing the first block. Fair scheduler copies the data blocks by default and causes to increase in NNLE. However, IDLACO minimized it when s is increased in each IS. For instance, consider RF=3, s=4 and RF=6, s=8. IDLACO improved, on average, up to 20.4% and 17% compared to fair scheduler and Holistic scheduler for RF = 3, s = 4. Similarly, 9.1% and 6.3% improvement is observed using IDLACO compared fair scheduler and Holistic scheduler for RF = 6, s = 8. ACO is time consuming, typically for NPH problems, when input search space is countably infinite. In the proposed algorithm, the search space for ACO is the number of IS (map task) for the next schedule. So, ACO does not take much time (not more than 3 seconds) to arrive optimal solution for the next schedule.

Based on the observation, when s is increased, NNLE significantly increases with fair scheduler and Holistic scheduler. In contrast, IDLACO showed its considerable performance improvement, as it initially finds a set of data blocks for each IS using ACO. When RF is increased, the number of combination of data blocks in each level increased, but it helps to identify the data blocks that need not be moved or that consumes less bandwidth for a map task. Thus, IDLACO finds a set of data blocks for each IS that could minimize the NNLE for a MapReduce job. This minimized map task latency up to 20.6% and 15.8% in average, as shown in Figure 4.5, when compared to fair scheduler and Holistic scheduler for different cases (as shown in Table 4.7). When s is increased in an IS, it minimized map task latency. However, when RF is doubled for s in an IS,there is a little improvement for map task latency in average. The reason is, even though RF increased that placed a copy of data blocks across the cluster, map task is executed where the first data block resides. Rest of the data blocks are yet to be copied over virtual network. So, the number of data blocks are high to copy across the cluster. Therefore, it is important to note that doubling the RF and s in an IS does not result in doubling the performance.

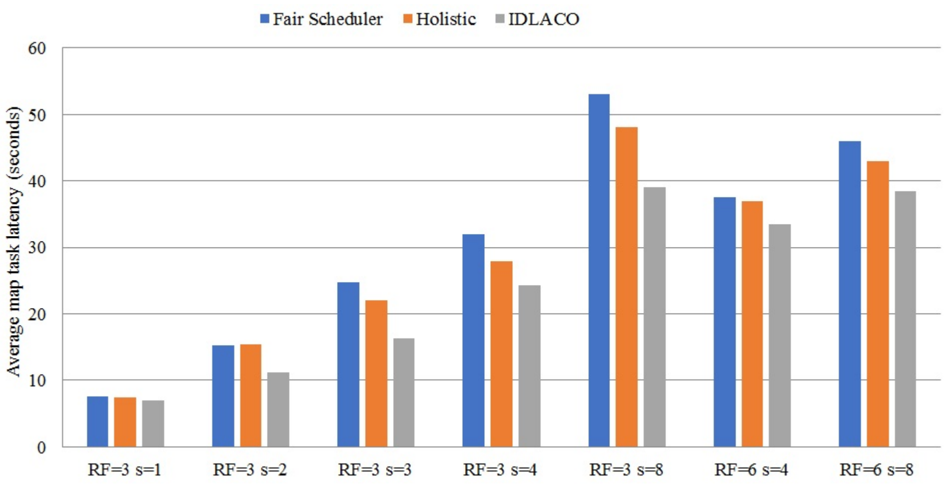

Figure 4.5: Average map task latency

Table 4.7: Comparison of average map task latency

Configuration parameters	**Map task latency (seconds)**			**Improvement in IDLACO (%age) over**	
	Fair Scheduler	**Holistic Scheduler**	**IDLACO**	**Fair Scheduler**	**Holistic Scheduler**
RF=3 s=1	7.5	7.4	7.0	6.67	5.41
RF=3 s=2	15.2	15.4	11.2	26.32	27.27
RF=3 s=3	24.7	22.1	16.3	34.01	26.24
RF=3 s=4	31.9	27.9	24.3	23.82	12.90
RF=3 s=8	53.0	48.0	39.0	26.42	18.75
RF=6 s=4	37.5	37.0	33.4	10.93	9.73
RF=6 s=8	46.0	43.0	38.5	16.30	10.47
Average map task latency				**20.64**	**15.82**

MapReduce job latency is minimized further by scheduling map tasks based on the dynamic performance of VMs. Figure 4.6 shows the job latency of different schedulers for different combinations of RF and s..

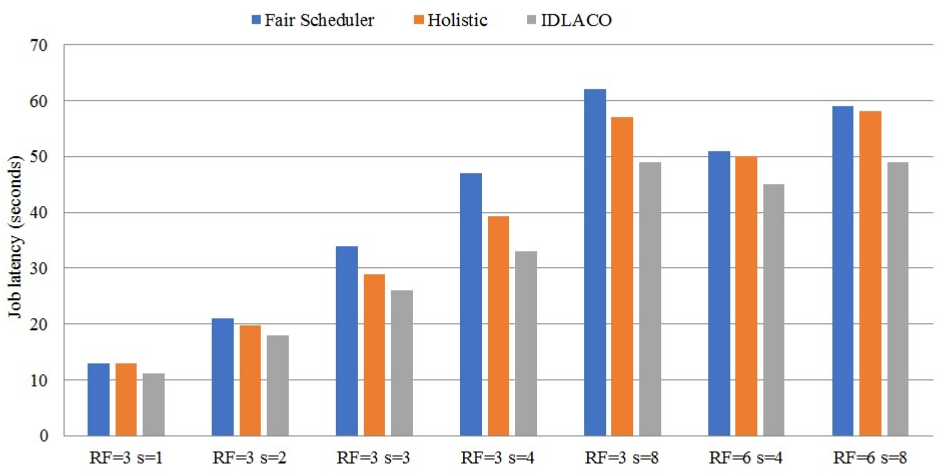

FIGURE 4.6: Job latency using different schedulers

TABLE 4.8: Comparison of Job latency using different schedulers

Configuration parameters	**Job latency**			**Improvement in IDLACO (%age) over**	
	Fair Scheduler	**Holistic Scheduler**	**IDLACO**	**Fair Scheduler**	**Holistic Scheduler**
RF=3 s=1	13	12.9	11.1	14.62	13.95
RF=3 s=2	21	19.8	18	14.29	9.09
RF=3 s=3	34	29	26	23.53	10.34
RF=3 s=4	47	39.3	33	29.79	16.03
RF=3 s=8	62	57	49	20.97	14.04
RF=6 s=4	51	50	45	11.76	10.00
RF=6 s=8	59	58.2	49	16.95	15.81
Average job latency improvement				**18.84**	**12.75**

High degree of heterogeneous configuration of physical machines and different flavours of VMs cause heterogeneity in performance. So, even though scheduling a map task in a node, there may be high-performing VMs to process all data blocks in the IS in a short time. Therefore, soon after finding a set of data blocks for an IS that constitutes small NNLE, all VMs that contain data blocks from IS are examined whether the current VM performance is good enough to finish the task quickly. So, IDLACO schedules the map tasks to the VM that delivers high performance in processing data blocks. In effect, it improved job latency up to 18.8% and 12.7% on average for all different configurations compared fair scheduler and Holistic scheduler as shown in Table 4.8.

More specifically, when s and RF increased, job latency was minimized. When RF is high, there is a greater number of blocks residing across the virtual cluster hosted in a CDC. Thus, MRAppMaster gets more opportunity to examine the performance of different VMs and assign map tasks accordingly. This ultimately caused reduction in the job latency. However,the number of combinations to check in each IS by ACO will be high when RF and s are increased. For instance, when RF and s are doubled like in RF=3, s=2 and RF=6, s=4, job latency is minimized up to 11.7% and 10% when compared to fair scheduler and Holistic scheduler. Thus, it is essential to understand the heterogeneous performance in heterogeneous environment to schedule map tasks. Another important claim of IDLACO is to minimize the ABC during the map task execution. We assumed no other MapReduce job stands in the shuffe phase. As NNLE minimized, with no surprise, the ABC during the map task execution is also minimized, as shown in Figure 4.7, up to 25.7% and 15.2% for all different combinations of RF and s compared to the fair scheduler and Holistic scheduler (as shown in Table 4.9).

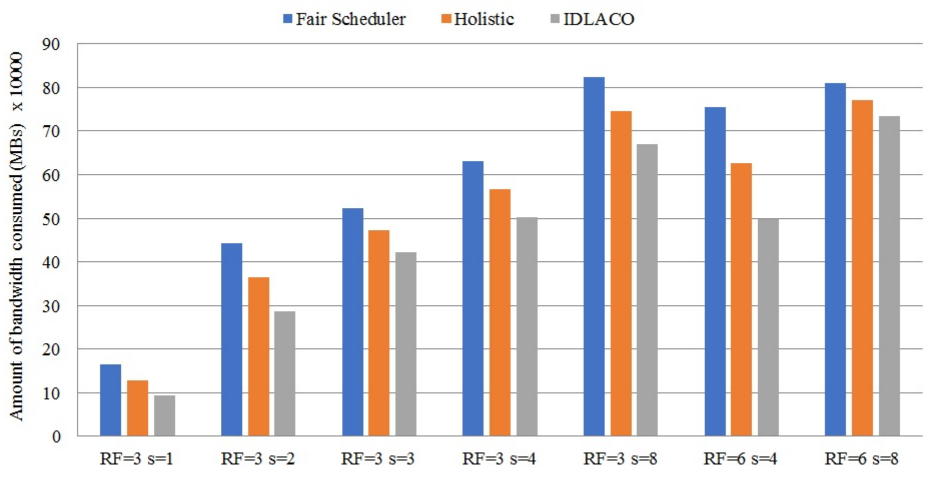

FIGURE 4.7: Overall ABC during non-local executions

Table 4.9: Comparison of overall ABC during non-local executions

Configuration parameters	**Amount of bandwidth consumed (MB)**			**Improvement in IDLACO (%age) over**	
	Fair Scheduler	**Holistic Scheduler**	**IDLACO**	**Fair Scheduler**	**Holistic Scheduler**
RF=3 s=1	165120	129408	93696	43.26	27.60
RF=3 s=2	442368	364864	287360	35.04	21.24
RF=3 s=3	522496	471808	421120	19.40	10.74
RF=3 s=4	630912	566400	501888	20.45	11.39
RF=3 s=8	822912	746176	669440	18.65	10.28
RF=6 s=4	755456	626688	497920	34.09	20.55
RF=6 s=8	808960	772032	735104	9.13	4.78
Average ABC				**25.72**	**15.23**

This significant performance gain is due to caching frequent set of data blocks copied over the virtual network. Once the data blocks are cached in the target VM, they are used for future map task execution, as MapReduce jobs in a production environment are periodically executed. If the pattern of data blocks executed is not the same, the cached data blocks are removed from the target VM.

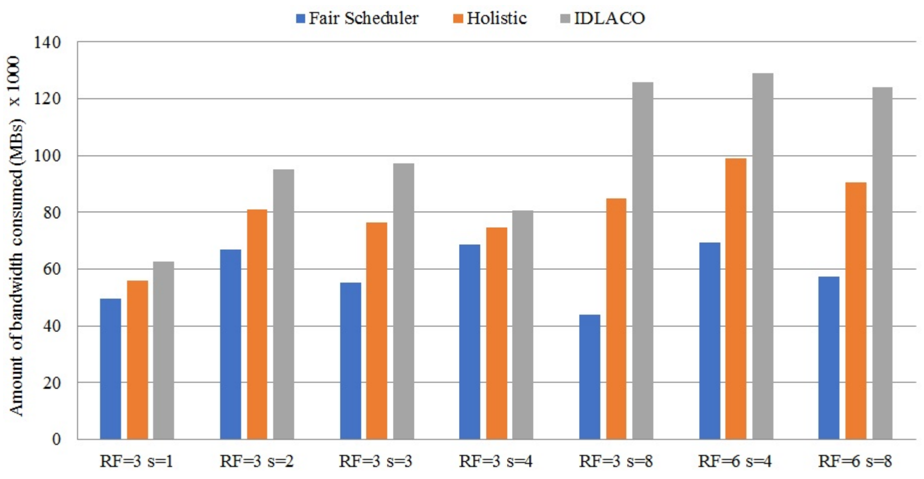

Figure 4.8: ABC within rack during job execution

Even though IDLACO performed better than fair scheduler and Holistic scheduler in minimizing the ABC, it is essential to analyse the bandwidth consumption within and between racks. Figure 4.8 shows the difference between the schedulers based on ABC within racks. Since there is no rack awareness in the Hadoop virtual cluster, it is not easy to achieve the RL execution. Here, IDLACO always

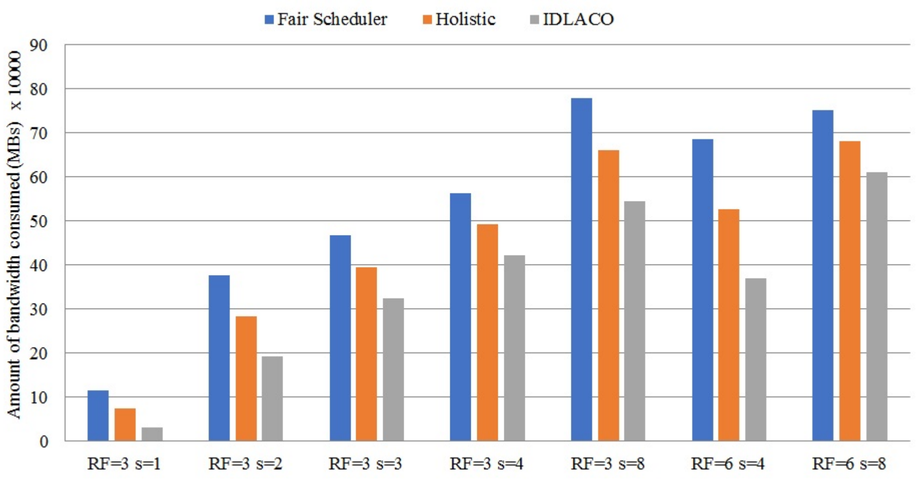

FIGURE 4.9: ABC across racks during job execution

prefers the VM located in the same rack for bringing data blocks mentioned in IS. Thus, moving data blocks between racks is mostly avoided. IDLACO aims to copy the data blocks required for different IS within the rack and avoids transferring them over a network. Thus, the proposed method minimized the number of non-local executions across the cluster. Due to this, IDLACO observed RL bandwidth consumption up to 78.7% and 25.6%, in average, to keep the relevant data blocks within the rack compared to the fair scheduler and Holistic scheduler.

TABLE 4.10: Comparison of ABC within-rack and across-racks during job execution

Configuration parameters	**Rack Local**			**Across Racks**			**Across Racks Improvement in IDLACO (%age) over**	
	Fair Scheduler	**Holistic Scheduler**	**IDLACO**	**Fair Scheduler**	**Holistic Scheduler**	**IDLACO**	**Fair Scheduler**	**Holistic Scheduler**
RF=3 s=1	49469	56066.5	62664	115651	73341.5	31032	73.17	57.69
RF=3 s=2	66879	81003.5	95128	375489	283860.5	192232	48.80	32.28
RF=3 s=3	55257	76234.5	97212	467239	395573.5	323908	30.68	18.12
RF=3 s=4	68734	74764.5	80795	562178	491635.5	421093	25.10	14.35
RF=3 s=8	43836	84779.5	125723	779076	661396.5	543717	30.21	17.79
RF=6 s=4	69168	98984	128800	686288	527704	369120	46.22	30.05
RF=6 s=8	57452	90667.5	123883	751508	681364.5	611221	18.67	10.29
Average improvement in across-rack bandwidth consumption							**38.98**	**25.80**

Therefore, bandwidth consumption between racks, as shown in Figure 4.9, is minimized up to 38.9% and 25.8% compared with fair scheduler and Holistic scheduler. When s is high RF=3,s=8, and RF=6,s=8 for an IS, IDLACO minimized the

ABC up to 30.2% and 18.6% over fair scheduler and 17.7% and 10.2% over Holistic scheduler. When s is a small number with default replication, it has little chance of getting data blocks from the VM residing in the rack. This is because there are an insuffcient number of copies of the same data block, especially in the same rack. Thus, it is essential to consider RF high when the number of data blocks in an IS is high. This is the case when RF=3,s=1, for which 73.1% and 57.6% of overall bandwidth consumption is recorded between racks for fair scheduler and Holistic scheduler. Thus, using the right combination of RF and s in an IS plays a significant role in improving the job latency by minimizing the NNLE and ABC.

4.6 Summary

Hadoop MapReduce is widely used as a service by different sectors. Improving the performance of the MapReduce is a primary objective, especially, in a heterogeneous virtualized cloud environment. When an IS is assigned with a greater number of data blocks, NNLE increases, leading to high bandwidth consumption and job latency. To overcome this, IDLACO is proposed to find a set of data blocks for each map task of a job to minimize NNLE and ABC. Then, the target virtual machine is determined based on its heterogeneous performance to perform non-local execution. Finally, if a set of data blocks are copied over a network repeatedly, it is decided to temporarily cache those data blocks in the target VM. IDLACO outperformed the fair scheduler for about 25.7%, 20.6%, 18.8%, 25.7%, and for the Holistic scheduler for about 13.5%, 15.8%, 12.7%, 15.2%, on average for parameters NNLE, average map task latency, job latency, and the ABC, respectively, for a MapReduce job.

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

Big data refers to the complex and huge data sets and big data mining is a process of discovering unknown patterns from big data.With the rising and quickly growing data, things are varying in the business environment. Big data is fetching the hottest ultimate edge for data research and for many business applications. Companies are currently using big data analysis to forecast the upcoming trends so that enormous value can be produced out of it. Because of tremendous measure of information around us, the current database tools face issues identified with colossal measure of information, speed of data, data sharing, scalability, efficiency, privacy and security of data. This thesis demonstrates a review of various scheduling algorithms used in MapReduce phase of Hadoop. The default schedulers of Hadoop: FIFO, Capacity Scheduler and Fair Scheduler accept the cluster environment to be homogeneous and work viably in homogeneous condition. In case the cluster is heterogeneous, the execution of Hadoop is significantly cut down. Various scheduling algorithms have been envisioned but they do bargain on some attributes or metrics to improve one or more attribute. This thesis, also device a review on data locality in MapReduce to find out few factors that troubles data locality and harms overall performance. A couple of issues that inconveniences data locality are mechanism for distribution of data, cluster and network load, complex load, cost, resource sharing, cluster environment (homogeneous or heterogeneous), unplanned clients requests, size of data blocks, number of mappers and reducers.

Without cloud service, it is impossible to use big data processing frameworks as it is expensive to set up on-premise. Hadoop is one of the efficient processing tools for crunching a large volume of data. It is also offered as cloud service on a cluster of virtual machines hosted in a cluster of physical servers in a cloud data-center. In this case, heterogeneity in physical servers and virtual machine performance is unavoidable. Therefore, we mainly focus on disk IO performance due to many virtual machines hosted in a same hard disk drive. It highly affects map task execution in map phase due to delay in bringing data blocks from disk into memory. This motivated us to investigate the disk performance when virtual machines are hosted in same disk. To distribute data blocks based on the performance of disk IO, simple linear regression algorithm is used to predict disk IO performance and distribute data blocks accordingly. Therefore, data blocks for map tasks are brought into memory quickly. Moreover, varying performance of virtual machine due to co-located virtual machine's interference affects MapReduce job latency. Therefore, a methodology "PBDBP" is proposed to dynamically exploit heterogeneous performance, and if any high variation in performance for long time, then data blocks are redistributed in such a way that, data local execution is improved. At any time, the number of data blocks in a virtual machine are stored based on the performance of that virtual machine. Finally the ideas are simulated based on Hadoop 2.7.0 and compared with classical fair scheduler, RWS-based scheduler, and HMJS. Results claim that the proposed scheduler outperformed up to 66%, 52%, and 19% compared to those existing schedulers for makespan.

Hadoop MapReduce is widely used as a service by different cloud service providers. Thus, improving the performance of the MapReduce scheduler is a primary objective, especially in a heterogeneous virtualized cloud environment. When an IS is assigned with a greater number of data blocks, NNLE increases, leading to high bandwidth consumption and job latency. To overcome this situation, a methodology “IDLACO” is proposed to find a list of data blocks for each map task of a job for performing non-local execution, minimizing the job latency and virtual network consumption. Then, the target virtual machine is determined on the basis of its heterogeneous performance to copy these data blocks. Finally, if a specific set of data blocks are transferred over a network for repeated job execution, it is decided to temporarily cache those data blocks in the target VM. IDLACO outperformed the fair scheduler for about 25.7%, 20.6%, 18.8%, 25.7%, and for the Holistic scheduler for about 13.5%, 15.8%, 12.7%, 15.2%, on average for parameters NNLE, average map task latency, job latency, and the ABC, respectively, for a MapReduce job.

5.2 Future Scope

There is always chance of improvements. Nobody can claim that no further improvements beyond this point. So the parameters job latency, makespan, number of non-local executions, average map task latency, and the amount of bandwidth consumed to improve data locality and amount of data processed in heterogeneous virtualized environment can further be improved. One of the important research question here is that whether we can extend this work for reduce phase also?

As we have considered parameters job latency, makespan, average map task latency, NNLE and ABC to enhance the performance. Therefore, in future the work can be extended by considering few other parameters like flow-time, resource utilization, utilization of machines and designing more specific cost evaluation model for reduce phase also. We have floated an idea by considering different types of heterogeneities in physical and virtual machines. The heterogeneity and dynamism for both physical and virtual machines can also be increased in future to test schedulers with increased cluster size along with more workloads of varying sizes to compare and contrast the performance on virtual as well as real environment.

REFERENCES

[1] M. Cox and D. Ellsworth, “Managing big data for scientific visualization,” *ACM Siggraph*, vol. 97, no. May, pp. 5.1–5.17, 1997.

[2] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, “Big data : The next frontier for innovation, competition, and productivity.” McKinsey Global Institute, Tech. Rep. June, 2011.

[3] P. C. Zikopoulos, D. DeRoos, K. Parasuraman, T. Deutsch, D. Corrigan, and J. Giles, *Harness the Power of Big Data.* The McGraw-Hill Companies, 2013.

[4] J. J. Berman, *Principles of Big Data : preparing, sharing, and analyzing complex information.* Morgan Kaufmann Elsevier, 2013.

[5] J. Gantz and D. Reinsel, “Extracting Value from Chaos,” Tech. Rep. june, 2011.

[6] M. Chen, S. Mao, and Y. Liu, “Big Data: A Survey,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, apr 2014. [Online]. Available: http://link.springer.com/10.1007/s11036-013-0489-0

[7] D. Reinsel, J. Gantz, and J. Rydning, “The Digitization of the World- From Edge to Core,” 2018.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable : A Distributed

Storage System for Structured Data," *ACM Transactions on Computer Systems(TOCS)*, vol. 26, no. 2, pp. 1–14, 2008.

[9] D. Agrawal, S. Das, and A. E. Abbadi, "Big Data and Cloud Computing : Current State and Future Opportunities," in *EDBT/ICDT '11 Proceedings of the 14th International Conference on Extending Database Technology.* ACM, 2011, pp. 530–533.

[10] A. E. Abbadi, D. Agrawal, and S. Das, "Big Data and Cloud Computing : New Wine or just New Bottles?" in *36th International Conference on Very Large Data Bases, September 13-17, 2010, Singapore.* VLDB Endowment, 2010, pp. 1647–1648.

[11] T. White, *Hadoop: The Definitive Guide*, 4th ed. O'Reilly Media.

[12] X. Zhao, L. Liu, Q. Zhang, and X. Dong, "Improving MapReduce Performance in a Heterogeneous Cloud : A Measurement Study," in *2014 IEEE 7th International Conference on Cloud Computing*, no. June 2014. IEEE, 2016, p. 8.

[13] V. Kalavri and V. Vlassov, "MapReduce: Limitations, optimizations and open issues," in *Proceedings - 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013.* IEEE, 2013, pp. 1031–1038.

[14] A. Sharma and G. Singh, "A Review on Data locality in Hadoop MapReduce," in *5th IEEE International Conference on Parallel, Distributed and Grid Computing(PDGC-2018).* IEEE, 2018, pp. 723–728.

[15] S. Kurazumi, T. Tsumura, S. Saito, and H. Matsuo, "Dynamic processing slots scheduling for I / O intensive jobs of Hadoop MapReduce," in *2012 Third International Conference on Networking and Computing.* IEEE, 2012, pp. 288–292.

[16] B. Saraladevi, N. Pazhaniraja, P. V. Paul, M. S. S. Basha, and P. Dhavachelvan, “Big Data and Hadoop-A Study in Security Perspective,” *Procedia Computer Science*, vol. 50, no. 2015, pp. 596–601, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.04.091

[17] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, “Big Data Processing in Cloud Computing Environments,” in *2012 International Symposium on Pervasive Systems, Algorithms and Networks*, no. December. IEEE, 2012, pp. 17–23.

[18] Y.-S. Song, “Storing Big Data — The Rise of the Storage Cloud,” 2012.

[19] M. R. Ghazi and D. Gangodkar, “Hadoop , MapReduce and HDFS : A Developers Perspective,” *Procedia Computer Science*, vol. 48, pp. 45–50, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.04.108

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop distributed file system,” in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010*, 2010, pp. 1–10.

[21] V. Martha, “Big Data Processing Algorithms,” in *Studies in Big Data*, volume 11 ed., H. Mohanty, P. Bhuyan, and D. Chenthati, Eds. Springer, 2015, pp. 61–92.

[22] M. Vaidya, “Parallel Processing of cluster by Map Reduce,” *International Journal of Distributed and Parallel Systems*, vol. 3, no. 1, pp. 167–179, 2012.

[23] E. D. Raj and L. D. Dhinesh Babu, “A two pass scheduling policy based resource allocation for mapreduce,” in *Procedia Computer Science, International Conference on Information and Communication Technologies (ICICT 2014)*, vol. 46. Elsevier B.V., 2015, pp. 627–634. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.02.110

[24] B. He, Q. Luo, Q. Luo, N. K. Govindaraju, and T. Wang, “Mars : A MapReduce Framework on Graphics Processors,” in *PACT ’08 Proceedings of the*

17th international conference on Parallel architectures and compilation techniques. ACM, 2008, pp. 260–269.

[25] J. Dean and S. Ghemawat, “MapReduce : Simplified Data Processing on Large Clusters,” *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51, no. 1, pp. 1–13, 2008.

[26] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, “SAMR : A Self-adaptive MapReduce Scheduling Algorithm In Heterogeneous Environment,” in *2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010).* IEEE, 2010, pp. 2736–2743.

[27] M. Khan, Y. Liu, and M. Li, “Data Locality in Hadoop Cluster Systems,” in *11th International Conference on Fuzzy Systems and Knowledge Discovery Data.* IEEE, 2014, pp. 720–724.

[28] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, “Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters,” in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW).* IEEE, 2010, p. 9.

[29] Z. Guo, G. Fox, M. Zhou, and Y. Ruan, “Improving Resource Utilization in MapReduce,” in *IEEE International Conference on Cluster Computing.* IEEE, 2012, pp. 402–410.

[30] J. Geetha, N. Udaybhaskar, and P. Chennareddy, “Data-local Reduce Task Scheduling,” in *Procedia Computer Science*, vol. 85, no. CMS 2016. Elsevier Masson SAS, 2016, pp. 598–605. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2016.05.226

[31] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "MapTask Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 190–203, 2016.

[32] S. Dong-Hee, "Demystifying big data : Anatomy of big data developmental process," *Telecommunications Policy*, vol. 40, no. 9, pp. 837–854, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.telpol.2015.03.007

[33] A. Paul and R. Jeyaraj, "Internet of Things : A primer," *Human Behaviour and Emerging Technologies*, vol. 1, no. 1, pp. 37–47, 2018.

[34] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Sixth Symp. Oper. Syst. Des. Implement.*, vol. 51, no. 1, pp. 107–113, 2004. [Online]. Available: https://www.usenix.org/legacy/event/osdi04/tech/full{_}papers/dean/dean.pdf

[35] Y. Guo, J. Rao, and C. Jiang, "Moving Hadoop into the Cloud with Flexible Slot Management and Speculative Execution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 798–812, 2016.

[36] R. Boutaba, L. Cheng, and Q. Zhang, "On Cloud computational models and the heterogeneity challenge," *Journal of Internet Services and Appl ications*, vol. 3, pp. 77–86, 2012.

[37] A. Forrest. (2007) What is apache hadoop? [Online]. Available: http://hadoop.apache.org

[38] V. Pandey and P. Saini, "How Heterogeneity Affects the Design of Hadoop MapReduce Schedulers : A State-of-the-Art Survey and Challenges," *Big Data*, vol. 6, no. 2, pp. 72–95, 2018.

[39] R. Jeyaraj, V. Ananthanarayana, and A. Paul, "Dynamic ranking-based MapReduce job scheduler to exploit heterogeneous performance in a

virtualized environment," *The Journal of Supercomputing*, vol. 75, no. 0123456789, pp. 7520–7549, 2019. [Online]. Available: https://doi.org/10.1007/s11227-019-02960-0

[40] K. Jackson, *OpenStack Cloud Computing Cookbook.* Packt Publishing, 2012.

[41] (2020) Hadoop mapreduce fair scheduler. [Online]. Available: https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/FairScheduler.html

[42] (2020) Hadoop mapreduce capacity scheduler. [Online]. Available: https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[43] R. Jeyaraj, A. V. S, and A. Paul, "Improving MapReduce scheduler for heterogeneous workloads in a heterogeneous environment," *Concurrency and Computation Practice and Experience*, no. December, pp. 1–10, 2019.

[44] L. Chen and Z.-h. Liu, "Energy-and locality-efficient multi-job scheduling based on MapReduce for heterogeneous datacenter," *Service Oriented Computing and Applications*, vol. 13, no. 4, pp. 297–308, 2019. [Online]. Available: https://doi.org/10.1007/s11761-019-00273-x

[45] V. Marx, "TECHNOLOGY FEATURE: THE BIG CHALLENGES OF BIG DATA," *Nature*, vol. 498, no. J U N E 2013, pp. 255–260, 2013.

[46] H. S. Bhosale and D. P. Gadekar, "A Review Paper on Big Data and Hadoop," *International Journal of Scientific and Research Publications*, vol. 4, no. 10, pp. 1–7, 2014.

[47] S. T. F. Al-janabi and M. A.-s. Rasheed, "Public-Key Cryptography Enabled Kerberos Authentication," in *2011 Developments in E-systems Engineering Public-Key*, no. December. IEEE, 2011, pp. 209–214.

[48] Z. Fadika, E. Dede, J. Hartog, and M. Govindaraju, "MARLA : MapReduce for Heterogeneous Clusters," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.* ACM, 2012, pp. 49–56.

[49] Y. Mao and J. Ling, "Research on Load Balance Strategy Based on Grey Prediction Theory in Cloud Storage," in *2nd International Conference on Electronic & Mechanical Engineering and Information Technology (EMEIT-2012).* Atlantis Press, Paris, France, 2012, pp. 199–203.

[50] X. Ye, M. Huang, D. Zhu, and P. Xu, "A novel blocks placement strategy for hadoop," in *Proceedings - 2012 IEEE/ACIS 11th International Conference on Computer and Information Science.* IEEE, 2012, pp. 3–7.

[51] J. Ling and X. Jiang, "Distributed storage method based on information dispersal algorithm," in *Proceedings - 2013 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation, IMSNA 2013.* IEEE, 2013, pp. 624–626.

[52] S. D. Madhu Kumar and T. P. Shabeera, "Bandwidth-Aware Data Placement Scheme for Hadoop," in *2013 IEEE Recent Advances in Intelligent Computational Systems (RAICS).* IEEE, 2013, pp. 64–67.

[53] K. Fan, D. Zhang, H. Li, and Y. Yang, "An Adaptive Feedback Load Balancing Algorithm in HDFS," in *2013 5th International Conference on Intelligent Networking and Collaborative Systems.* IEEE, 2013, pp. 23–29. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6630284

[54] C. W. Lee, K. Y. Hsieh, S. Y. Hsieh, and H. C. Hsiao, "A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments," *Big Data Research*, vol. 1, no. October, pp. 14–22, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.bdr.2014.07.002

[55] Z. Gao, D. Liu, Y. Yang, J. Zheng, and Y. Hao, "A load balance algorithm based on nodes performance in Hadoop cluster," in *APNOMS 2014 - 16th Asia-Pacific Network Operations and Management Symposium*. IEEE, 2014, pp. 1–4.

[56] C. Y. Lin and Y. C. Lin, "A load-balancing algorithm for hadoop distributed file system," in *Proceedings - 2015 18th International Conference on Network-Based Information Systems*. IEEE, 2015, pp. 173–179.

[57] D. Kim, E. Choi, and J. Hong, "System Information-based Hadoop Load Balancing for Heterogeneous Clusters," in *RACS '15 International Conference on Research in Adaptive and Convergent Systems*. ACM, 2015, pp. 465–467.

[58] N. S. Islam, X. Lu, D. Shankar, and D. K. D. K. Panda, "Triple-H : A Hybrid Approach to Accelerate HDFS on HPC Clusters with Heterogeneous Storage Architecture," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing Triple-H:*. ACM, 2015, pp. 101–110.

[59] S. Wang and H. Zhou, "The Research of MapReduce Load Balancing Based on Multiple Partition Algorithm," in *IEEE/ACM 9th International Conference on Utility and Cloud Computing*. IEEE/ACM, 2016, pp. 339–342.

[60] X. Hou, D. Pal, A. Kumar T K, J. P. Thomas, and H. Liu, "Privacy Preserving Rack-based Dynamic Workload Balancing for Hadoop MapReduce," in *IEEE 2nd International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, IEEE International Conference on Intelligent Data and Security*. IEEE, 2016, pp. 30–35.

[61] J. J. V. Nayahi and V. Kavitha, "Privacy and utility preserving data clustering for data anonymization and distribution on Hadoop," *Future*

Generation Computer Systems, vol. 74, no. September 2017, pp. 393–408, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.future.2016.10.022

[62] Y. Song, Y.-s. Shin, M. Jang, and J.-w. Chang, "Design and Implementation of HDFS Data Encryption Scheme using ARIA Algorithm on Hadoop," in *4th International Conference on Big Data and Smart Computing (BigComp 2017).* IEEE, 2017, pp. 84–90.

[63] D. Tao, Z. Lin, and B. Wang, "Load Feedback-Based Resource Scheduling and Dynamic Migration-Based Data Locality for Virtual Hadoop Clusters in OpenStack-Based Clouds," *Tsinghua Science and Technology*, vol. 22, no. 2, April 2017, pp. 149–159, 2017.

[64] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *8th USENIX Symposium on Operating Systems Design and Implementation.* USENIX Association, 2008, pp. 29–42.

[65] K. Kc and K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines," in *2nd IEEE International Conference on Cloud Computing Technology and Science.* IEEE, 2010, pp. 388–392.

[66] X. Dai and B. Bensaou, "Scheduling for response time in Hadoop MapReduce," in *IEEE ICC 2016 SAC Cloud Communications and Networking.* IEEE, 2016, pp. 3627–3632.

[67] J. Polo, D. de Nadal, D. Carrera, Y. Becerra, V. Beltran, J. Torres, and E. Ayguadé, "Adaptive Task Scheduling for MultiJob MapReduce Environments," Tech. Rep., 2009. [Online]. Available: http://capinfo.e.ac.upc.edu/PDFs/dir04/file003484.pdf

[68] S. Seo, I. Jang, K. Woo, I. Kim, J.-s. Kim, and S. Maeng, "HPMR : Prefetching and Pre-shuffling in Shared MapReduce Computation Environment," in

IEEE International Conference on Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE, 2009, p. 8.

[69] T. Sandholm and K. Lai, "Dynamic proportional share scheduling in hadoop," in *LJSSPP'10 Proceedings of the 15th international conference on Job scheduling strategies for parallel processing Atlanta, GA.* ACM, 2010, pp. 110–131.

[70] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling : A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," in *EuroSys '10 Proceedings of the 5th European conference on Computer systems.* ACM, 2010, pp. 265–278.

[71] M. J. Fischer, X. Su, and Y. Yin, "Assigning Tasks for Efficiency in Hadoop," in *SPAA '10 Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures.* ACM, 2010, pp. 30–39.

[72] X. Zhang, Z. Zhong, S. Feng, B. Tu, and J. Fan, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments," in *Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications*, vol. 2. IEEE, 2011, pp. 1–7.

[73] C. L. Abad, Y. Lu, and R. H. Campbell, "DARE : Adaptive Data Replication for Efficient Cluster Scheduling," in *2011 IEEE International Conference on Cluster Computing DARE:.* IEEE, 2011, pp. 159–168.

[74] L. M. Khanli, A. Isazadeh, and T. N. Shishavan, "PHFS : A dynamic replication method , to decrease access latency in the multi-tier data grid," *Future Generation Computer Systems*, vol. 27, no. 3, pp. 233–244, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.08.013

[75] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality and fairness in MapReduce," in *Proceedings of third international workshop on*

MapReduce and its Applications Date, 2012, pp. 25–32. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2287016.2287022

[76] J. Lee, J. Lim, D. Jung, HeonchangYu, K. Chung, and JoonMin Gil, "Adaptive Data Replication Scheme Based on Access Count Prediction in Hadoop," in *WORLDCOMP'13 - The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing*, 2013, pp. 1–7.

[77] W. Wang and L. Ying, "Data Locality in MapReduce : A Network Perspective," in *Fifty-second Annual Allerton Conference Allerton House, UIUC, Illinois, USA*. IEEE, 2014, pp. 1110–1117.

[78] T. A. Phuong, "Reliable and Locality Driven Scheduling in Hadoop," Ph.D. dissertation, 2014.

[79] J. Bo, W. Jiaying, S. Xiuyu, and H. Ruhuan, "Hadoop Scheduling Base On Data Locality," 2015.

[80] C. H. Hsu, K. D. Slagter, and Y. C. Chung, "Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications," *Future Generation Computer Systems*, vol. 53, pp. 43–54, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.future.2015.04.006

[81] P. Zhang, C. Li, and Y. Zhao, "An improved task scheduling algorithm based on cache locality and data locality in Hadoop," in *17th International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 2016, pp. 244–249.

[82] I. A. T. Hashem, N. B. Anuar, M. Marjani, A. Gani, A. K. Sangaiah, and A. K. Sakariyah, "Multi-objective scheduling of MapReduce jobs in big data processing," *Multimedia Tools and Applications*, vol. 77, no. 8, pp. 9979–9994, 2017.

[83] S. M. Nabavinejad and M. Goudarzi, "Data Locality and VM Interference Aware Mitigation of Data Skew in Hadoop Leveraging Modern Portfolio Theory," in *SAC '18 Proceedings of the 33rd Annual ACM Symposium on Applied Computing.* ACM, 2018, pp. 175–182.

[84] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, E. M. Nguifo, and F. Nancy, "An Experimental Survey on Big Data Frameworks," *Future Generation Computer Systems*, vol. 86, no. September 2018, pp. 546–564, 2018. [Online]. Available: https://doi.org/10.1016/j.future.2018.04.032

[85] D. Choi, M. Jeon, N. Kim, and B.-D. Lee, "An Enhanced Data-Locality-Aware Task Scheduling Algorithm for Hadoop Applications," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3346–3357, 2018.

[86] W. Chen, I. Paik, and Z. Li, "Tology-Aware Optimal Data Placement Algorithm for Network Traffic Optimization," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2603–2617, 2015.

[87] S. M. Nabavinejad, M. Goudarzi, and S. Abedi, "MapReduce service provisioning for frequent big data jobs on clouds considering data transfers," *Computers and Electrical Engineering*, vol. 71, pp. 594–610, 2018. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2018.08.005

[88] A. Atrey, G. V. Seghbroeck, H. Mora, F. D. Turck, and B. Volckaert, "SpeCH : A scalable framework for data placement of data-intensive services in geo-distributed clouds," *Journal of Network and Computer Applications*, vol. 142, no. February, pp. 1–14, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.05.012

[89] X. Zhang, Z. Hu, M. Zheng, J. Li, and L. Yang, "A novel cloud model based data placement strategy for data- intensive application in clouds," *Computers and Electrical Engineering*, vol. 77, no. July 2019, pp. 445–456,

2019. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2018.07.007

[90] C. Li, J. Zhang, Y. Chen, and Y. Luo, “Data prefetching and file synchronizing for performance optimization in Hadoop-based hybrid cloud,” *The Journal of Systems & Software*, vol. 151, pp. 133–149, 2019. [Online]. Available: https://doi.org/10.1016/j.jss.2019.02.007

[91] N. Mansouri and M. M. Javidi, “A New Prefetching-aware Data Replication to Decrease Access Latency in Cloud Environment,” *The Journal of Systems & Software*, vol. 144, no. October 2018, pp. 197–215, 2018. [Online]. Available: https://doi.org/10.1016/j.jss.2018.05.027

[92] K. H. K. Reddy and D. S. Roy, “DPPACS : A Novel Data Partitioning and Placement Aware Computation Scheduling Scheme for Data-Intensive Cloud Applications,” *The Computer Journal*, vol. 59, no. 1, pp. 64–82, 2016.

[93] A. Abdul, R. Iqbal, A. Jame, M. Odetayo, and N. Shah, “NORA : Network Oriented Resource Allocation for Data Intensive Applications in the Cloud Environment,” in *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design.* IEEE, 2014, pp. 295–300.

[94] X. Ma, S. Member, X. Fan, and S. Member, “Dependency-aware Data Locality for MapReduce,” *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 667–679, 2015.

[95] L. Chunlin, W. Yaping, T. Hengliang, and L. Youlong, “Dynamic multi-objective optimized replica placement and migration strategies for SaaS applications in edge cloud,” *Future Generation Computer Systems*, vol. 100, no. May, pp. 921–937, 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.05.003

[96] C. Li, Y. Wang, Y. Chen, and Y. Luo, “Energy-efficient fault-tolerant replica management policy with deadline and budget constraints in edge-cloud environment,” *Journal of Network and Computer Applications*, vol. 143, no. April, pp. 152–166, 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.04.018

[97] R. Derouiche, Z. Brahmi, and M. M. Gammoudi, “FCA-based Energy Aware-data Placement Strategy for Intensive Workflow in Cloud Computing,” in *Engineering Systems 23rd International Conference on Knowledge-Based and Intelligent Information & FCA-based Energy Aware-data Placemen (Procedia Computer Science)*, vol. 159. Elsevier B.V., 2019, pp. 387–397. [Online]. Available: https://doi.org/10.1016/j.procs.2019.09.193

[98] A. Gandomi, M. Reshadi, A. Movaghar, and A. Khademzadeh, “HybSMRP : a hybrid scheduling algorithm in Hadoop MapReduce framework,” *Journal of Big Data*, vol. 6, no. 106, pp. 1–16, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0253-9

[99] C. Li, J. Zhang, and H. Tang, “Replica-aware task scheduling and load balanced cache placement for delay reduction in multi-cloud environment,” *The Journal of Supercomputing*, vol. 75, no. 5, pp. 2805–2836, 2018. [Online]. Available: https://doi.org/10.1007/s11227-018-2695-9

[100] Y. Yao, H. Gao, J. Wang, B. Sheng, and N. Mi, “New Scheduling Algorithms for Improving Performance and Resource Utilization in Hadoop YARN Clusters,” *IEEE Transactions on Cloud Computing*, p. 12, 2019.

[101] I. A. T. Hashem, N. B. Anuar, M. Marjani, E. Ahmed, H. Chiroma, A. Firdaus, M. T. Abdullah, F. Alotaibi, W. K. M. Ali, I. Yaqoob, and A. Gani, “MapReduce scheduling algorithms : a review,” *The Journal of Supercomputing*, vol. 76, no. 7, pp. 4915–4945, 2020.

[102] C.-t. Chen, L.-j. Hung, S.-y. Hsieh, R. Buyya, and Albert Y. Zomaya, "Heterogeneous Job Allocation Scheduler for Hadoop MapReduce Using Dynamic Grouping Integrated Neighboring Search," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 193–206, 2020.

[103] J. Wang, X. Li, R. Ruiz, J. Yang, and D. Chu, "Energy Utilization Task Scheduling for MapReduce in Heterogeneous Clusters," *IEEE Transactions on Services Computing*, vol. 1374, p. 14, 2020.

[104] A. Gandomi, A. Movaghar, M. Reshadi, and A. Khademzadeh, "Designing a MapReduce performance model in distributed heterogeneous platforms based on benchmarking approach," *The Journal of Supercomputing*, vol. 76, no. 9, pp. 7177–7203, 2020. [Online]. Available: https://doi.org/10.1007/s11227-020-03162-9

[105] J. Wang, X. Li, R. Ruiz, H. Xu, and D. Chu, "Allocating MapReduce workflows with deadlines to heterogeneous servers in a cloud data center," *Service Oriented Computing and Applications*, vol. 14, no. 2, pp. 101–118, 2020. [Online]. Available: https://doi.org/10.1007/s11761-020-00290-1

[106] M. Handaoui, J.-e. Dartois, L. Lemarchand, and J. Boukhobza, "Salamander : a Holistic Scheduling of MapReduce Jobs on Ephemeral Cloud Resources," in *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID) Salamander:*, 2020, pp. 320–329.

[107] S. Ghemawat, H. Gobioff, and S.-t. Leung, "The Google File System," in *SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles.* ACM, 2003, pp. 29–43.

[108] V. Ubarhande, A. M. Popescu, and H. Gonzalez-Velez, "Novel Data-Distribution Technique for Hadoop in Heterogeneous Cloud Environments," in *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, 2015, pp. 217–224.

[109] Y. Mao, H. Zhong, and L. Wang, "A fine-grained and dynamic mapreduce task scheduling scheme for the heterogeneous cloud environment," in *2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*. IEEE, 2015, pp. 155–158.

[110] Z. Zhang, L. Cherkasova, and B. T. Loo, "Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 4, pp. 38–50, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2788402.2788409

[111] F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni, "Dyscale: A mapreduce job scheduler for heterogeneous multicore processors," *IEEE Transactions on Cloud Computing*, vol. 5, no. 2, pp. 317–330, April-June 2017. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TCC.2015.2415772

[112] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.02.004

[113] A. Julio C S, C. Ivan, W. Kolberg, A. L. Tibola, L. B. Arantes, and C. R. Geyer, "MRA ++ : Scheduling and Data Placement on MapReduce for Heterogeneous Environments," *Future Generation Computer Systems*, vol. 42, pp. 22–35, 2014.

[114] J. Xie, S. Yin, X. Ruan, Z. Ding, and Y. Tian, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010, pp. 1–10.

[115] K. Deng, K. Ren, J. Song, D. Yuan, Y. Xiang, and J. Chen, "A clustering based coscheduling strategy for efficient scientific workflow execution in

cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 18, pp. 2523–2539, 2013.

[116] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic mapreduce scheduler for heterogeneous workloads," in *2009 Eighth International Conference on Grid and Cooperative Computing*. IEEE, 2009, pp. 218–224.

[117] Z. Ren, J. Wan, W. Shi, X. Xu, and M. Zhou, "Workload Analysis , Implications , and Optimization on a Production Hadoop Cluster : A Case Study on Taobao," *IEEE Transactions on Services Computing*, vol. 7, no. 2, pp. 307–321, 2014.

[118] M. Malik, K. Neshatpour, S. Rafatirad, and H. Homayoun, "Hadoop Workloads Characterization for Performance and Energy Efficiency Optimizations on Microservers," *IIEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 355–368, 2018.

[119] Z. Yu, W. Xiong, L. Eeckhout, Z. Bei, A. Mendelson, and C. Xu, "MIA : Metric Importance Analysis for Big Data Workload Characterization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1371–1384, 2018.

[120] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Resource and Deadline-Aware Job Scheduling in Dynamic Hadoop Clusters," in *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium, IPDPS 2015*, 2015, pp. 956–965.

[121] N. S. Naik, A. Negi, T. B. B.R., and R. Anitha, "A data locality based scheduler to enhance MapReduce performance in heterogeneous environments," *Future Generation Computer Systems*, vol. 90, no. 2019, pp. 423–434, 2019. [Online]. Available: https://doi.org/10.1016/j.future.2018.07.043

[122] Q. Zhang, M. F. Zhani, Y. Yang, R. Boutaba, and B. Wong, “Prism: Fine-grained resource-aware scheduling for mapreduce,” *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 182–194, April-June 2015. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TCC.2014.2379096

PUBLICATIONS

Conferences

1. Gurwinder Singh, Anil Sharma, ”A Review on Big Data: Technologies, Opportunities, and Challenges.”, Proceedings of International Conference on Science and Technology: Trends and Challenges(ICSTTC-2018), 184-188. (Published)

2. Anil Sharma, Gurwinder Singh, ”A Review on Data locality in Hadoop MapReduce”, Proceedings of 5th International Conference on Parallel, Distributed and Grid Computing (PDGC-2018),IEEE Xplore (June-2019) 723-728. (Scopus Indexed)(Published)

3. Anil Sharma, Gurwinder Singh, ”A Review of Scheduling Algorithms in Hadoop”, Proceedings of 2nd International Conference on Recent Innovations in Computing (ICRIC-2019), Lecture Notes in Electrical Engineering-597 (November-2019) 125-135. (Scopus Indexed)(Published)

4. Anil Sharma, Gurwinder Singh and Shabnum Rehman, ”A Review of Big Data Challenges and Preserving Privacy in Big Data”,Proceedings of 2nd International Conference on Data and Information Sciences (ICDIS-2019), Lecture Notes in Networks and Systems-94 (2020) 57-66. (Scopus Indexed) (Published)

Journals

1. Gurwinder Singh, Anil Sharma, ”A Study on Big Data Tools and Applications”, International Journal of Research in Electronics and Computer Engineering (A Unit Of I2OR) (April-June 2018) 477-479. (UGC approved) (Published)

2. Gurwinder Singh, Anil Sharma,”Tuning Hadoop Parameters for Heterogeneous Multi-node Cluster”, International Journal of Future Generation Communication and Networking (March-2020)507-518. Web of Science(Published)

3. Gurwinder Singh, Anil Sharma, Rathinaraja Jeyaraj, Anand Paul, ”Handling Non-Local Executions to Improve MapReduce Performance using Ant Colony Optimization”, IEEE Access (June-2021)96176-96188. (Published) (SCIE INDEXED)(Impact Factor: 3.7)

4. Gurwinder Singh, Anil Sharma, Rathinaraja Jeyaraj, Anand Paul, ”Heterogeneous Performance Based Data Block Placement in Virtualized Environment”, Cluster Computing: The Journal of Networks, Software Tools and Applications. (SCIE INDEXED) [Communicated]