IMPROVING HADOOP PERFORMANCE BY USING METADATA OF RELATED JOBS IN TEXT DATASETS VIA ENHANCING MAPREDUCE WORKFLOW

Hamoud Alshammari

Under the Supervision of Dr. Hassan Bajwa

DISSERTATION
SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

AND ENGINEERING

THE SCHOOL OF ENGINEERING

UNIVERSITY OF BRIDGEPORT

CONNECTICUT

April, 2016

IMPROVING HADOOP PERFORMANCE BY USING METADATA OF RELATED JOBS IN TEXT DATASETS VIA ENHANCING MAPREDUCE WORKFLOW

Hamoud Alshammari

Approvals

Committee Members

Name	Signature	Date
Dr. Hassan Bajwa		04/18/16
Dr. JeongKyu Lee		04/18/16
Dr. Navarun Gupta		4/18/16
Dr. Miad Faezipour		04,18,2016
Dr. Xingguo Xiong		04/18/16
Dr. Adrian Rusu		04/18/16

Ph.D. Program Coordinator

Dr. Khaled M. Elleithy — 5/10/16

Chairman, Computer Science and Engineering Department

Dr. Ausif Mahmood — 5-10-2016

Dean, School of Engineering

Dr. Tarek M. Sobh — 5/25/16

IMPROVING HADOOP PERFORMANCE BY USING METADATA OF RELATED JOBS IN TEXT DATASETS VIA ENHANCING MAPREDUCE WORKFLOW

IMPROVING HADOOP PERFORMANCE BY USING METADATA OF RELATED JOBS IN TEXT DATASETS VIA ENHANCING MAPREDUCE WORKFLOW

ABSTRACT

Cloud Computing provides different services to the users with regard to processing data. One of the main concepts in Cloud Computing is BigData and BigData analysis. BigData is a complex, un-structured or very large size of data. Hadoop is a tool or an environment that is used to process BigData in parallel processing mode. The idea behind Hadoop is, rather than send data to the servers to process. Hadoop divides a job into small tasks and sends them to servers. These servers contain data, process the tasks and send the results back to the master node in Hadoop.

Hadoop contains some limitations that could be developed to have a higher performance in executing jobs. These limitations are mostly because of data locality in the cluster, jobs and tasks scheduling, CPU execution time, or resource allocations in Hadoop.

Data locality and efficient resource allocation remains a challenge in cloud computing MapReduce platform. We propose an enhanced Hadoop architecture that

reduces the computation cost associated with BigData analysis. At the same time, the proposed architecture addresses the issue of resource allocation in native Hadoop.

The proposed architecture provides an efficient distributed clustering approach for dedicated cloud computing environments. Enhanced Hadoop architecture leverages on NameNode's ability to assign jobs to the TaskTrakers (DataNodes) within the cluster. By adding controlling features to the NameNode, it can intelligently direct and assign tasks to the DataNodes that contain the required data.

Our focus is on extracting features and building a metadata table that carries information about the existence and the location of the data blocks in the cluster. This enables NameNode to direct the jobs to specific DataNodes without going through the whole data sets in the cluster. It should be noted that newly build lookup table is an addition to the metadata table that already exists in the native Hadoop. Our development is about processing real text in text data sets that might be readable such as books, or not readable such as DNA data sets. To test the performance of proposed architecture, we perform DNA sequence matching and alignment of various short genome sequences. Comparing with native Hadoop, proposed Hadoop reduced CPU time, number of read operations, input data size, and another different factors.

ACKNOWLEDGEMENTS

In the name of Allah most gracious most merciful, I thank him for everything I do and everyone he has allowed me to have in my life to help me in achieving my doctoral degree. I would like to start with my academic mentor Dr. Hassan Bajwa. I am grateful for his endless support; his guidance and leadership are qualities of true leader in the field. The commitment and dedication he provides to his advisees inspires me to be a leader in the field. His support has allowed me to reach this point and for that thank you.

Also, I would like to thank my Co-advisor Dr. Jeongkyu Lee for his support during my classes and research. A special appreciation goes to Dr. Khaled Elleithy for his support and direction during my study. To all my committee members, thank you for your points and directions that I have received from you to develop my work.

To my mother, I ask Allah to save you, give you the strength, and to live healthy during the rest of your life. I appreciate your patience and support during all of my life. To my wife and my kids Sari and Mariam I love you and ask Allah to save you and save our family. To my brothers, sisters, and friends I appreciate your support at all times and I wish you all the best in your lives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Human Genome Project is arguably the most important scientific project that has produced the most valuable dataset scientific community has ever seen. Translating the data to meaningful information requires substantial computational power and efficient algorithms to identifying the degree of similarities among multiple sequences [1]. Sequential data applications such as DNA sequence aligning usually require large and complex amounts of data processing and computational capabilities [2]. Efficiently targeting and scheduling of computational resources is also required to such complex problems [3]. Although, some of these data are readable by humane, it can be very complex to be understood and processed using the traditional techniques [3, 4]. Availability of open source and commercial cloud computing parallel processing platforms have opened new avenues to explore structured, semi-structured or unstructured genomic data [5]. Before we go any further, it is necessary to define certain definitions that are related to BigData, Hadoop and MapReduce.

1.1 What is BigData?

It is either relational database (Structured) such as stock market data or non-relational database (Semi-structured or Unstructured) such as social media data or DNA data sets. Usually, very large in size, BigData cannot be processed using traditional

processing techniques such as Relational Database Management Systems -RDBMS [6]. A survey that has been done explaining different studies that are exploring some issues in BigData, and it explains the 4V model of BigData.

4V's of BigData are 1) Volume of the data, which means the data size, and we are talking about zetabyte these days. 2) Velocity, which means the speed of the data or streaming of the data. 3) Varity of the data, which means the data forms that different applications deal with such as sequence data, numeric data or binary data. 4) Veracity of the data, which means the uncertainty of the status of the data or how clear is the data to these applications [7].

Different challenges in BigData have been discussed in [8] and they are described as a technical challenges such as the physical storage that stores the BigData and reduce the redundancy. Also, the process of extracting the information, the cleaning the data is one challenge, data integration, data aggregation and representation.

1.2 What are Hadoop and MapReduce?

Hadoop is an Apache open-source software framework that is written in Java for distributed storage and distributed processing. It provides solutions for BigData processing and analysis. It has a file system that provides an interface between the users' applications and the local file system, which is the Hadoop Distributed File System HDFS. Hadoop distributed File System assures reliable sharing of the resources for efficient data analysis [9].

The two main components of Hadoop are: (i) Hadoop Distributed File System (HDFS) that provides the data reliability (distributed storage) and (ii) the MapReduce that provides the system analysis (distributed processing) [9, 10]. Relying on the principle that "moving computation towards data is cheaper than moving data towards computation" [11], Hadoop employs HDFS to store large data files across the cluster. There are several reasons that make companies use Hadoop, some of them are it is an open source and can be run on commodity hardware, the initial cost savings are dramatic and continue to grow as your organizational data grows, and it has a robust Apache community behind it that continues to contribute to its advancement.

MapReduce provides stream reading access and runs tasks on a cluster of nodes and provides a data managing system for a distributed data storage system [12]. MapReduce algorithm has been used for applications such as generating search indexes, document clustering, access log analysis, and different other kinds of data analysis [13].

"Write-once and read-many" approach permits data files to be written only once in HDFS and then allows it to be read many times over with respect to the numbers of jobs [9]. During the writing process, Hadoop divides the data into blocks with a predefined block size. The blocks are then written and duplicated in the HDFS. The blocks can be duplicated a number of times based on a specific value which is 3 times by default [14].

In HDFS, the cluster that Hadoop is installed in is divided into two main components, which are (i) the master node called NameNode and (ii) the slaves called DataNodes. In Hadoop cluster single NameNode is responsible for overall management

of the files system including saving the data and directing the jobs to the appropriate DataNodes that store related application data [15]. DataNodes facilitates Hadoop MapReduce to process the jobs with streaming execution in a parallel processing environment [9, 16].

Running on the master node, JobTracker coordinates and deploys the applications to the DataNodes with TaskTracker services for execution and parallel processing [14]. Each task is executed in an available slot in a worker node, which is configured with a fixed number of map slots, and another fixed number of reduced slots. The input to MapReduce is in text format, so the data must be read as text files, and the output also is in text file format [9].

1.3 What is MapReduce Job?

A MapReduce job is an access and process-streaming job that splits the input dataset into independent chunks (blocks) and stores them in HDFS. It has two main tasks Map and Reduce, which are completely in parallel manner. Storing data in HDFS has many forms, one of them is a <Key, Value> concept to determine the given parameter and retrieve the required result as a value in the end of the whole job.

Figure 1.1 explains MapReduce example of wordcount as a common example to apply MapReduce in such unstructured data like books. As input file, it consists of sequence of strings that are separated by space, so we can consider the space as a delimiter that separate words.

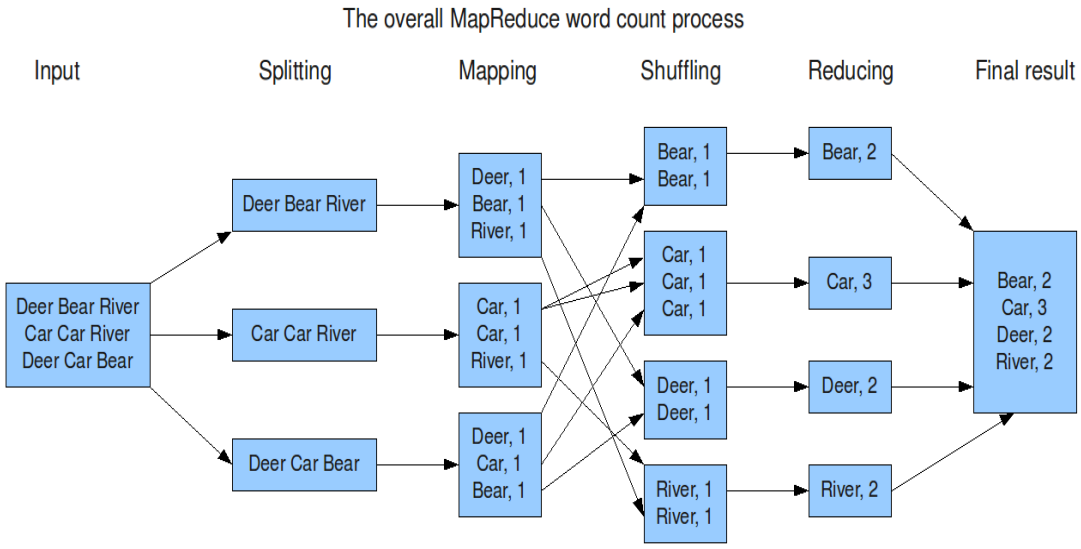


Figure 1.1: Overall MapReduce WordCount MapReduce

First step, Hadoop divides the data to blocks (Splitting phase). Then, Mapping phase dose <key, value> for each word (e.g. <Deer, 1>. Then, Shuffling phase collects the values of the same key to be in one intermediate result. After that, Reducing phase does the addition of the values to have one final value for each key. Finally, NameNode provides a final result that has all keys and their values as a one final result from the MapReduce job.

HDFS cluster is composed of a centralized indexing system called NameNode and its data processing units called DataNodes; together they form a unique distributed file system. NameNode plays an important role in supporting the Hadoop Distributed File System by maintaining a File-Based block index map; this map is responsible for locating all the blocks related to the HDFS. HDFS is the primary storage system. It creates multiple replicas of data blocks and is further responsible for distributing data blocks throughout a cluster to enable reliable and extremely rapid computations [17]. Figure 1.2

shows the native Hadoop architecture that gives an overview on Cloud Computing architecture too.

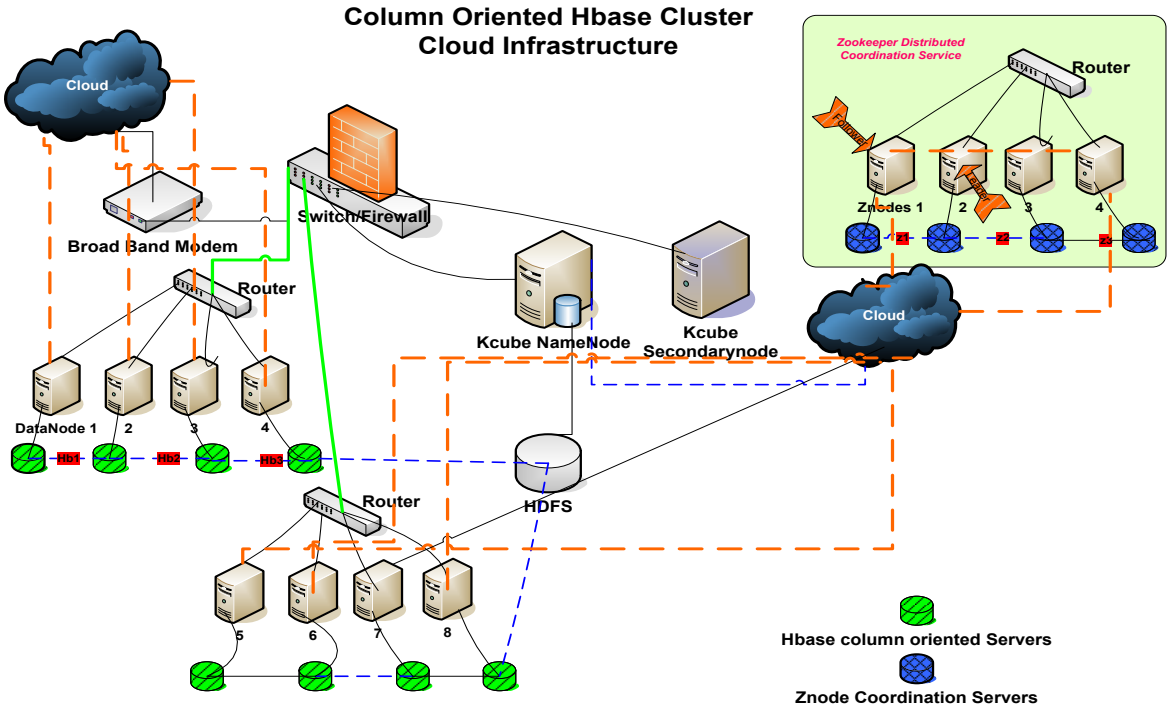


Figure 1.2: Native Hadoop Architecture.

ZooKeeper is critical component of the infrastructure, it provides coordination and messaging across applications [18]. The ZooKeeper capabilities include naming, distributing synchronization, and group services. Hadoop framework leverages on large-scale data analysis by allocating data-blocks among distributed DataNodes.

Hadoop Distributed File System shown in Figures 1.2 and 1.3 allows the distribution of the data set into many machines across the network that can be logically prepared for processing. HDFS adopted “Write-Once and Read-Many” model to store data in distributed DataNodes. NameNode is responsible for maintaining namespace hierarchy, managing data blocks and DataNodes mapping [19]. Once job information is

received from the client, NameNode provides a list of available data nodes for the job. NameNode maintains the list of available data nodes and is responsible for updating the index list when a DataNode is unavailable due to failure in hardware or network issues. A heartbeat is maintained between the NameNode and the DataNodes to check the keep-alive status and health of the HDFS [20]. Client writes data directly to the DataNode, this is shown in Figure 1.3.

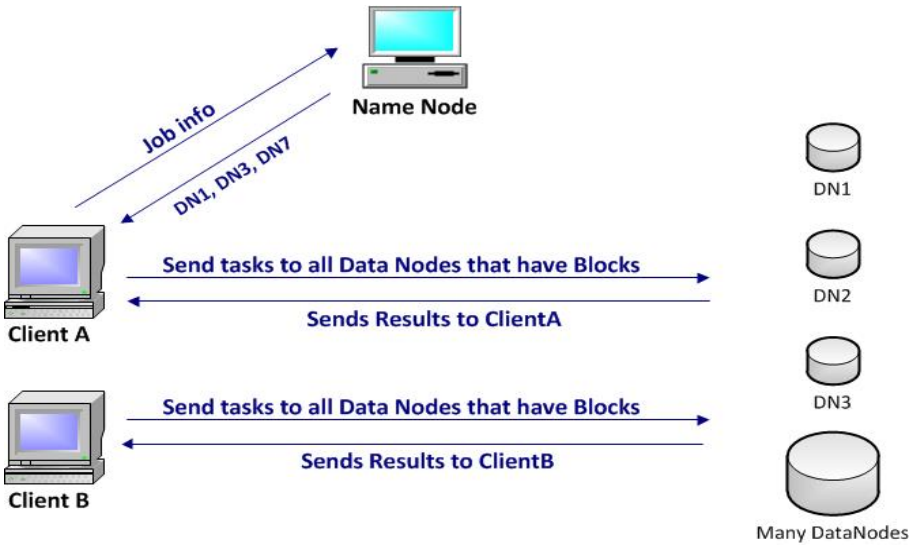


Figure 1.3: DataNodes and Task Assignment.

HDFS is architected to have the block fault and replication tolerance. NameNode is the responsible for maintaining a healthy balance between disk processing on various DataNodes and has the ability to restore failed operations on the remote blocks. Data Locality is achieved through cloud file distribution, the file processing is done local to each machine, and any failed reads from the blocks are recovered through block replication.

The process of selecting the mappers and reducers is done by the JobTracker immediately after lunching a job [16, 21]. A client operating on the HDFS has network file transparency, and the distribution of blocks on different machines across the Cloud is transparent to the client. HDFS is oriented towards hardware transparency. Processing DataNodes can be commissioned or decommissioned dynamically without affecting the client.

1.4 Comparison of Hadoop with Relational Database Management Systems –RDBMS

Hadoop MapReduce has been the technology of choice for many data intensive applications such as email spam detections, web indexing, recommendation engines, predictions on weather and financial services. Though Relational Database Management Systems (RDBMS) can be used to implement these applications, Hadoop is a technology of choice due to its higher performance despite RDBMS's higher Functionality [22].

RDBMS can process the data in real-time with high level of availability and consistency. Organizations such as banks, e-business and e-commerce websites employ RDMBS to provide reliable services. However, RDBMS does not work seamlessly with unstructured heterogeneous BigData and processing BigData can take a very long time.

On the other hand, scalable, high-performing massive parallel processing Hadoop platform can store and access petabytes of data on thousands of nodes in a cluster. Though comparing to RDMBS, processing of BigData is efficient in Hadoop. Hadoop and MapReduce is not intended for real-time processing [23]. The RDMBS and a Hadoop cluster can be complementary and coexist in the data warehouse together.

1.5 Hadoop for Bioinformatics Data

We are using the DNA sequence data sets as an example to test the proposed work and it is not the only data set that we can use but it is a very good data in terms of structured data and has high value in researches. While there is many applications other than Hadoop can handle the Bioinformatics data, Hadoop framework was primarily designed to handle unstructured data [24].

Bioinformatics tools such as (BLAST, FASTA etc) can process unstructured genomic data in parallel workflow [5]. Most of the users are not trained to modify the existing applications to incorporate parallelism effectively [25, 26]. Parallel processing is crucial in rapid sequencing of large unstructured dataset that can be both time-consuming and expensive. Aligning multiple sequences using sequence alignment algorithms remains a challenging problem due to quadratic increases in computation and memory limitation [27]. BLAST-Hadoop is one implementation of sequence aligning that uses Hadoop to implement BLAST in DNA sequences, and there is more information about this experiment in [1].

1.6 Research Problem and Scope

Searching for sequences or mutation of sequences in a large unstructured dataset can be both time-consuming and expensive. Sequence alignment algorithms are often used to align multiple sequences. Due to memory limitation, aligning more than three to four sequences is often not allowed by traditional alignment tools.

As expected, Hadoop cluster with three nodes was able to search the sequence data much faster than single node, it is expected that search time will reduce as the

number of DataNodes are increased in the cluster. However, when we execute a MapReduce job in the same cluster for more one time, each time it takes the same amount of time. This study aims to present this problem and propose a solution that would improve the time involved in the execution of MapReduce jobs.

Many Big Data problems such as genomic data focus on similarities, sequences and sub-sequences searches. If a sub-sequence is found in specific blocks in a DataNode, sequence containing that sub-sequence can only exist in the same DataNode. Since current Hadoop Framework does not support caching of job execution metadata, it ignores the location of DataNode with sub-sequence and reads data from all DataNodes for every new job [28]. Shown in Figure 1.2, Client A and Client B are searching for similar sequence in BigData. Once Client A finds the sequence, Clint B will also go through the whole BigData again to find the same results. Since each job is independent, clients do not share results. Any client looking for Super sequence with sub-sequence that has already been searched will have to go through the BigData again. Thus the cost to perform the same job will stay the same each time. The outlines and the scope behind this research is as follows:

- Discuss the concept of BigData and the need to use new approaches to process it.
- Overview of the type of data that Hadoop can process to get better result than the traditional ways of computing.
- Discuss the architecture and workflow of existing Hadoop MapReduce algorithm and how users can develop their work based on the size of their data.

- Investigate and discuss the benefits and limitations of existing Hadoop MapReduce algorithm to come up with a possible solution on how to develop the MapReduce performance.
- Propose an enhancing process to the current Hadoop MapReduce to improve the performance by reducing the CPU execution time and power costs.

1.7 Motivation behind the Research

While there are many applications of Hadoop and BigData, Hadoop framework was primarily designed to handle unstructured data. Massive genomic data, driven by unprecedented technological advances in genomic technologies, have made genomics a computational research area. Next Generation Sequencing (NGS) technologies produce High Throughput Short Read (HTSR) data at a lower cost [29]. Scientists are using innovative computational tools that allow them rapid and efficient data analysis [30, 31] .

DNA genome sequence consists of 24 chromosomes. The compositions of nucleotides in genomic data determine various traits such as personality, habits, and inherited characteristics of species [32]. Finding sequences, similarities in sequences, sub-sequences or mutation of sequences are important research areas in genomic and bioinformatics. Scientists need to find sub-sequences within chromosomes to determine either some diseases or proteins frequently [33]. Therefore, one of the important motivations is to speed up processing time like finding sequence process in DNA. The need to reduce different costs like those spent in power for the service provider, paying the users and data size for the computation process is also a pressing motivation.

1.8 Potential Contributions of the Proposed Research

Different research areas can use the current Hadoop architecture and may face the same problem frequently. Scientific data and others that employ the use of data sequence that need more jobs and processes may spend more time and power doing their research. This contribution is to improve the Hadoop MapReduce performance by enhancing the current one by getting the benefits from the related jobs that share some parameters with the new job.

Building tables that store some results from the jobs gives us ability to skip some steps either in reading the source data or do some processing on the shared parameters. In addition, by enhancing the Hadoop we can reduce processing time, data size to read and other parameters in Hadoop MapReduce environment.

CHAPTER 2: LITERATURE SURVEY

Hadoop is considered as a new technology that provides processing services for BigData issues in cloud computing, thus, research in this field is considered as a hot topic. Many studies have discussed and developed different ways to improve the Hadoop MapReduce performance from different considerations or aspects.

Cloud Computing is emerging as an increasingly valuable tool for processing large datasets. While several limitations such as bandwidth, security and cost have been mentioned in the past [2, 34], most of these approaches are only considering public cloud computing and ignores one of the most important features of cloud computing " write-once and read-many".

Many studies have discussed different solutions that can help to improve Hadoop performance. These improvements could be implemented in the two major components of Hadoop, which are MapReduce, which is the distributed parallel processing algorithm, and the HDFS, which is the distributed data storage in the cluster.

The first area of improving Hadoop performance is optimizing job scheduling and execution time in MapReduce jobs [35-46]. In this area of development, many studies have improved the Hadoop performance and they have achieved positive results, as we will discuss later on in our research. This part also includes optimizing the process of cashing data in the MapReduce job.

In addition, there are different studies that have focused on the data in cloud computing and improving its related issues such as data locality and type [11, 15, 25, 33, 47-52]. This part of the research focuses on the data based on data type or based on the data location. Other studies focused on the Cloud Computing environment that Hadoop works in and develops this environment to be more suitable for Hadoop [53-59].

In this part, we present a couple of these areas of research and discuss the important points that each one proposed in order to improve Hadoop performance as well as the studies' limitations and results.

2.1 Optimizing Job Execution and Job Scheduling Processes

One of the important features of Hadoop is the process of MapReduce job scheduling and execution processes. Different studies have done some of improvements and have come up with good results based on some assumptions. We will discuss some of these studies as follows:

2.1.1 Optimizing Job Execution Process

Many applications need a quick response time with high performance to get high throughput especially for short jobs. So, one of the important features of the execution job is the MapReduce job response time. In SHadoop, this study proposed changing in MapReduce job performance by enhancing and optimizing the job and tasks execution mechanism [35].

The study developed the work by optimizing two parts of Hadoop workflow. First, they optimized the time cost of initialization and termination the job stages. Second,

they optimized the sending of the Heartbeats-based communication mechanism that communicates between the JobTracker and TaskTrackers on the cluster to have a developed mechanism that accelerates the task scheduling and execution performance.

The experiments show that, the developed Hadoop "SHadoop" has improved the execution performance by around 25% on average comparing with the native Hadoop without loosing the scalability and speedup especially for short jobs. The chart in Figure 2.1 explains the performance between standard Hadoop and developed Hadoop that is proposed in SHadoop:

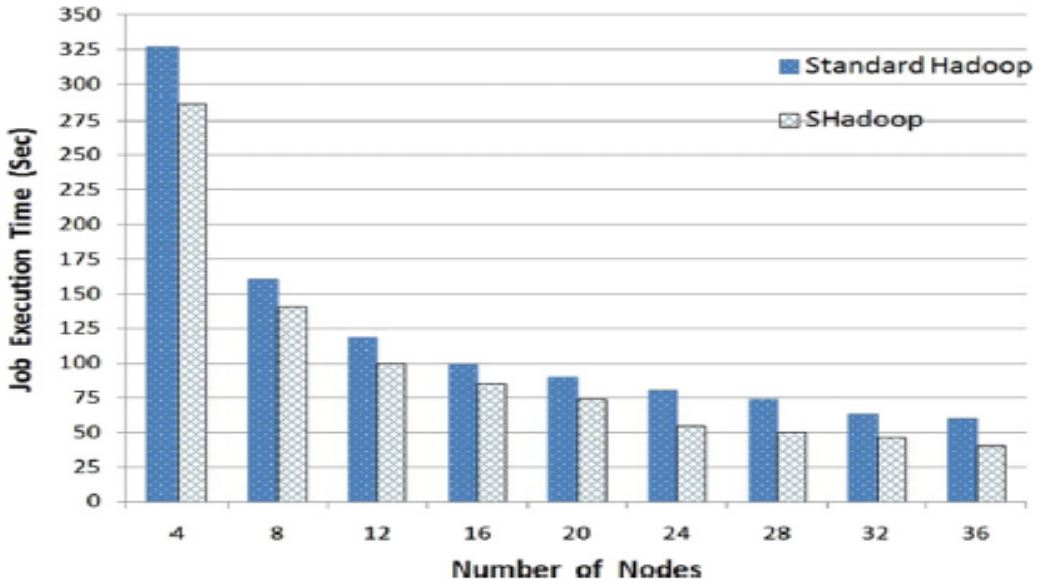
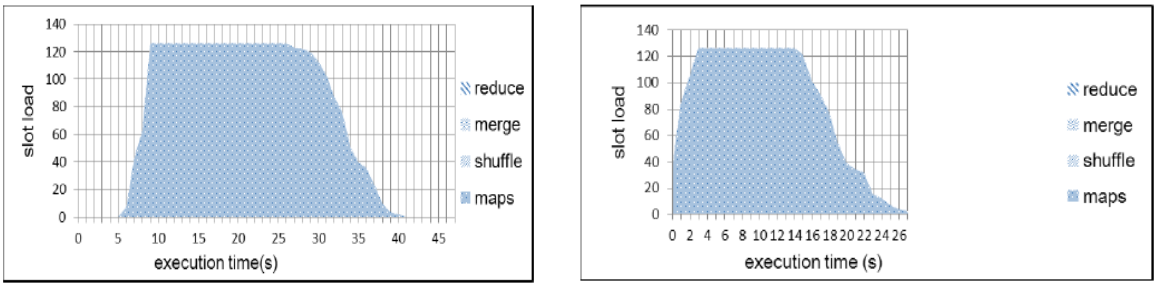


Figure 2.1: Execution time of wordcount benchmark in SHadoop/StandardHadoop with different number of nodes.

The technique of optimizing the time of job initialization and termination is followed by [40] to develop Hadoop performance by speedup the setup and cleanup

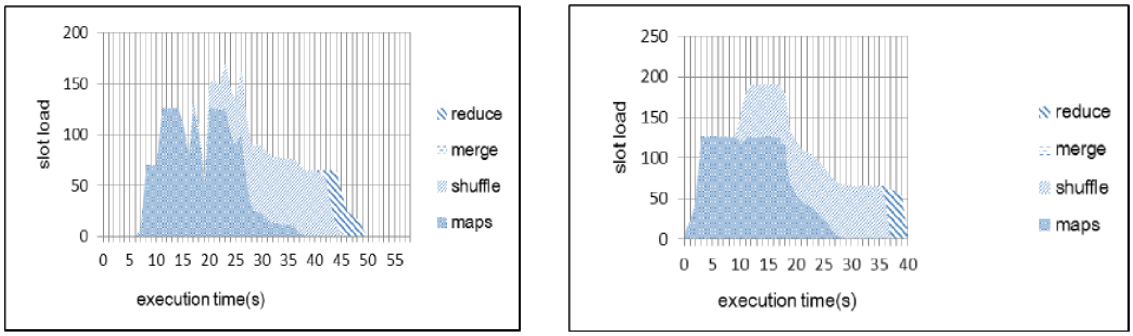
tasks. This study, also, optimized the mechanism of assigning the tasks from pull-model to push-model.

The results from experiments show that the work speeds up the job execution time by 23% on average comparing with the standard Hadoop. The compression in the work shown in Figure 2.2 and Figure 2.3:



(a) Standard Hadoop (b) Proposed Hadoop

Figure 2.2: Comparing in Execution time using Push-model between (a) Standard Hadoop and (b) Proposed Hadoop in Map phase.



(a) Standard Hadoop (b) Proposed Hadoop

Figure 2.3: Comparing in Execution time using Push-model between (a) Standard Hadoop and (b) Proposed Hadoop in Reduce phase.

As shown in Figures 2.2 and 2.3, after further applying the optimization of the posh-model task assignment and instant message communication mechanisms, the total

job execution time is further shortened to 27 seconds for the map phase extension BLAST, and to 40 seconds for the reduce phase extension BLAST.

2.1.2 Optimizing Jobs Scheduling Process

Zookeeper [18] is one component of Hadoop ecosystem that is considered as a centralized control service that maintains a couple of services such as configuration information, naming, provides grouping information in the cluster, and job scheduling since it is one of the Hadoop configurable services.

Different studies have been discussed in a survey about MapReduce job scheduling algorithms [60]. In this work, the authors listed the most used scheduling algorithms that control the order and distribution of users, tasks, and jobs. So, by having better scheduling algorithms, we can improve Hadoop performance. Most of the proposed algorithms in that survey have been developed to meet some requirements under some circumstances and assumptions.

Job scheduling process in Hadoop follows the First-In First-Out (FIFO) algorithm, which may cause different weaknesses in Hadoop performance. Zaharia and others, provide some solutions to improve the job scheduling in Hadoop and they implied it to Facebook [38]. Some systems have services that allow for multiple users on the same system. In [38] the authors discuss job scheduling for multi-users on the system. This study is considered as one of the common studies in this section. The authors focused on sharing a MapReduce environment between many users and described that as attractive, because it enables sharing common large data between them. The traditional scheduling

algorithms perform a very poor process in MapReduce because of the data locality and dependency between map tasks and reduce tasks.

The study experimented with scheduling for MapReduce in Facebook with a 600-node multi-user data warehouse in Hadoop. Two techniques have been developed which, included delay scheduling and copy-compute splitting, this provided very good results, which improved the throughput and response time by many factors.

Different ideas have been discussed in [37] which is about having information about each task and node in the cluster, so the idea is to schedule algorithms for distributing the sources of Hadoop between nodes in the cluster. Improving the process by assigning and selecting best and most suitable task for the node. Consequently, the authors avoided the overloading on any node in the cluster and utilized maximum resources on the node to control and decrease the accessing rate between tasks for the resources. This reduces the job runtime process.

Also, in this paper, the authors discussed the ability of scheduling jobs on Hadoop by adding an algorithm for assigning features to jobs. Figure 2.4, shows the comparison between two algorithms that are applied in the cluster to show a decrease in the runtime of jobs. The amount of the saving in the runtime of the jobs increases as the number of jobs increase. Transferring data is one of the issues that paper [39] has discussed. While transferring data is considered as a cornerstone of the process, transferring un-necessary data can also be considered an important issue that we can manipulate. By placing the tasks on the nodes that carry the data, we can improve the performance, which means it is all about the locality of the input data.

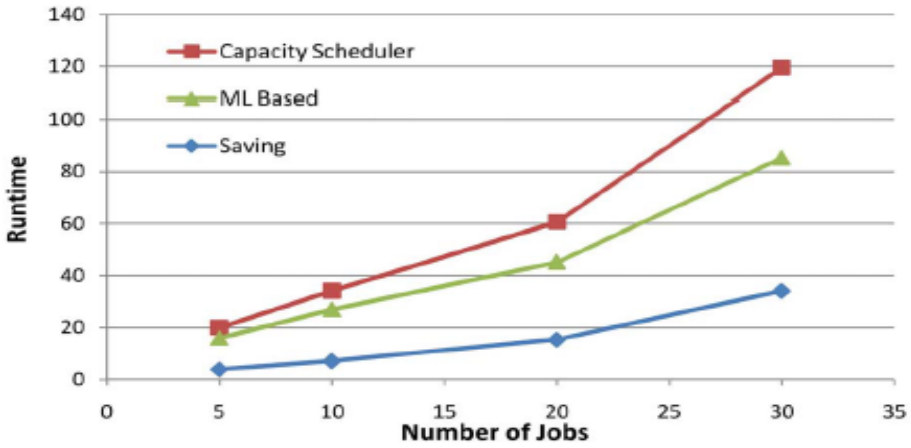


Figure 2.4: Comparison of runtime (in hours) of the jobs between Capacity and machine learning based algorithms.

The authors in [39] integrated their work into FIFO, which is the default algorithm in Hadoop job scheduling and into Hadoop Fair Scheduling algorithm. Some comparisons have been done between the proposed technique in this study and the native and other proposed techniques. The experiment shows its results to the map tasks have highest data locality rate and lower response time.

Another way to reschedule the map tasks has been discussed in [41] which is about prediction and selection of the task who has the highest probability to be executed in a data node based on the location and select it to be the first in the list after calculating all map tasks probabilities and rescheduling them. After implementing the proposed solution, one of the results was that there was a 78% reduction of the map tasks processed without node locality, a 77% reduction in the network load caused by the tasks, and improved the performance of Hadoop MapReduce when comparing with the default task scheduling method in native Hadoop.

MapReduce uses the parallel computing framework to execute the jobs within the cluster. Some of the tasks may become struggled somehow, which means they take long time to finish the whole job, and that affects the total cluster throughput. There are different approaches that deal with this kind of problem one of them is the Maximum Cost Performance (MCP) that is presented in [42] which improves the effectiveness of speculative execution significantly. To determine the slow task, this study provides a strategy by calculate the progress rate and the process bandwidth. Also, they predict the process speed and the remaining runtime by using exponentially weighted moving average (EWMA).

For the slow tasks, they need to choose a proper machine to store these tasks to run them. The proposed strategies and taking the data locality and data skew into consideration, they can choose the proper worker node for backup tasks. They execute different applications on a cluster contains about 30 physical machines, and the experiments show that MCP can run jobs up to 39% faster and improve the cluster throughput by up to 44 percent compared to Hadoop-0.21.

2.1.3 Optimizing Hadoop Memory and Cashing Data Control

Memory systems could have many issues that could be addressed to improve system performance. In Hadoop, Apache performs a centralized memory approach it is implemented to control the cashing and resources [43]. There are different approaches that discuss memory issues. ShmStreaming [44] introduces a Shared memory Streaming schema to provide lockless FIFO queue that connects Hadoop and external programs.

ShmStreaming claim that they have improved the performance by 20-30% comparing with native Hadoop streaming implementation.

Apache Hadoop supports centralized data cashing. However, some studies utilize a distributed cashing approach to improve Hadoop performance [36, 45]. The studies identified enhanced architecture of Hadoop as an advancement that would accelerate job execution time. The proposed solutions are the efforts to speed up the access to map tasks and reduce tasks by using distributed memory cache. By using the distributed cache memory, the process of shuffling data between tasks improves the performance of MapReduce jobs. This decreases the time that is spent transferring data and increases the utility of the cluster.

Another study that focuses on cashing data in memory is L. Lum et.al [46]. In this study, the researchers claim that all the data cannot be cached in the memory due to memory limitations, thus the cashed data should be selected only when needed to improve the performance of Hadoop. The proposed solution provides unbinding technology, which could delay the binding of the programs and data together until the real computation starts. With the strategy of cashing and prefetching, Hadoop can get higher performance.

Unbinding-Hadoop is the framework that this study provides to decide the input data for map tasks in the map start-up phase not at the job submission phase. Also, prefetching does the same development in performance comparing with native Hadoop. The Unbinding-Hadoop performs better results after some experiments and reduces the

execution time by 40.2% for word-count application and by 29.2% for k-means algorithm as shown in Figure 2.5.

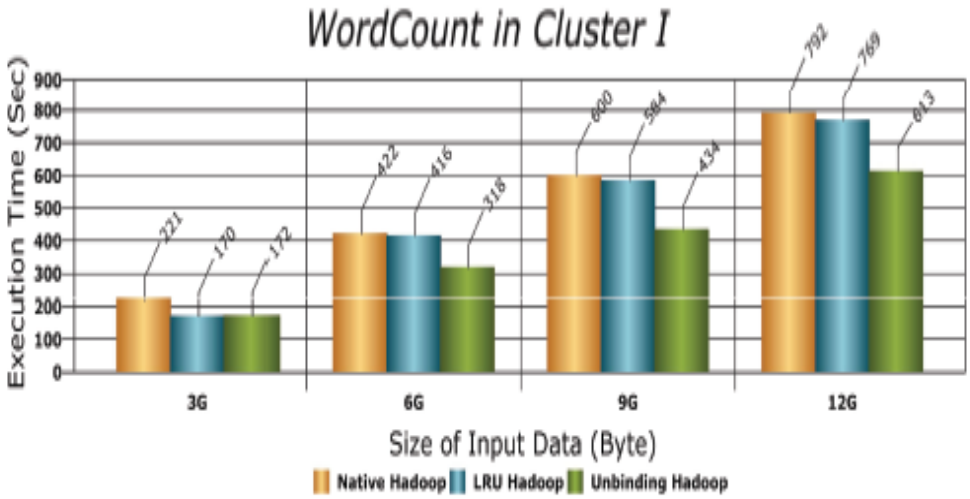


Figure 2.5: Cash system in Unbinding-Hadoop.

Table 2.1 shows a summary of comparisons between studies related to the improvement of Hadoop performance via developing the job execution time, job scheduling, and cashing data in the memory systems.

Table 2.1: Improvement in Job Scheduling, Execution Time, and Cashing Data

Study	*Problem Definition*	*Improvements*	*Results*
M. Zaharia and others	Sharing Job scheduling between multi-users of MapReduce in the cluster	Enables sharing common large data between users Delay scheduling and copy-compute splitting	Improve the system throughput and response time
R. Nanduri and others	Scheduling algorithms to distribute the sources of Hadoop between nodes in the cluster	Assigning and selecting best and most suitable task for the node	Avoid the overloading on any node in the cluster and utilized maximum resources on the node to decrease the accessing rate between tasks for the resources to reduce the job runtime process.
C. He and others "Matchmaking"	Hadoop Fair Scheduling algorithm	Placing the tasks on the nodes that carry the required data	Map tasks have higher data locality rate and lower response time comparing with native Hadoop
Z. Xiaohong and others	Comparing the probabilities of jobs	Prediction and selection of the task who has the highest probability to be executed in a data node based on the location	78% reduction of the map tasks processed without node locality 77% reduction in the network load caused by the tasks
R. Gu and others "SHadoop"	Optimizing job execution mechanism	Optimizing the time cost of job initialization and termination Optimizing the sending of Heartbeats-based communication mechanism	Reduce the execution time up to 25%
Y. Jinshuang and others	Optimizing job assigning mechanism	Optimizing the mechanism of assigning the tasks from pull-model to push-model	Speed up the job execution time by 23% on average
L. Longbin and others "ShmStreaming"	Optimizing cashing data	Provide shared memory Streaming schema to provide lockless FIFO queue that connects Hadoop and external programs	Improved the performance by 20-30% comparing with native Hadoop
S. Zhang and others	Distribute Memory Cashing	Process of shuffling data between tasks improves the performance of MapReduce jobs	Decreases the time that is spent transferring data and increases the utility of the cluster
L. Kun and others	Unbinding Technology	Delay the binding of the programs and data together until the real computation starts	Reduces the execution time by 40.2% for word-count application and by 29.2% for k-means algorithm

2.2 Improving Data considerations in Cloud Computing

Location of the input data has been determined in current Hadoop to be located in different nodes in the cluster. Since there is a default value for duplication of the data, which is 3 times, Hadoop distributes the duplicated data into different nodes in different network racks.

This strategy helps for various reasons, one of which is for the false tolerant issue to have more reliability and scalability. However, the default data distribution location strategy causes some poor performance in terms of mapping and reducing tasks. In this section, we will discuss a couple of studies that introduce different strategies related to improving the Hadoop performance by controlling the data distribution location, type, and size strategies.

2.2.1 Improving performance Based on Data Type

SciHadoop [15] focuses on a specific type of data such as scientific data. The use of specific data in this work has made it incompatible with other data formats. Using native Hadoop for scientific data analysis is not an attractive choice due to the required format transformation and data management costs. SciHadoop presents a critical problem related to the proficiency and locality of the data in the Hadoop MapReduce cluster. In addition to the DataNodes, there can be physical location problems in a cluster [15, 49], SciHadoop discusses the problem that is associated with array-based scientific data.

SciHadoop leverages on strategic physical location of scientific data to reduce the data transfer, remote reads process, and unnecessary reads. SciHadoop uses location

based optimization techniques such as planned partitioning of the input data, during the earlier phases of the MapReduce job for specific data types. This allows avoiding unnecessary block scans by examining data dependencies in an executing query.

This paper proposes an idea, which applies some optimizations to allow the scientists to use logical queries that are executed as MapReduce jobs over array-based data models. The main goal of SciHadoop is to perform these three goals: reduce total data transfer, reduce remote reads, and reduce unnecessary reads. This paper explains how MapReduce works by explaining the concept of array-based data models, which is indeed the structure of the data when it gets formulated after the mapping phase and before "shuffle and sort" phase. Then the combine function comes after that to produce the final result of one map function to be sent to the reducer. The reducer receives many final results from different mappers to calculate the final result of the whole job. The paper states that the storage devices these days are actually built to store the data format of byte stream data models.

Two points that make the scientific data work against the efficiency of using MapReduce are the high-level data model and the interface to meet some particular problem domain (n-dimensional format). The second point is that this data hides some details. Processing the scientific data will be done using a scientific access library, which means the mappers focus on (interacts with) data using a formulated data model and that partition occurs on the logical-level of the data model.

The Baseline Partitioning Strategy approach relies on how much the user knows about the information (knowledge) to construct the input partitions manually. First, the

approach formats the input data to be in one array, and second divides the logical input file to blocks. Each block is represented by a sub-array to make it easier to control. One of the drawbacks of this work is that formulating input data is done to meet the requirements of a single MapReduce job, so if there is another job to be done on the same file, we will have to reformat the data again.

The type of the input data might lead some researchers to develop a new algorithm or enhance the current one to make the accessing data process easier. One of the studies focuses on the binary data as source data for MapReduce algorithm that is read from HDFS. The authors in Bi-Hadoop [50] studied the degradation level in application performance. Bi-Hadoop develops an easy-to-use user interface, a caching subsystem in Hadoop and a binary-input aware task scheduler.

This study discusses the data locality for binary inputs, which shows to what extent the binary data is shared between multiple applications that use that data. The proposed problem is the distribution of that data and the overhead from data transfer that different applications produce when the tasks read the data. So, they proposed a developed solution to group the binary data to be close in the location of the tasks to reduce the overhead that was previously explained and assigns the tasks to the same compute node. Experiments show a 48% reduction in data read operations and up to 3.3x improvement in execution time than the native Hadoop.

A novel approach that is proposed for multiple sequence alignment using Hadoop [33] that proposes a time efficient approach for sequence alignment. It is discussed that

the dynamic feature in a sequence aligning in grid network using Hadoop. Due to scalability of Hadoop, this approach works perfectly in large-scale alignment problems.

Bidoop [25] discussed the benefits of applying Hadoop in bioinformatics data. It reports on its application to three relevant algorithms: BLAST, GSEA and GRAMMAR. The results show a very good performance using Hadoop due to some Hadoop features like scalability, computational efficiency and ease to maintenance.

Spatial data is one common data type that is used continuously. Authors in [61] proposed a very helpful solution by improving native Hadoop to a version that reads spatial data as two related numeric values for each location.

2.2.2 Improving performance Based on Data Size

Hadoop stores the metadata of the cluster on the NameNode that contains the blocks ID, location in the cluster, DataNodes, etc. Hadoop memory could run out and cause a slowdown in Hadoop performance when the metadata becomes large in size. Some studies provide solutions for this problem which is having a new archive system (NHAR) to improve the memory utilization for metadata and enhance the efficiency of accessing small files in HDFS [48] as shown in Figure 2.6. The experiments show that the access efficiency of small files in the new approach has been improved up to 85%.

Hadoop can perform a high degree of the performance processing of large size data files because the size of the blocks in HDFS usually is less that the source data size. However, if we have many small data files, Hadoop performance becomes less successful.

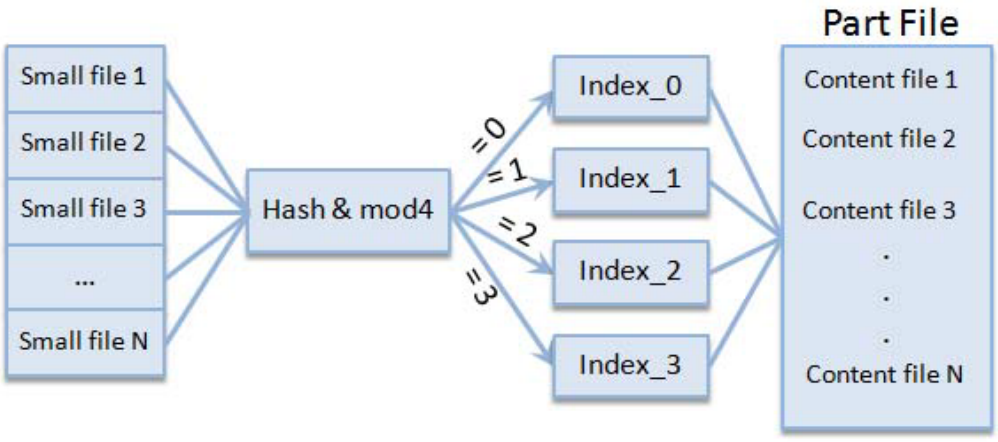


Figure 2.6: New Hadoop Archive technique that is presented in NHAR.

In [62] the WebGIS source data files are very small in size and cause the problem that we have above mentioned here before. So, this study proposes a solution that they can combine multiple files to be in one source file then this source file can be divided into many blocks and works well. WebGIS files have different characteristics to support easy access pattern. Figure 2.7 shows the Architecture of the proposed design in [62].

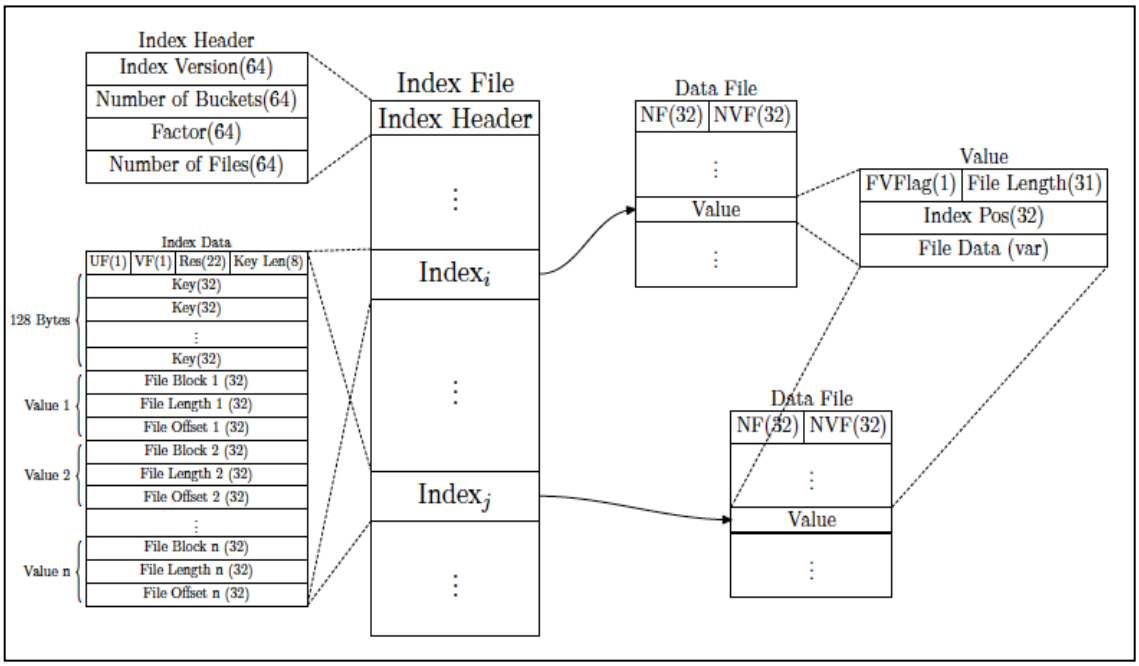


Figure 2.7: Architecture of combining multiple small files into one large file.

The WebGIS study also shows that there are some improvements in number of read operations. The number of read operations for blocks that meet the parameter of the size of data block following the proposed system is less than the number of read operations for the same data in native Hadoop. Figure 2.8 shows the difference between the number of read operations in the proposed solution and native Hadoop:

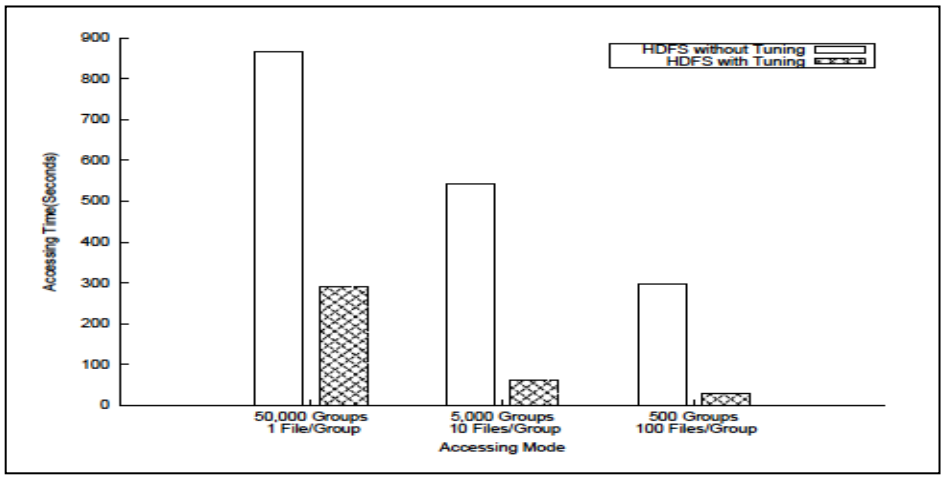


Figure 2.8: Number of Read Operations in Native Hadoop and Proposed Hadoop.

2.2.3 Improving performance Based on Data Location

Transferring data is one of the issues that paper [39] has discussed. While transferring data is considered as a cornerstone of the process, transferring un-necessary data can also be considered an important issue that we can study. By placing the tasks on the nodes that carry the data, we can develop the performance, which means it is all about the locality of the input data.

The authors integrated their work into FIFO, which is the default algorithm in Hadoop job scheduling and into Hadoop fair scheduling. Some comparisons have been

completed between the proposed techniques in this study and native Hadoop. The experiment's results show that map tasks have higher data locality rate and lower response time.

In [11], Hadoop data source locality is considered as one of the most important issue that is associated with the performance and costs of MapReduce. However, current Hadoop MapReduce workflow does not consider the locality of the data at requesting nodes during reducing the tasks. The Tasks Scheduler starts scheduling the tasks after a certain percentage, 5% by default, of mappers commit. That is when Hadoop starts early shuffling or scheduling the tasks on nodes. There is no consideration of data locality by JobTracker when early shuffling starts.

In this work, the authors propose a novel solution about the Locality-Aware Reduce Task Scheduler (LARTS) strategy. The idea behind LARTS is to defer the process of tasks scheduling until input data size is recognized. In comparison with native Hadoop, LARTS defers task scheduler by an average of 7% and up to 11.6%. To avoid scheduling delay, poor system utilization, and low degree of parallelism, LARTS employs a relaxation strategy and fragments some reduced tasks among several cluster nodes. LARTS improves node-local by 34.4%, rack-local by 0.32%, and off-rack traffic by 7.5%, on average, versus native Hadoop.

The discussion of locality issues is proposed in [47]. In this paper, the authors introduce a Hadoop MapReduce resource allocation system that enhances the performance of Hadoop MapReduce jobs by storing the datasets directly to the nodes that execute the MapReduce job. Thus there will not be any delay loading the data into the

nodes during the phase of copying data into HDFS Cloud. Optimum job mapper and reducers location is also discussed in this paper. It is also argued that the locality of the reducers is more important than the locality of the mappers.

Locality aware resource allocation reduces the network traffic generated in the cloud data center. Experimental results from a cluster of 20 nodes show reduction of job execution time by 50% and a decrease in the cross-rack network traffic by 70%.

The following charts in Figure 2.9 show the MapReduce input workload and compare random data placement and proposed algorithms in the paper [47].

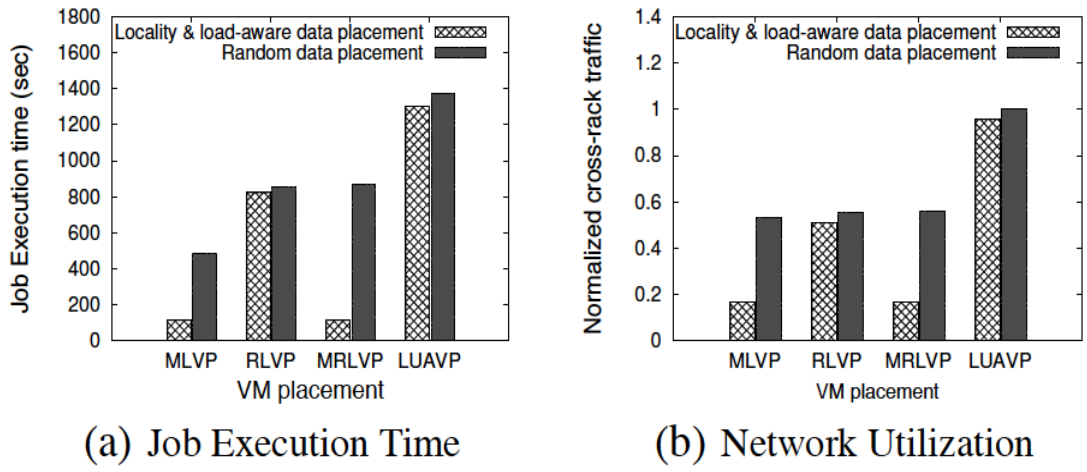


Figure 2.9: Map and Reduce-input heavy workload.

They compared different techniques, which are as follows:

- Locality-unaware VM Placement (LUAVP).
- Map-locality aware VM placement (MLVP).
- Reduce-locality aware VM placement (RLVP).
- Map and Reduce-locality aware VM placement (MRLVP).

Another study has discussed the same point, which is reduce task location [51] and discussed the same idea about the degradation that might be caused because of fetching the intermediate data after map tasks. The proposed solution is about reducing the intermediate results fetching cost in MapReduce jobs by assigning the reduce tasks to the nodes that do the intermediate results and pick the low fetching cost. This study follows the optimum placement strategy for the threshold-base structure.

The experiments show that the proposed solution improves the performance of Hadoop MapReduce. Figure 2.10 shows one results:

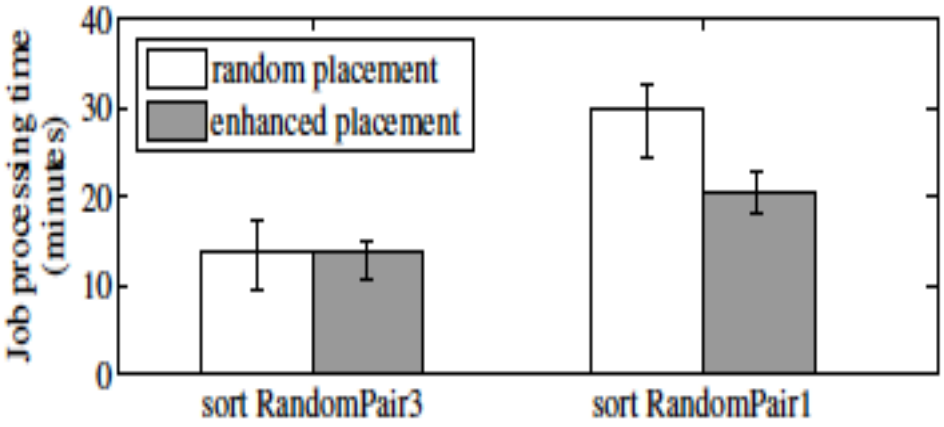


Figure 2.10: Mean and range of the job processing times by repeating 5 times.

In Hadoop, the distribution storage and datacenters located happens geographically in a Cloud Computing environment. In [52] Single-datacenter or multi-datacenters have been experimented in the same jobs to evaluate the location of the data in Cloud and to evaluate the system performance. Using a prediction-based system to predict the job location in this study reduces the error rate to be less than 5% and develops the execution time of Reduce phase by 48%.

Results in Figure 2.11 show the efficiency of the prediction in the Map phase comparing with the real execution time (a) and the time is very close. In (b) the Reduce phase execution time with optimizations is less than the native one that is without the optimizations [52].

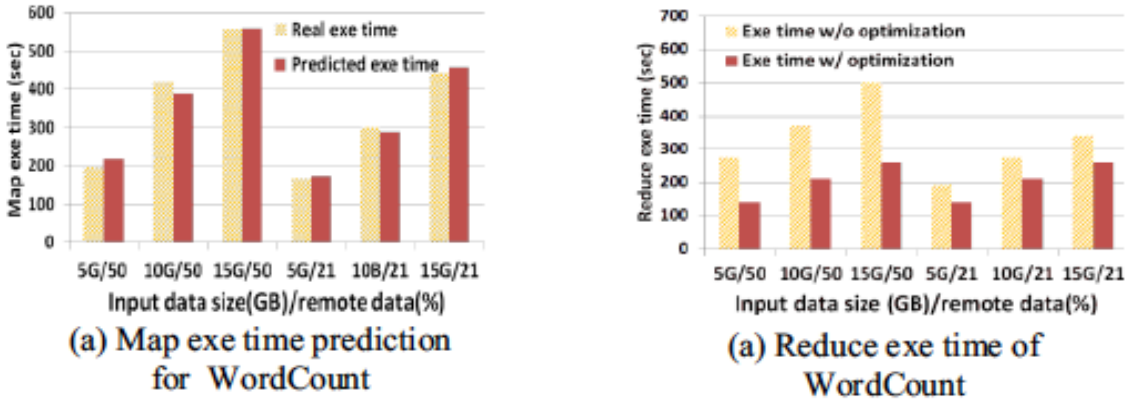


Figure 2.11: Map and Reduce execution time of WordCount example.

Table 2.2 shows a summary of comparisons between studies related to the improvement of Hadoop performance via developing some data considerations in Cloud Computing, such as data locality and data size.

Table 2.2: Improvement in Data Considerations in Cloud Computing

Study	*Problem Definition*	*Improvements*	*Results*
J. Buck and others " SciHadoop"	Formatting the scientific data to be in Array-based binary format	• Reduce transferred data • Reduce remote reads	Reduce the data transferred by 20% before submitting jobs
Y. Xiao and others "Bi-Hadoop"	Control distributing data and the overhead of data transfer to different nodes	Group the binary data to be close in the location of the tasks to reduce the overhead	48% reduction in data read operations Up to 3.3x improvement in execution time
S. Leo and others "Bidoop"	Applying Hadoop in bioinformatics data	Formatting Data to be in a simple format to be processed using Hadoop	Good results on BLAST, GSEA and GRAMMAR
A. Eldawy "Spatialhadoop"	Applying Hadoop in spatial data such as GIS location x and y axis	Read two related numeric Data to be processed using Hadoop	Good results since the data is organized and formatted earlier
M. Hammoud and others " LARTS"	Delay the process of tasks scheduling until input data size is recognized	Defers task scheduler by an average of 7% and up to 11.6%	Improves node-local by 34.4% Rack-local by 0.32% Off-rack traffic by 7.5% on average
B. Palanisamy and others "-ieus"	Introduce a resource allocation system to enhance the performance of Hadoop jobs	Storing the datasets directly to the nodes that execute the MapReduce job	Reduce job execution time by 50% Decrease the cross-rack network traffic by 70%
Vorapongkitipun and others	Small file accessing in Hadoop	Improve the memory utilization for metadata and enhance the efficiency of accessing small files in HDFS	Access efficiency of small files in the new approach has been improved up to 85%
L. Xuhui and others	WebGIS source data files are very small in size	Combine multiple files to be in one source file then this source file can be divided into many blocks	Improvements in number of read operations

2.3 Optimizing Cloud Computing Environment

Location of the data in Hadoop can perform big change in the performance. One of the studies discussed the reusing of the intermediate result of MapReduce between jobs [55]. This approach discussed the relation between jobs in MapReduce and the relationship between them to make them reuse the output of the intermediate related job. They discussed the jobs that are created by one of the ecosystem tools of Hadoop like Pig or Hive.

These tools allow users to write queries similar to sql queries then the tools' compilers convert the queries to MapReduce jobs in java. These jobs are related, so some of them will not start until the previous ones finish. Usually, after job finishes its work, the intermediate results get deleted. The proposed solution here is about reusing these data for future work.

This study provides Restore system to store the intermediate data. It implements the Restore on top of Hadoop to improve the performance by speed up the execution time of the process. This approach provides benefits to the same jobs that have the same parameters to not be executed again and again. So, if we have different parameters, this approach does not support the improving the performance of Hadoop.

The authors in [53] introduce one important point, which is the role of Hadoop in some applications like web indexing, data mining and some simulations. Hadoop is considered as an open-source implementation that follows the parallel processing techniques to perform the processes; it is also used frequently for short jobs to have better

response time. One of the implicit assumptions of Hadoop is that the cluster is homogeneous which means the all nodes are similar in process, which affects the task scheduling process negatively.

Based on that assumption, Hadoop assigns the tasks linearly and decides when to re-execute the struggled tasks within the jobs. Figure 2.12 explains the three scenarios (Worst, Best and Average) of LATE comparing with native Hadoop:

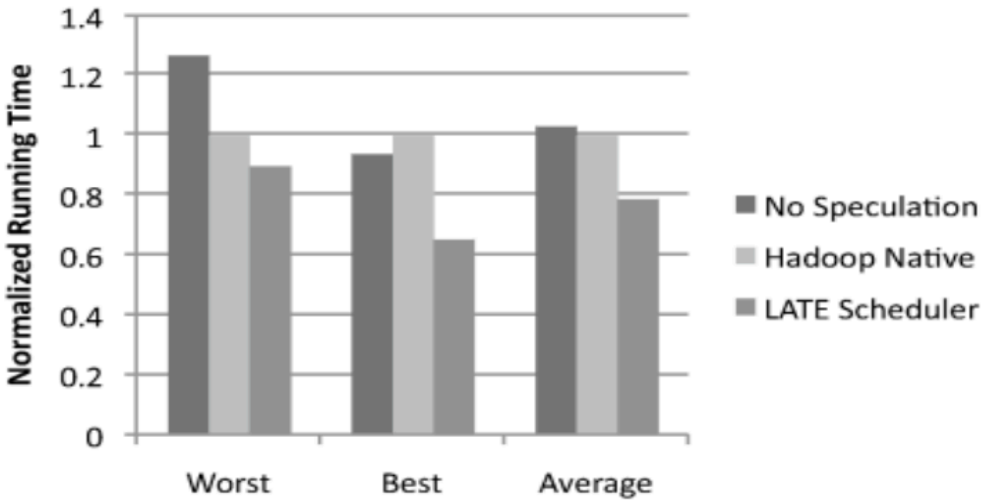


Figure 2.12: EC2 Sort running times in heterogeneous cluster: Worst, best and average-case performance of LATE against Hadoop's scheduler and no speculation.

Homogeneity assumption is not always the perfect way to assign tasks because of the nature of the clusters in reality. In this study, it shows that in a heterogeneous environment there might be a severe degradation in performance because of the homogeneity assumption. They developed a new algorithm for scheduling purposes named Longest Approximate Time to End (LATE), and they described it as it has a high robust to heterogeneity.

We can summarize the working of LATE as it determines the tasks that cause low performance in response time and execute these tasks first. They tested LATE in Amazon EC2 and the result showed a better performance then the native Hadoop.

The authors in [54] present a data placement strategy to improve Hadoop MapReduce performance in an homogeneous environment. The current Hadoop assumes that the nodes in the cluster are homogeneous in nature, and the locality of the data is also assumed as a data-local. Unfortunately, this assumption is not the case in virtualized data centers. So, heterogeneous environment can reduce the MapReduce performance.

The problem of load balance process has been addressed in [54] to give a good data load balance process in a heterogeneous environment by using data-intensive application running on the cluster. So, the results showed that the data placement process could improve the MapReduce performance by re-balancing data within the nodes before launching the data-intensive application. The dynamic heterogeneity for resource allocation is also discussed in [56].

Another study shows the importance of having the accessing process to the distributed data to be accessed from multiple-source streaming. Some of the DataNodes are located either in-rack or off-rack network topology, which may cause delay in accessing performance [57]. The study proposed a solution for that issue by having a circular buffer slice reader that enables data to be accessed by multiple tasks at the same time to be less static topology.

Another issue that is related to the data locality in the nodes has been discovered in [58] which is about the ability to determine the related data to the jobs that are located in the same node. The study identified that as a bottleneck of the workflow of MapReduce. CoHadoop is a proposed solution that gives the ability to the applications to control the location of the data to be stored in specific locations. CoHadoop gives some hints to the applications about the data that are related to the job, thus CoHadoop can do some preparation on the data, such as some joining operations. Then CoHadoop tries to collocate these file in some locations based on its choice.

CoHadoop can apply different operations such as aggregations, joining, indexing or grouping. The experiments show that the proposed solution provides a better collocation for some applications specially, which only have map tasks. CoHadoop++ is another study that adds some achievement to CoHadoop by selecting the nodes not randomly, so it provides load balance [59].

Improving Hadoop cloud architecture – named Enhanced Hadoop in a different published paper - also discusses one approach to improve Hadoop performance by following and catching the metadata of the related executed jobs on the same data sets [63]. Enhanced Hadoop reduced the size of read data by eliminating the unrelated data between the related jobs. It produced a very high improvement in Hadoop performance. Enhanced Hadoop focuses on the real text data type.

There are different studies that are being developed to improve the performance of Hadoop in order to reduce processing costs, reducing runtime, or increasing the efficiency. However, most of these solutions solve specific problems or cases under some

circumstances, so some companies still use the native Hadoop in their work. In addition, many companies and developers use NoSQL databases to administrate their work as a good choice alternative to Hadoop.

Table 2.3 shows a summary of comparisons between studies related to the improvement of Hadoop performance via developing some factors related to Cloud Computing environments such as homogeneity of the nodes in the cluster.

Table 2.3: Improvement in Cloud Computing Environment

Study	***Problem Definition***	***Improvements***	***Results***
I. Elghandour and others "ReStore"	Reusing intermediate result of MapReduce between jobs	Provides restore system to store the intermediate data It implements the restore on top of Hadoop to improve the performance by speeding up the execution time of the process	Provides benefits to the jobs that have the same parameters to not be executed again and again If we have different parameters, this approach does not support the performance
M. Zaharia and others	Degradation in Hadoop performance because of the homogeneity assumption	Developed a new algorithm for scheduling purposes named Longest Approximate Time to End (LATE) Determines the tasks that cause low performance in response time and execute these tasks first	Tested LATE in Amazon EC2 and the result showed a better performance then the native Hadoop
J. Xie and others	Heterogeneous environment can reduce the MapReduce performance	Have a good data load balance process in a heterogeneous environment by having data-intensive application running on the cluster	Improve the MapReduce performance by re-balancing data within the nodes before launching the data-intensive application
M. Eltabakh and others "CoHadoop"	Determine the related data to the jobs that are located in the same node	CoHadoop can do some preparation on the data based on some hints to the applications about the data that are related to the job	Provides better collocation for some applications specially which only have map tasks

2.4 Drawback of some solutions

Although many of the proposed solutions have been discussed and implemented perfectly either by using specific data or by applying them under some conditions, some of these solutions are not applicable as expected when they are being deployed in a real network such as Hadoop Cloud [64]. When applying such solutions, the runtime analysis and debugging of that process is not easy to be addressed and monitored by using the traditional approaches and techniques. The authors in [64] have discussed that issue and they provided a lightweight approach to cover the difference between pseudo and large-scale Cloud improvements. A couple of solutions are implemented to address specific problems especially those that are related to addressing the input data issue.

CHAPTER 3: RESEARCH PLAN

Based on the previous studies, improving Hadoop MapReduce performance can develop many issues like reducing the processing time of a job. In my research, I will focus on improving the MapReduce performance by enhancing the native Hadoop architecture. One of the important levels that research can improve is the locality of data.

Many studies have improved the reading data process for mapping tasks by controlling the locality of the data within the cloud architecture either physically or logically as shown in literature survey section. However, some of these studies have some limitations and drawbacks as I discussed in chapter 2.

In my work, I represent a new Hadoop enhanced architecture that improves the reading data processes in mapping tasks, which reduces all stages that are related to the size of the data. For example, based on the data size, the default block size is 64Mb. We can calculate the number of all blocks that the system reads, so all related stages will work based on that number. Consequently, if we can reduce the number of blocks that the system read, we can improve Hadoop MapReduce performance.

3.1 Overview of Native Hadoop Architecture

In different studies, users focus on improving Hadoop performance and evaluate that by comparing their proposed solutions with the current one. In this section, we will discuss the native Hadoop workflow and its limitations in terms of the MapReduce algorithm performance. After that, we will propose our enhanced Hadoop MapReduce

workflow and compare the two architectures in terms of the developing MapReduce performance.

3.1.1 Native Hadoop MapReduce Workflow

In current Hadoop MapReduce architecture, the client first sends a job to the cluster administrator, which is the NameNode or the master of the cluster. Job can be sent either using Hadoop ecosystem (Query language such as Hive) or by writing Java code [65]. Before that, the data source files should be uploaded to the Hadoop Distributed File System by dividing the BigData into blocks that have the same size of data, usually 64 or 128 MB for each block. Then, these blocks are distributed among different Data Nodes within the cluster. Any job now has to have the name of the data file in HDFS, the source file of MapReduce code (e.g. Java file), and the name of the file that the result will be stored in also in HDFS.

Current Hadoop architecture follows the concept of "write-once and read-many", so there is no ability to do any changes in the data source files in HDFS. Each job has the ability to access the data from all blocks. Therefore network bandwidth and latency is not a limitation in dedicated cloud, where data is written once and read many times. Many iterative computations utilize the architecture efficiently as the computations need to pass over the same data many times. Several research groups have also presented locality aware solutions to address the issue of latency while reading data from DataNodes.

Hadoop falls short of query optimization and reliability of conventional database systems. In the existing Hadoop MapReduce architecture, multiple jobs with the same data set work completely independent of each other. We also noticed that searching for

the same sequence requires the same amount of time each time we execute the job. Also, searching for the sub-sequence of a sequence that has already been searched requires the same amount of time.

MapReduce workflow in native Hadoop has been explained in Figure 3.1 as follows:

Step 1: Client " A" sends a request to NameNode. The request includes the need to copy the data files to DataNodes.

Step 2: NameNode replays with the IP address of DataNodes. In the above diagram NameNode replies with the IP address of five nodes (DN1 to DN5).

Step 3: Client " A" accesses the raw data for manipulation in Hadoop.

Step 4: Client "A" formats the raw data into HDFS format and divides blocks based on the data size. In the above example the blocks B1to B4 are distributed among the DataNodes.

Step 5: Client "A" sends the three copies of each data block to different DataNodes.

Step 6: In this step, client "A" sends a MapReduce job (job1) to the JobTracker daemon with the source data file name(s).

Step 7: JobTracker sends the tasks to all TaskTrackers holding the blocks of the data.

Step 8: Each TaskTracker executes a specific task on each block and sends the results back to the JobTracker.

Step 9: JobTracker sends the final result to Client "A". If client "A" has another job that requires the same datasets it repeats the set 6-8.

Step10: In native Hadoop client “B” with a new MapReduce (job2) will go through step 1-5 even if the datasets are already available in HDFS. However, if client “B” knows that the data is exist in HDFS, he will sends job2 directly to JobTracker.

Step 11: JobTracker sends job2 to all TaskTrackers.

Step12: TaskTrackers execute the tasks and send the results back to the JobTracker.

Step 13: JobTracker sends the final result to Client “B”.

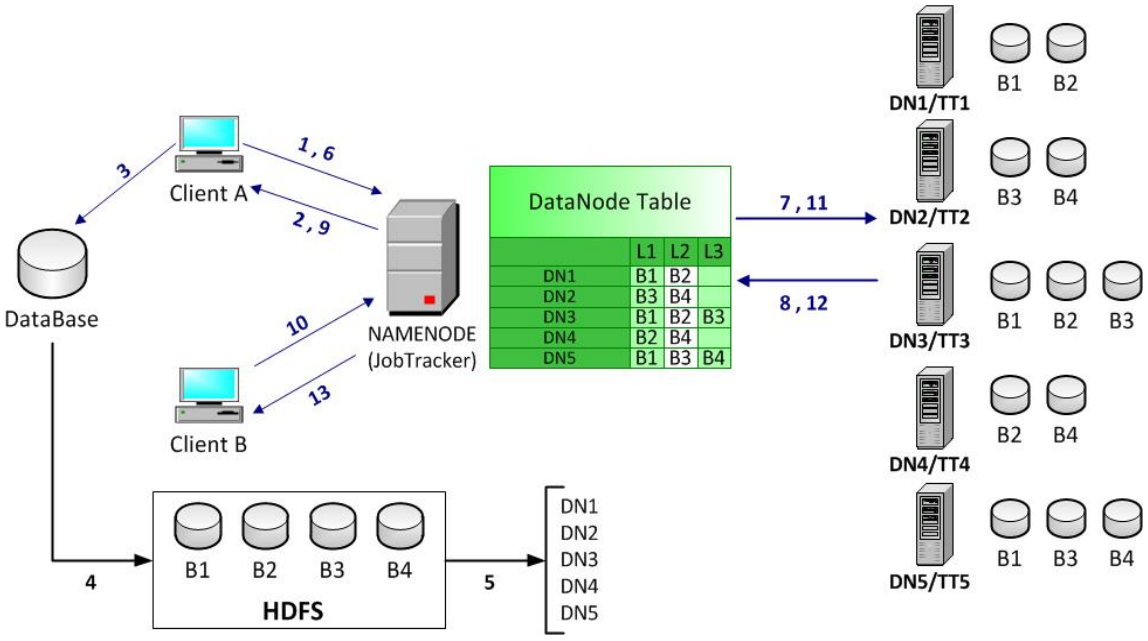


Figure 3.1: Native Hadoop MapReduce Architecture

Figure 3.2 shows the workflow chart for Native Hadoop. We can see that there is independency between jobs because there are no conditions that test the relationship between jobs in Native Hadoop. So, every job deals with the same data every time it gets processed. In addition, if we have the same job executed more than one time; it reads all the data every time, which can cause weakness in Hadoop performance.

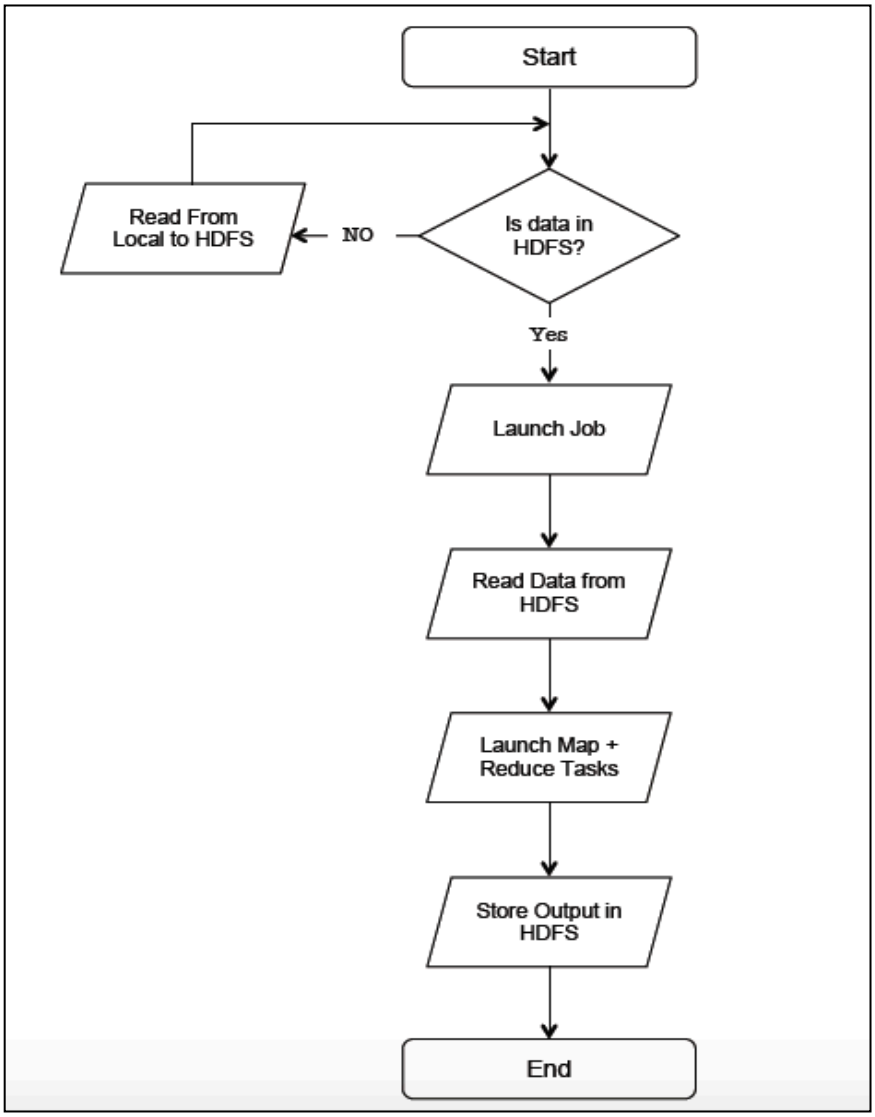


Figure 3.2 Native Hadoop MapReduce Workflow Flowchart

3.1.2 Overview of native Hadoop Performance

Native Hadoop compiler processes MapReduce job by dividing the job into multiple tasks, then distributes these tasks to multiple nodes in the cluster. By studying Hadoop performance in [66] the authors discussed Hadoop MapReduce model to estimate MapReduce job cost by giving some parameters to the model.

Different parameters that jobs need to have to be executed efficiently. These parameters are:

1) *Hadoop Parameters*: which is a set of predefined configuration parameters that are in Hadoop setting files.
2) *Profile Statistics*: which are a set of user-defined properties of input data and functions like Map, Reduce, or Combine.
3) *Profile Cost Factor*: which are I/O, CPU, and Network cost job execution parameters.

We will focus on the third category of parameters, which is the Profile Cost Factor. In this section we are going to explain the job execution cost in details. We will further explain the relationship between the number of blocks and the cost associated with the reading of the data from HDFS.

$$NumberOfBlocks = DataSize / BlockSize \quad (3.1)$$

Where *DataSize* is the size of the raw data that we want to upload to HDFS, and *BlockSize* is the pre-defined size for data block (by default it is 64MB). There is a compression ratio that is applied to each block to have it less in size before it is stored in the HDFS. We will not discuss the compression ratio point here because it is not one of our concerns and it has been discussed clearly in [66].

MapReduce job reads data from HDFS where the cost of reading a single data block from the HDFS is *HdfsReadCost.* The cost of reading the whole data from HDFS is *IOCostRead* and it is calculated as:

$$IOCostRead = NumberOfBloks \text{ X } HdfsReadCost \quad (3.2)$$

Cost of writing a single data block to HDFS is *HdfsWriteCost*. The cost of writing any data, such as MapReduce job results or raw data, is *IOCostWrite* and is calculated as follows:

$$IOCostWrite = NumberOfBloks \text{ X } HdfsWriteCost \quad (3.3)$$

From the above equations we clearly see that the total costs of reading and writing from HDFS depends on the number of blocks, which is the data size. So, by reducing the data size, we can reduce the costs of these processes, which will lead to improving the Hadoop's performance.

In addition, it is true for every Hadoop's process that the number of blocks is related to its costs. For example, the CPU cost of reading is *CPUCostRead* and is calculated as follows:

$$CPUCostRead = NumberOfBlocks \text{ X } InUncompeCPUCost + InputMapPairs \text{ X } MapCPUCost \quad (3.4)$$

Where *InUncompeCPUCost* is the compression ratio of blocks, *InputMapPairs* is the number of pairs for mapping process, and *MapCPUCost* is the cost of mapping one pair.

3.1.3 Hadoop MapReduce Limitations

Many Hadoop MapReduce jobs, especially tasks associated with the science data such as genomic data, deal with the similarities in sequences, superstring and sub-sequences searches [28]. Such tasks usually require multiple MapReduce Jobs to access the same data many times. For a DNA sequence matching task, if an n-nucleotide long sub-sequence exists in a specific DataNode than it's a superstring-sequence, sequence containing the sub-sequence, can only be found in the same DataNodes. Since current Hadoop Framework does not support storing metadata of previous jobs, it ignores location of DataNodes with sub-sequence and reads data from all DataNodes for every new job.

As shown in Figure 3.1, let's suppose that Client A and Client B are searching for the same sequence in BigData source file(s). Once client A finds the sequence, client B will also go through the same steps again to find the same results. Since each job is independent, clients do not share results and process redundancy remains a major unsolved problem in native Hadoop MapReduce infrastructure.

3.2 Proposed Enhanced Hadoop Architecture

In existing Hadoop architecture; NameNode knows the location of the data blocks in HDFS. NameNode is also responsible for assigning jobs from the client and divides it into tasks and assigns these tasks to the TaskTrakers (DataNodes). Knowing which DataNode holds the blocks containing the required data; NameNode should be able to direct the jobs to a specific DataNodes without going through the whole cluster. In this

study we propose a pre-processing phase in the NameNode before assigning tasks to a DataNodes.

We focus on identifying and extracting features and building a metadata table that carries information related to the location of the data blocks with these features. Any job with the same features should only read the data from these specific blocks on the cluster without going through the whole data again [67]. Explanation of the proposed solution is as follows:

3.2.1 Common Job Blocks Table (CJBT)

Proposed Hadoop MapReduce architecture is same as the original Hadoop in terms of hardware, network, and nodes. However, the software level has been enhanced. We added features in NameNode that allow it to save specific data in a look up table named Common Job Blocks Table (CJBT).

CJBT stores information about the jobs and the blocks associated with a specific data and features. This enables the related jobs to get the results from those blocks without checking the entire cluster. CJBT is related to only one HDFS data file, which means that there is only one table for each data source file(s) in HDFS. In this study, we take an example of genome BigData to show the functionality of enhanced Hadoop architecture.

Sequence aligning is an essential step for many molecular biology and bioinformatics applications such as phylogenetic tree construction, gene finding, gene function and protein structure prediction [24]. Computationally intensive algorithms are

used for sequence alignment. Scalable parallel processing Hadoop framework has been proposed for the sequence alignment of genomic data [15, 26, 68, 69].

Proposed Hadoop architecture relies on CJBT for efficient data analysis. Each time a sequence is aligned using dynamic programming and conventional alignment algorithms, a common feature that is, a sequence or a sub-sequence is identified and updated in CJBT. Common feature in CJBT can be compared and updated each time clients submit a new job to Hadoop. Consequently, the size of this table should be controlled and limited to a specific size to keep the architecture reliable and efficient. A typical CJBT consists of three main components or columns (Table 3.1), which are explained below:

Table 3.1: Common Job Blocks Table components

Common Job Name	Common Feature	Block Name		
Sequence_Alignment	GGGATTTAG	B1	B2	B3
	TTTAGA	B1	B4	
Fining_Sequence	TTTAGCC	B3	B6	
	GCCATTAA	B1	B3	B4
	AATCCAGG	B3	B5	

Common Job Name represents a shared name of a job that each MapReduce client must use while submitting a new job in order to get the benefit of the proposed architecture. We defined a library containing a list of pre-coded jobs is made available to the user by an Application Program Interface (API). The Jobs APIs provide a brief job description and access to job data. The users select a job name (or shared database name)

from the list of jobs already identified for a shared MapReduce job (or data). This feature helps NameNode identify and match a job to a DataNode(s) containing block(s) in the CJBT.

Common Features are defined as the shared data between jobs. Proposed enhanced Hadoop architecture supports caching and enables output (or part of output) to be written in the CJBT during the reduce step. We use Common Features to identify the DataNodes or the blocks with shared data entries. TaskTracker directs any new jobs with the shared common features to block names in CJBT. Suppose J1 and J2 are sequence search jobs, J1 uses MapReduce to find the sequence in a DataNode or a block. If J2 contains the common feature (sub-sequence) of J1, it is logical to map the task and allocate same data resources of J1.

When a sub-sequence arrives to the NameNode as the result of a new job, the old common feature will be replaced with the shortest one. However, feature selection should be done carefully as the response time for the jobs can increase if the common features exist in every DataNode. For example, in genomic data, regulatory sequences and protein binding sites are highly recurring sequences. Using such sequences as common features can degrade the performance of the proposed architecture.

Lengths of common features also play in important role in the proposed architecture. If the sequence is too short it would be present many times in all chromosomes and all datasets. For a random sequence Dn is the likelihood of how many times a DNA sequence occurs in the whole human genome. The likelihood of the binding sites for 9, 12 and 15 fingers ZNF is presented in Table 3.2.

For a random sequence of length Dn, where n is the length of nucleotide sequence, the likelihood of how many times a sequence occurs in the whole human genome is given by:

$$D_n = 3 \times 10^9 / (4)^n \quad (3.5)$$

Table 3.2: Likelihood of Random Nucleotides

# of Nucleotides	likelihood of finding any random 9 – 15 nucleotides sequence in the human genome: $D_{(n)}$
genome	3×10^9
09 -nucleotides	$D_9 = 11444$
12 -nucleotides	$D_{12} = 178$
15 -nucleotides	$D_{15} = 2.7$

As shown in Table 3.2, the likelihood of any random 9 base pair (bp) of a long nucleotides sequence in a whole genome is quite large, and using a 9 bp long sequence as a common feature will result in the performance degradation of the proposed architecture. The probability of any random 12 bp long sequence in a human genome is 5.96 x 10-8 equaling 178 times.

BlockName or BlockID is the location of the common features. It identifies the block(s) in a cluster where certain information is stored. BlockName helps NameNode to direct the job to specific DataNodes that store these blocks in HDFS. CJBT has the list of all blocks that are related to the results of the common feature. For example, if a sequence "TTTAGATCTAAAT" is only stored in B1 and B4, NameNode will direct any job that

has this particular sub-sequence to B1 and B4. This CJBT is a dynamically configurable table and the BlockName entries are changing as the common feature change.

3.2.2 End-User Interface

A user interface gives the user a list of Common Job Name (CJN) to choose from. As the tasks are completed, CJBT is dynamically updated and more relationships are defined. If the common job block table is empty, the user will execute the MapReduce job in a traditional way without getting the benefits of the proposed architecture. The predefined CJN and CF are defined either by the user himself or by the user interface manager, which might become a central source for updating the lists for all clients.

Common Job Blocks Table should not become too large because larger lookup table tends to decrease the system perforce. The size of CJBT can be limited by employing 'leaky bucket' algorithm [70]. The 'leaky bucket' parameters can be adjusted to keep the size of CJBT constant. This can be discussed more in future work.

3.2.3 Enhanced Hadoop MapReduce Workflow

Enhanced Hadoop architecture doesn’t have any hardware changes different than native Hadoop architecture, so there will be enhancing only in the software level like CJBT. Following chart in Figure 3.3 shows the proposed changing in NameNode that works as a lookup table contains metadata for the executed jobs in Hadoop.

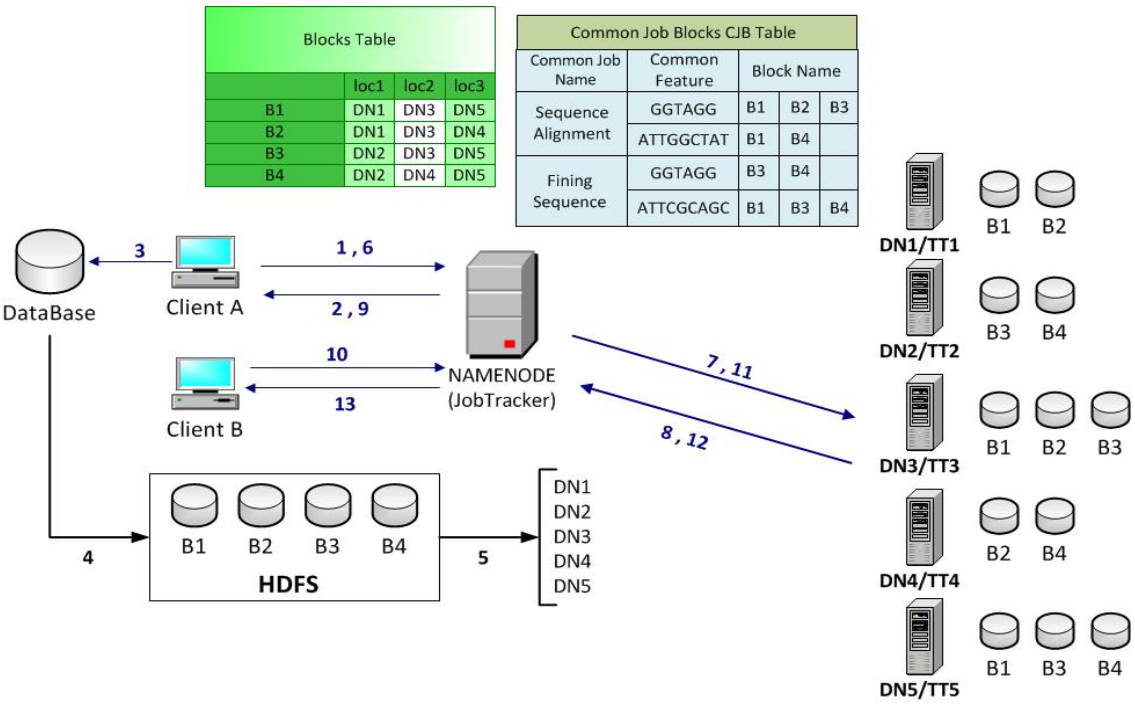


Figure 3.3: Enhanced Hadoop MapReduce Architecture

MapReduce workflow in enhanced Hadoop has been explained in Figure 3.3 as follows:

Step 1 to Step 8 have the same workflow as native Hadoop. Results from the first 7 steps are stored in a CJBT.

Step 9: JobTracker sends the result to Client "A". In this step, NameNode keeps names of the blocks that produced the result in a local lookup table (CJBT) by the Common Job Name (Job1) that has common feature as explained earlier.

Step 10: Client "B" sends a new MapReduce job "job2" to the JobTracker with the same common job name and same common feature or super-sequence of it.

Step 11: JobTracker sends "job2" to TaskTrackers who hold the blocks that have the first result of MapReduce "job1" (DN2, DN4, DN5). In this step, JobTracker starts

with checking the CJBT first to find is the new job has the same common name and common features of any previous ones or not – In this case yes. Then JobTracker sends "Job2" only to TT2, TT4 and TT5. We may assume here the lookup table will be updated with more details OR just keep it as it because every time we have a new job that will carry the same name of "Job1".

Step 12: TaskTrackers execute the tasks and send the results back to the JobTracker.

Step 13: JobTracker sends the final result to Client "B".

The workflow that is shown above explains the normal flow steps of the enhanced Hadoop MapReduce. In addition, there should be a training phase before starting the process of MapReduce to have some metadata in the CJBT to get the benefits of the new architecture.

In Figure 3.4, after launching a job there is a condition that tests the name of the job. If the job uses a CJN, which means this job is commonly used and there might be a relationship between this job and others. Otherwise, if the name of the job is not common, it skips the second condition and reads the whole data from the HDFS and completes the execution.

If the name of the job is common, which means the first condition is "Yes", it will check the second condition, which tests the common feature of the job. If the feature of the new job is common with any previous job, the new job reads the specific data blocks from the HDFS and sets them as source data files, not the whole data block. Then the new job will be executed normally.

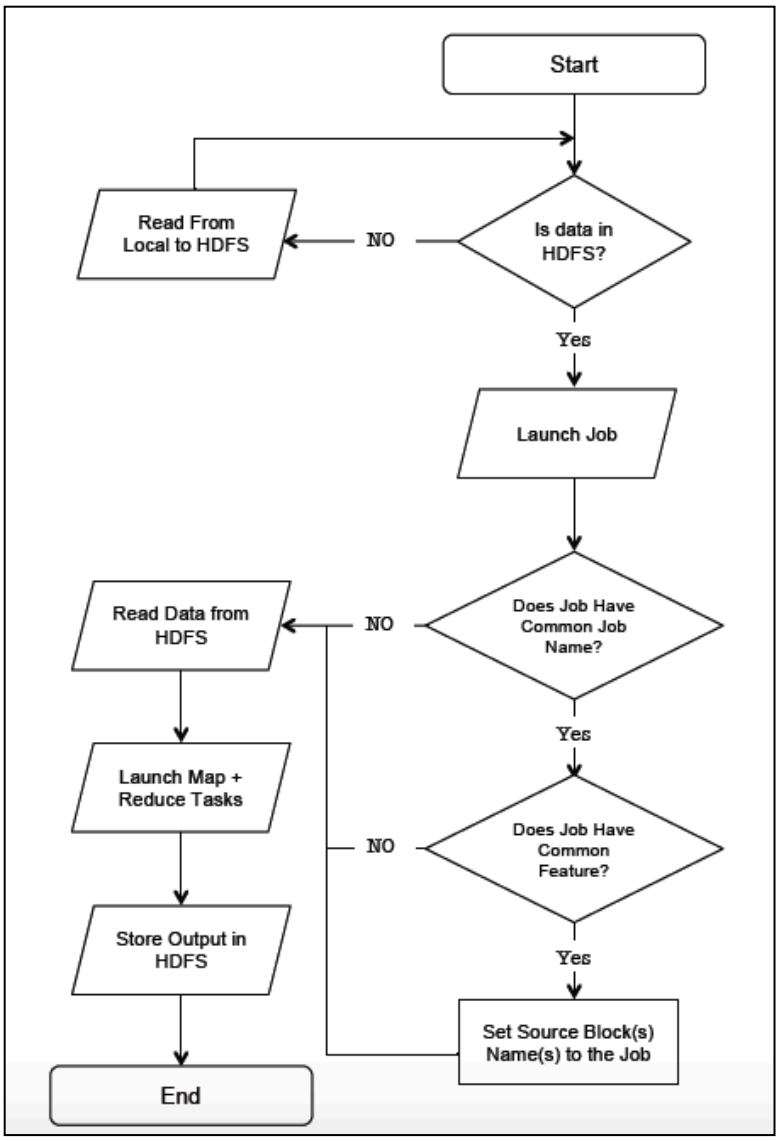


Figure 3.4: Enhanced Hadoop MapReduce Workflow Flowchart

Under these two conditions, the Enhaned Hadoop reduces the size of the data that is being read by the new job. Consequently, this improves on the Hadoop performance for jobs that are working on similar data files.

CHAPTER 4: IMPLEMENTATION AND TEST PLAN

In this chapter, we will discuss the implementation plan for the proposed solution and expected results. Under specific circumstances, proposed solution cloud be implemented in two different ways. These circumstances are number of data files and the size of each file. We will discuss the two concepts as follows:

First: In cases where there are many source data files and each one is less than the default value of the block size, we prefer to implement the proposed solution out of map and reduce phases in NameNode. Map and Reduce phases in NameNode is used in this particular stance this approach enables us to read the source files separately from HDFS and before creating the blocks, gives us less data size because each file size is less than the block size.

Second: In cases where there is a one or couple of data source files and most of the files are larger than the default block in size, we prefer to implement the proposed solution to read the data after map and reduce are launched in Hadoop because the block size is less than the source file size.

In our implementation, we follow the first proposed case because we have the DNA chromosomes data and this data is about 24 files. Each file is less than the default block size in Hadoop. A couple of jobs can be implemented using this data. The implementation of the proposed solution goes in three parts:

4.1 Creating the Common Job Block Table (CJBT)

Using different techniques can perform designing and creating the CJBT. One of these is using HBase as a NoSQL database. HBase is a column-oriented database of which a main property is that it expands horizontally [71].

There is a reason for using HBase, which is that HBase is Apache open source software that is one of NoSQL databases that works on top of Hadoop. We use HBase as an indexing table here to complete our job and enable the proposed solution work perfectly.

4.2 Designing User Interface (UI)

As we proposed before, the user interface should contain some important points that give the user an ability to get the benefits of the enhanced design by choosing some common data from lists. For example, choosing the Common Job Name from a list of common job names that are related to the same data files.

Different forms of user interfaces can be designed based on the user needs. One of the common user interfaces is the command line that is commonly used when the user knows the commands and the related parameters. Hadoop and HBase are controlled by the same command line, which is a shell command line in Linux. Therefore, in our work, we use the shell command line as a user interface to implement the proposed solution.

4.3 Proposed Solution Environment

We can implement the proposed solution following some directions in [72] to prepare the cluster first then we can do the modifications on the environment. In addition,

since we have Hadoop and HBase both run on the shell interface on Linux, we use it for the implementation of the proposed solution. We use the following applications and tools:

- Linux OpenSUSE as an operating system on all nodes in the cluster. We use both versions of OpenSUSE11.1 and OpenSUSE12.3. We can use different versions at the same time with no conflicts between the nodes.
- Apache Hadoop1.2.1, which is the stable version of Hadoop at the time of implementing the cluster.
- Apache HBase0.98, which is the stable version of HBase at the time of implementing the cluster.

4.4 Execute some experiments on the enhanced Hadoop

Having common features exist in all files is not a common case, but it does happen. In DNA chromosomes, there are a couple of sequences that are common for searching protein process. The following examples are some sequences and their locations Table 4.1 (store the ChromosomeName in which chromosomes they occur):

Table 4.1: Common Job Block Table (DNA example)

	Common Feature (Sequence)	Block Name/ID (Chromosome Name)
sq1	GGGGCGGGG	In All Chromosomes
sq2	AAGACGGTGGTAAGG	1, 8
sq3	CATTTCTGCTAAGA	1,2,3,4,6,7,9,10,11,12,13,18,19,21
sq4	GAATGTCCTTTCTCT	1,3,6,7,9,17,19,20,21
sq5	GATCTCAGCCAGTGTGAAA	3,7,16

We launched many experiments on different text file formats to test the sequence finding job with different common features. One of the experiments is finding a sequence of DNA data files. We stored the common job block table as shown in Table 4.1 using HBase for easy access in the Enhanced Hadoop environment.

4.5 Amazon Elastic MapReduce EMR experiments

Amazon Web Services AWS is considered as one of the most common web services provider. It provides distributed storage service S3 that stores data in different nodes in the Cloud. Also, AWS provides a distributed processing service, which is Elastic Map Reduce EMR service that processes the MapReduce jobs in the Cloud environment. S3 and EMR work together to provide a good solution for Hadoop and BigData analysis applications for users who don't have the ability to create their own Cloud cluster. Users should consider the network bandwidth when they use AWS.

Amazon EMR service provides a distributing process for BigData. Users have to create a cluster in the AWS and complete some configurations, such as the number of nodes, hardware capability of the nodes, and a couple of directories to retrieve and store data etc. The Amazon EMR provides different versions of Hadoop to be selected by the user during the creation of the cluster.

Using Amazon EMR makes the process of analyzing the data much easier than creating your own cluster in terms of hardware and settings. However, it costs more time because of the distribution of data between nodes that are located in different racks on the network. In the results section, we will go over this point and discuss the reasons of having this cost by comparing with the Enhanced Hadoop.

CHAPTER 5: RESULTS

Up to this point, there are indications that we got good results comparing with the native Hadoop MapReduce environment. By implementing the proposed solution, we have less data size to be read by the related jobs. Reducing the number of reads has direct effect on the performance of Hadoop [73]. As expected, we also noticed that performance of Hadoop MapReduce depends upon the length of common feature and the likelihood of finding the common features in the source files and DataNodes. If the common features exist in all source files, the enhanced Hadoop MapReduce will not improve the performance as the job reads the all files that contain the common feature.

From Table 4.1, sequence1 is located in all chromosomes, which means it is located in all data blocks. So, Enhanced Hadoop reads the whole data sets again if the common feature is sequence1. In this case it gives no benefits of having the Enhanced Hadoop. However, all other sequences have better performance when we use them as common feature using the Enhanced Hadoop rather than Native Hadoop since they are not present in all data files.

5.1 Comparing Native Hadoop with Enhanced Hadoop

The above example gives us indications of positive results from the implementation in the number of blocks that are read from HDFS. Figures 5.1 and 5.2 show the results, which are the number of read operations and CPU processing time in native Hadoop compared with Enhanced Hadoop.

5.1.1 HDFS: Number of Read Operations

Number of read operations is one component of Hadoop MapReduce and it is the number of times that MapReduce reads blocks from HDFS. So, based on the data size we can determine the number of blocks that should be read by the MapReduce job. As we mentioned before, by reducing the number of read operations we can improve the performance.

Figure 5.1 shows improvement in Hadoop performance by reducing the number of read operations from HDFS. In native Hadoop, the number of read operations remains the same in every job because it reads all data files again during each job. While, in Enhanced Hadoop there is difference in number of read operations based on how frequent the sequence exists in the DNA.

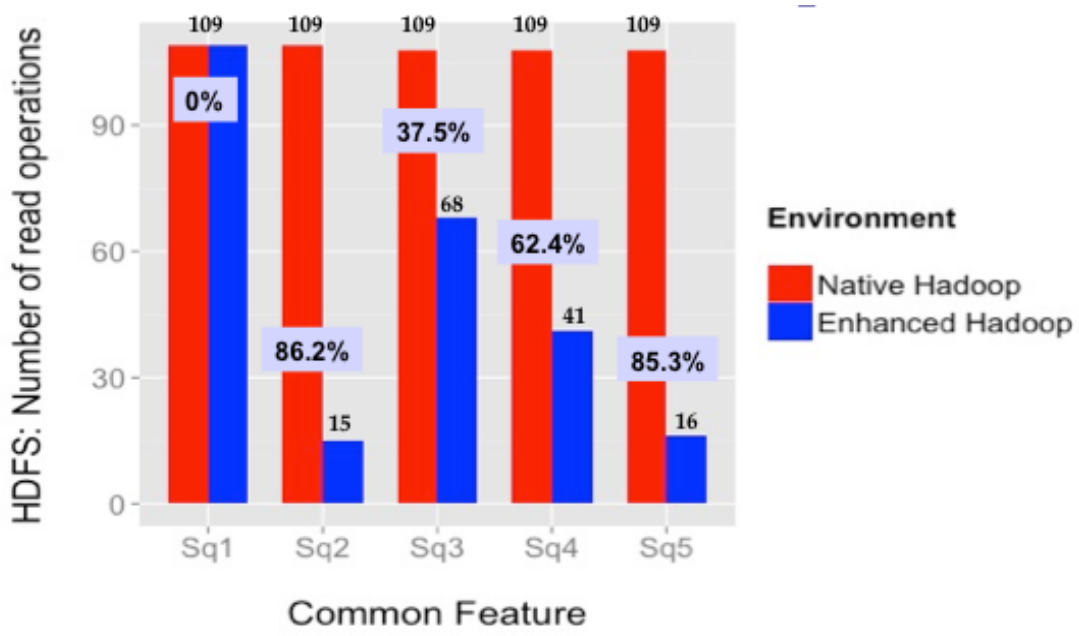


Figure 5.1: Number of read operations in Native Hadoop and Enhanced Hadoop for the same jobs.

When we implemented native Hadoop, the number of read operations was 109. By implementing the Enhanced Hadoop, the number of read operations was reduced to be 15, which is an 86.2% improvement in performance. On the other hand, since sequence1 exists in every chromosome, the number of read operations remains the same 109 in Enhanced Hadoop as native Hadoop.

One additional point that we should mention is the length of the sequence. Finding short sequences in length take less time than finding longer ones. However, the chance of having a common feature that is very long is minute as we explained before in Table 3.2.

5.1.2 CPU Processing Time

Another Hadoop MapReduce component is CPU processing time. Figure 5.2 shows the processing time of each feature in DNA data files, which used for finding the sequence of jobs in both native Hadoop and enhanced Hadoop.

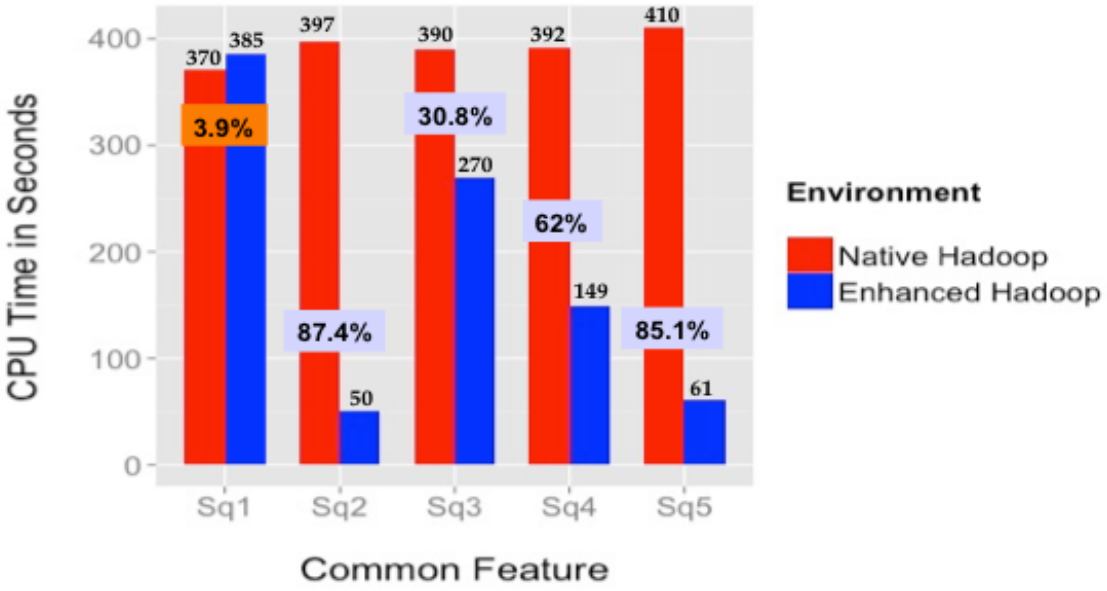


Figure 5.2. CPU processing time in Native Hadoop and Enhanced Hadoop for the same jobs.

We can see a huge difference in the CPU processing-time in enhanced Hadoop, which is less than native Hadoop since H2Hadoop does not read all data blocks from HDFS. For example, CPU processing-time in native Hadoop to process the job search for sequence2 is 397 seconds whereas it is 50 seconds in enhanced Hadoop. Figure 5.2 shows that enhanced Hadoop reduces the CPU processing time by 87.4% compared to native Hadoop.

However, in sequence1 the CPU processing time in native Hadoop is less than enhanced Hadoop. Since sequence 1 exists in all chromosomes, enhanced Hadoop reduces the efficiency by 3.9%. So, there is an overhead time in enhanced Hadoop, which is the process of looking for related jobs in the lookup table (CJBT) in enhanced Hadoop.

Although, this might happen it rarely occurs based on our study showed above in Table 3.2. This overhead is exists in all jobs because it is the processing time of checking the lookup table. However, it costs very tiny amount of time comparing with the benefit that can be gained by using enhanced Hadoop.

There are different factors in native Hadoop we can study and then compare with Enhanced Hadoop. Table 5.1 shows the processing results when finding the job sequence in sequence2, which is (AAGACGGTGGTAAGG) in DNA data blocks. So, we can say that all operations or factors that are related to output from MapReduce remain the same in both native Hadoop and enhanced Hadoop. That is because our improvement is to reduce the input to MapReduce not its output. So, the number of write operations is the same in both native Hadoop and enhanced Hadoop, which is 1 since the result is the same and its size is very small.

Finding the location of the data blocks with the common features can result in latency during the reading process. However, the benefits of the proposed system are much more than the disadvantages. Advantages of the proposed system go beyond the number of read operations and the performance of the system. The proposed system further reduces the data transfer within the network and reduces the cost of execution of the MapReduce job as the number of active DataNodes during the action of a job reduces.

Table 5.1: A list of factors that we can use to compare between native Hadoop and Enhanced Hadoop for sequence2 results.

Facility: AAGACGGTGGTAAGG	NativeHadoop	EnhancedHadoop
HDFS: Number of bytes read	2849886343	366081652
HDFS: Number of bytes written	18	18
HDFS: Number of read operations	109	15
HDFS: Number of write operations	1	1
Launched map tasks	54	7
Launched reduce tasks	1	1
Data-local map tasks	54	7
Total time spent by all maps in occupied slots (ms)	1081818	125025
Total time spent by all reduces in occupied slots (ms)	541908	51611
Map input records	40136519	5155633
Map output records	40136519	5155633
Map output bytes	802730380	103112660
Input split bytes	6016	1610
Reduce shuffle bytes	883003742	113423968
Reduce input records	40136519	5155633
Reduce output records	1	1
CPU time spent (ms)	396850	50650
Physical memory (bytes) snapshot	10718003200	1559015424
Virtual memory (bytes) snapshot	21330644992	3103256576
Total committed heap usage (bytes)	8884019200	1328115712

5.2 Comparing Amazon EMR with Enhanced Hadoop

Another experiments have been done on Amazon Web Services using S3 and EMR. The experiments show that there are indications that Enhanced Hadoop provides more positive results than Amazon EMR environment. Since Enhanced Hadoop provides less data read in size, it reduces many related factors to the jobs, such as CPU processing time, number of read operations, and the data read size in bytes.

5.2.1 Number of Read Operations Form HDFS and S3

The most important factor for Enhanced Hadoop is the number of read operations and data read size, which means the number of blocks that are read by the system. In addition, the block data size also has an important role here. Also, we have to mention about the length of the common features because it plays an important role in the processing time. The processing time for longer sequences is more than the processing of the shorter one. Figure 5.3 shows one of the results, which is the number of read operations in Amazon EMR compared with Enhanced Hadoop.

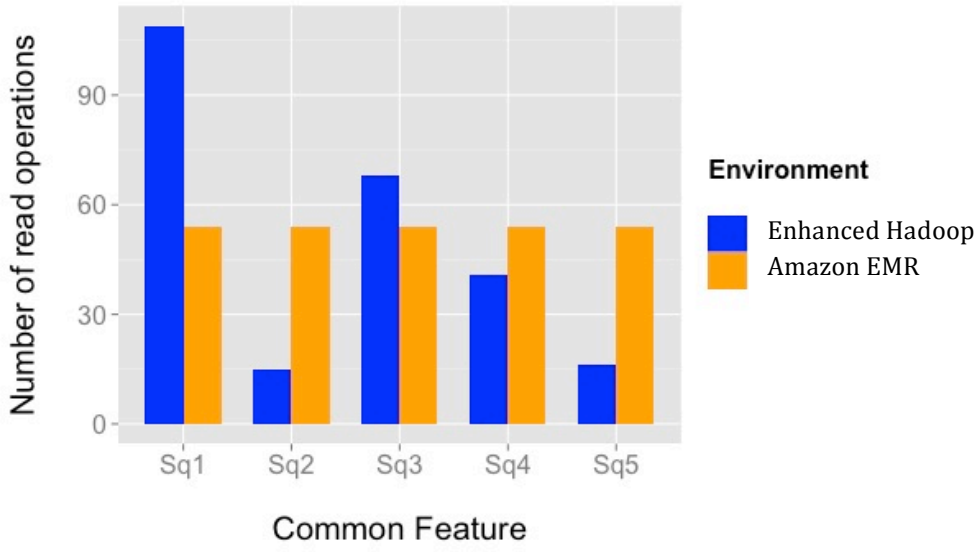


Figure 5.3: Number of read operations in Amazon EMR and Enhanced Hadoop for the same jobs.

Also, Figure 5.3 shows the comparison between Enhanced Hadoop, which reads data from HDFS and Amazon EMR, which read data from S3. In Amazon EMR, the number of read operations remains the same in every sequence, which is 54 times because it reads all data sets again during each job. While, in Enhanced Hadoop there is a difference in the number of read operations based on how frequent the sequence exists in the DNA.

By using Enhanced Hadoop, the number of read operations for sequence2 was reduced to 15 times, which increases the efficiency by 72.2%. On the other hand, sequence1 exists in every chromosome, so the number of read operations in Amazon EMR is 54, which is less than the number of read operations in Enhanced Hadoop, which is double because the block size in Amazon EMR is double the block size in Enhanced Hadoop. So, we can pretend the two jobs cost the same number of read operations.

5.2.2 CPU Processing Time

Figure 5.4 shows the CPU processing time of each sequence in Amazon EMR and Enhanced Hadoop. In Enhanced Hadoop, we can see a huge difference between the CPU processing time for Enhanced Hadoop and Amazon EMR, since Enhanced Hadoop does not read all data blocks from HDFS.

For example, the CPU processing time in Amazon EMR for sequence2 is 669 seconds whereas it is 50 seconds in Enhanced Hadoop, which speeds up the processing performance by 92.5%. For sequence1, the CPU processing time in Amazon EMR is more than Enhanced Hadoop even if they read the all data sets each time. That is because the Amazon S3 stores data in nodes where they are not in the same rack.

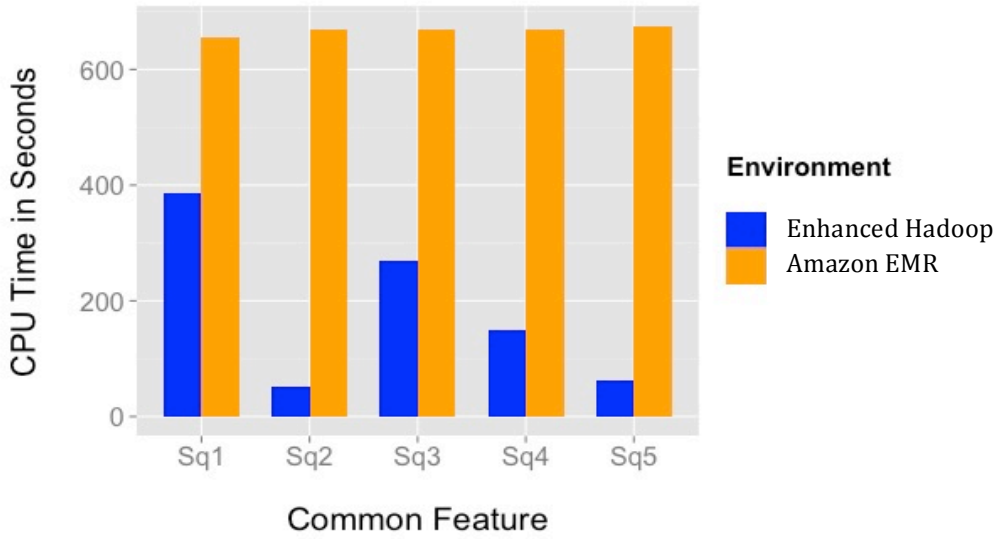


Figure 5.4: CPU processing time in Amazon EMR and Enhanced Hadoop for the same jobs.

5.2.3 Number of Bytes Read

The third comparison we have is about the data read size, which is the number of bytes that are read by the system. Figure 5.5 shows the comparison between Enhanced Hadoop and Amazon EMR for the sequences that we tested in our work. The numbers show for sequence1, that the data read size is almost the same in both environments, which is more than 2 GB (2850102554 bytes in Amazon EMR and 2849892769 in Enhanced Hadoop) because they read all of the data each time. However, in sequence2 it is more than 2 GB in Amazon EMR (2,850,088,714 bytes) and less than 500 MB in Enhanced Hadoop (366,081,652 bytes).

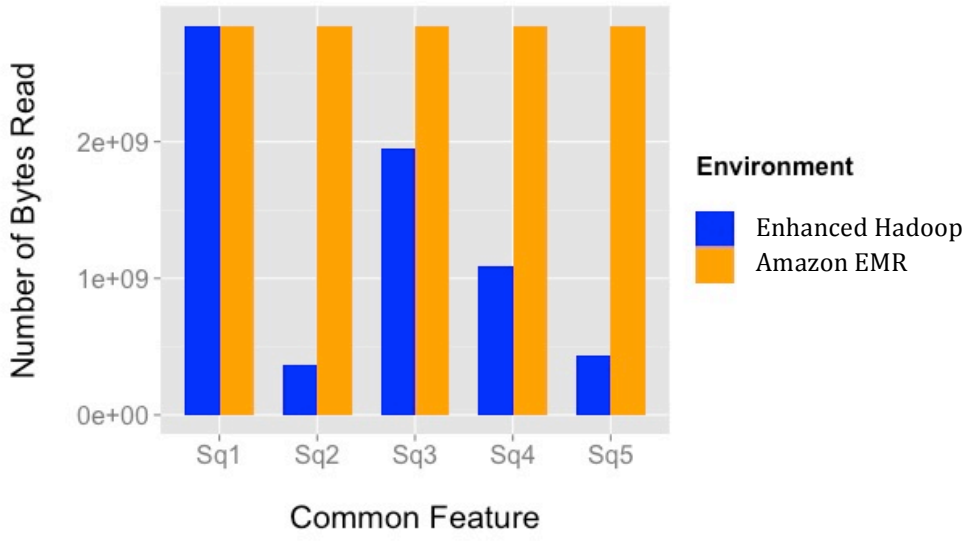


Figure 5.5 Number of Bytes Read in GB in Amazon EMR and Enhanced Hadoop for the same jobs.

The previous comparisons clearly show that the Enhanced Hadoop provide more efficiency in MapReduce jobs than Amazon EMR in jobs that are about searching for sequences in pure text files. In addition to these comparisons, there are different factors that we can use to compare between Enhanced Hadoop and Amazon EMR, which support all of our observations and results.

CHAPTER 6: CONCLUSIONS

In this research we proposed an enhanced Hadoop Architecture. The proposed architecture allows NameNode to identify the blocks in the cluster where certain information is stored. We discussed the proposed architecture in Hadoop MapReduce and compared the expected performance of proposed architecture to that of native Hadoop.

In Enhanced Hadoop architecture, the data size and number of read operations is reduced as the number of DataNodes carrying the source data blocks is identified prior to sending a job to TaskTracker. The maximum number of data blocks that the TaskTracker will assign the job to, be equal to the number of blocks that carries the source data related to a specific common job.

Finding the location of the data blocks with the common features can result in latency during the reading process. However, the benefits of the proposed system are much more than the disadvantages. Advantages of the proposed system go beyond the number of read operations and the performance of the system. The proposed system further reduces the data transfer within the network and reduces the cost of execution of the MapReduce job as the number of active DataNodes during the action of a job reduces.

The implemented jobs read real text data sets from HDFS in enhanced Hadoop as well as in native Hadoop while Amazon EMR reads from S3. The shared factor between the three environments is the size of read data. Read data size plays a main role in

improving Hadoop performance. We improved the Hadoop performance in enhanced Hadoop by 65.4% on average comparing with native Hadoop, and by 72% on average comparing with the Amazon EMR environment.

The results show more efficient work in Enhanced Hadoop comparing with Amazon EMR and native Hadoop in terms of reading less data size. So, it improves the related factors to data size, such as CPU processing time, number of read operations, and the input data size of MapReduce job. Some other factors cause latency in Amazon EMR processing that are we can avoid when implementing our private Cloud Computing cluster, such as network bandwidth, which causes delays in retrieving the data from the DataNodes.

The proposed system has some limitations. Enhanced Hadoop currently works on text data that contain patterns that users try to find frequently. The proposed Hadoop implementation works on top of Hadoop and provides sufficient results. However, if the enhanced Hadoop is coded as part of the core of native Hadoop, we may be able to increase performance of the enhanced Hadoop. In addition, users don't have to worry about the data source files' names.

In continuation of this research, the following can be investigated in the future:

- Improvements to the implementation, such as including the enhanced Hadoop source code to native Hadoop code. In addition, the implementation could be developed to be a web service.
- Consider other data types, such as numerical data.

- Improve user interface to be friendlier than the existing interface in Hadoop, which is the command line.
- Develop the Common Job Block Table CJBT using different implementation environment than HBase. Since CJBT is in a key value format, it might be presented using traditional database systems, such as RDBMS.

REFERENCES

[1] M. Meng, J. Gao, and J.-j. Chen, "Blast-Parallel: The parallelizing implementation of sequence alignment algorithms based on Hadoop platform," in *6th International Conference on Biomedical Engineering and Informatics (BMEI)*, pp. 465-470, 2013.

[2] M. C. Schatz, B. Langmead, and S. L. Salzberg, "Cloud computing and the DNA data race," *Nature biotechnology,* vol. 28, pp. 691-693, 2010.

[3] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," *Nature Reviews Genetics,* vol. 11, pp. 647-657, 2010.

[4] K. Farrahi and D. Gatica-Perez, "A probabilistic approach to mining mobile phone data sequences," *Personal Ubiquitous Computing,* vol. 18, pp. 223-238, 2014.

[5] V. Marx, "Biology: The big challenges of big data," *Nature,* vol. 498, pp. 255-260, 2013.

[6] S. Lohr, "The age of big data," *New York Times,* vol. 11, 2012.

[7] M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," *Mobile Networks and Applications,* vol. 19, pp. 171-209, 2014.

[8] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan*, et al.*, "Big data and its technical challenges," *Communications of the ACM,* vol. 57, pp. 86-94, 2014.

[9] T. White, *Hadoop: The definitive guide*: " O'Reilly Media, Inc.", 2012.

[10] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," in *International Conference on Engineering (NUiCONE), Nirma University*, pp. 1-5, 2012.

[11] M. Hammoud and M. F. Sakr, "Locality-Aware Reduce Task Scheduling for MapReduce," in *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 570-576, 2011.

[12] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.

[13] F. Li, B. C. Ooi, M. Tamer, #214, zsu, and S. Wu, "Distributed data management using MapReduce," *ACM Computing Survey,* vol. 46, pp. 1-42, 2014.

[14] W. Xu, W. Luo, and N. Woodward, "Analysis and optimization of data import with hadoop," pp. 1058-1066, 2012.

[15] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis*, et al.*, "SciHadoop: Array-based query processing in Hadoop," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1-11, 2011.

[16] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce Online," in *NSDI*, pp. 20-35, 2010.

[17] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden*, et al.*, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 165-178, 2009.

[18] A. H. Zookeeper, "http://hadoop.apache.org/zookeeper/," accessed Feb 2015.

[19] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop high availability through metadata replication," presented at the Proceedings of the first international workshop on Cloud data management, pp 37-44, 2009.

[20] S. Chandrasekar, R. Dakshinamurthy, P. G. Seshakumar, B. Prabavathy, and C. Babu, "A novel indexing scheme for efficient handling of small files in Hadoop Distributed File System," in *International Conference on Computer Communication and Informatics (ICCCI)*, pp. 1-8, 2013.

[21] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot*, et al.*, "Online aggregation and continuous query support in mapreduce," in *ACM SIGMOD International Conference on Management of data*, pp. 1115-1118, 2010.

[22] P. L.-O. Corporation, "Leveraging Massively Parallel Processing in an Oracle Environment for Big Data Analytics,", Oracle White Paper, November 2010.

[23] S. Madden, "From databases to big data," *IEEE Internet Computing,* vol. 16, pp. 4-6, 2012.

[24] J. A. Cuff and G. J. Barton, "Application of multiple sequence alignment profiles to improve protein secondary structure prediction," *Proteins: Structure, Function, and Bioinformatics,* vol. 40, pp. 502-511, 2000.

[25] S. Leo, F. Santoni, and G. Zanetti, "Biodoop: Bioinformatics on Hadoop, Parallel Processing Workshops," *International Conference on Parallel Processing Workshops,* pp. 415-422, 2009.

[26] H. Alshammari, H. Bajwa, and J. Lee, "Hadoop Based Enhanced Cloud Architecture," presented at the ASEE, USA, 2014.

[27] P. C. Church, A. Goscinski, K. Holt, M. Inouye, A. Ghoting, K. Makarychev*, et al.*, "Design of multiple sequence alignment algorithms on parallel, distributed memory supercomputers," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC*, pp. 924-927, 2011.

[28] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," in *IEEE Fourth International Conference on eScience*, pp. 222-229, 2008.

[29] Y. Liu, B. Schmidt, and D. L. Maskell, "DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI," *BMC bioinformatics,* vol. 12, pp. 85-97, 2011.

[30] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky*, et al.*, "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data," *Genome research,* vol. 20, pp. 1297-1303, 2010.

[31] P. Khatri and S. Drăghici, "Ontological analysis of gene expression data: current tools, limitations, and open problems," *Bioinformatics,* vol. 21, pp. 3587-3595, 2005.

[32] H. Mathkour and M. Ahmad, "Genome sequence analysis: a survey," *Journal of Computer Science,* vol. 5, pp. 651-660, 2009.

[33] G. S. Sadasivam and G. Baktavatchalam, "A novel approach to multiple sequence alignment using hadoop data grids," presented at the Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud, pp. 1-7, 2010.

[34] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biology,* vol. 11, pp. 207-214, 2010.

[35] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan*, et al.*, "SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters," *Journal of Parallel and Distributed Computing,* vol. 74, pp. 2166-2179, 2014.

[36] S. Zhang, J. Han, Z. Liu, K. Wang, and S. Feng, "Accelerating MapReduce with distributed memory cache," in *15th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 472-478, 2009.

[37] R. Nanduri, N. Maheshwari, A. Reddyraja, and V. Varma, "Job aware scheduling algorithm for mapreduce framework," in *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 724-729, 2011.

[38] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55,* pp. 1-16, 2009.

[39] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," in *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 40-47, 2011.

[40] Y. Jinshuang, Y. Xiaoliang, G. Rong, Y. Chunfeng, and H. Yihua, "Performance Optimization for Short MapReduce Job Execution in Hadoop," in *Second International Conference on Cloud and Green Computing (CGC)*, pp. 688-694, 2012.

[41] Z. Xiaohong, Z. Zhiyong, F. Shengzhong, T. Bibo, and F. Jianping, "Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments," in *IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pp. 120-126, 2011.

[42] C. Qi, L. Cheng, and X. Zhen, "Improving MapReduce Performance Using Smart Speculative Execution Strategy," *IEEE Transactions on Computers,* vol. 63, pp. 954-967, 2014.

[43] Apache, "Centralized Cache Management in HDFS," Update date 2014.

[44] L. Longbin, Z. Jingyu, Z. Long, L. Huakang, L. Yanchao, T. Feilong*, et al.*, "ShmStreaming: A Shared Memory Approach for Improving Hadoop Streaming Performance," in *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 137-144, 2013.

[45] J. Zhang, G. Wu, X. Hu, and X. Wu, "A Distributed Cache for Hadoop Distributed File System in Real-Time Cloud Services," presented at the Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, pp. 12-21, 2012.

[46] L. Kun, D. Dong, Z. Xuehai, S. Mingming, L. Changlong, and Z. Hang, "Unbinds data and tasks to improving the Hadoop performance," in *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 1-6, 2014.

[47] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for MapReduce in a cloud," in *Proceedings of 2011 International*

Conference for High Performance Computing, Networking, Storage and Analysis, pp. 58-71, 2011.

[48] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in Hadoop," in *11th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 200-205, 2014.

[49] M. Ishii, J. Han, and H. Makino, "Design and performance evaluation for hadoop clusters on virtualized environment," in *International Conference on Information Networking (ICOIN)*, pp. 244-249, 2013.

[50] Y. Xiao and H. Bo, "Bi-Hadoop: Extending Hadoop to Improve Support for Binary-Input Applications," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 245-252, 2013.

[51] T. Jian, M. Shicong, M. Xiaoqiao, and Z. Li, "Improving ReduceTask data locality for sequential MapReduce jobs," in *INFOCOM, 2013 Proceedings IEEE*, pp. 1627-1635, 2013.

[52] Z. Qi, L. Ling, L. Kisung, Z. Yang, A. Singh, N. Mandagere*, et al.*, "Improving Hadoop Service Provisioning in a Geographically Distributed Cloud," in *IEEE 7th International Conference on Cloud Computing (CLOUD)*, pp. 432-439, 2014.

[53] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *OSDI*, pp. 1-7, 2008.

[54] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors*, et al.*, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *IEEE*

International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1-9, 2010.

[55] I. Elghandour and A. Aboulnaga, "ReStore: reusing results of MapReduce jobs," *Proc. VLDB Endow,* vol. 5, pp. 586-597, 2012.

[56] Z. Qi, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," *IEEE Transactions on Cloud Computing,* vol. 2, pp. 14-28, 2014.

[57] W. Jiadong and H. Bo, "Improving MapReduce Performance by Streaming Input Data from Multiple Replicas," in *IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 623-630, 2013.

[58] M. Y. Eltabakh, Y. Tian, Fatma, #214, zcan, R. Gemulla*, et al.*, "CoHadoop: flexible data placement and its exploitation in Hadoop," *Proc. VLDB Endow.,* vol. 4, pp. 575-585, 2011.

[59] S. Nishanth, B. Radhikaa, T. J. Ragavendar, C. Babu, and B. Prabavathy, "CoHadoop++: A load balanced data co-location in Hadoop Distributed File System," in *Fifth International Conference on Advanced Computing (ICoAC)*, pp. 100-105, 2013.

[60] N. Tiwari, S. Sarkar, U. Bellur, and M. Indrawan, "Classification Framework of MapReduce Scheduling Algorithms," *ACM Comput. Surev.,* vol. 47, pp. 1-38, 2015.

[61] A. Eldawy, "Spatialhadoop: towards flexible and scalable spatial processing using mapreduce," in *Proceedings of the 2014 SIGMOD PhD symposium. ACM*, pp. 46-50, 2014.

[62] L. Xuhui, H. Jizhong, Z. Yunqin, H. Chengde, and H. Xubin, "Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS," in *IEEE International Conference on Cluster Computing and Workshops, CLUSTER '09*, pp. 1-8, 2009.

[63] H. Alshammari, H. Bajwa, and L. Jeongkyu, "Enhancing performance of Hadoop and MapReduce for scientific data using NoSQL database," in *Systems, Applications and Technology Conference (LISAT), 2015 IEEE Long Island*, pp. 1-5, 2015.

[64] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds, Proceedings of the 2013 International Conference on Software Engineering, pp. 402-411, 2013.

[65] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi, "Query optimization for massively parallel data processing," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pp. 12-23, 2011.

[66] H. Herodotou, "Hadoop performance models," *arXiv preprint arXiv:1106.0940,* vol. 1, pp. 1-16, 2011.

[67] H. Alshammari, J. Lee, and H. Bajwa, "H2Hadoop: Improving Hadoop Performance using the Metadata of Related Jobs," *IEEE Transactions on Cloud Computing,* vol. (PP), 2016.

[68] G. S. Sadasivam and G. Baktavatchalam, "A novel approach to multiple sequence alignment using hadoop data grids," in *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, pp. 1-7, 2010.

[69] K. Erodula, C. Bach, and H. Bajwa, "Use of Multi Threaded Asynchronous DNA Sequence Pattern Searching Tool to Identifying Zinc-Finger-Nuclease Binding Sites on the Human Genome," in *Eighth International Conference on Information Technology: New Generations (ITNG)*, pp. 985-991, 2011.

[70] N. Yin and M. G. Hluchyj, "Analysis of the leaky bucket algorithm for on-off data sources," in *Global Telecommunications Conference, 1991. GLOBECOM '91. 'Countdown to the New Millennium. Featuring a Mini-Theme on: Personal Communications Services*, vol.1, pp. 254-260, 1991.

[71] M. N. Vora, "Hadoop-HBase for large-scale data," in *International Conference on Computer Science and Network Technology (ICCSNT)*, pp. 601-605, 2011.

[72] M. G. Noll, "Running hadoop on ubuntu linux (multi-node cluster)," *Apr-2013.[Online]. Available:* http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-nodecluster/

[73] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of MapReduce programs," *Proceedings of the VLDB Endowment,* vol. 4, pp. 1111-1122, 2011.

APPENDIX A: PUBLICATIONS

Journal Papers:

- Alshammari, H., J. Lee, and H. Bajwa, "*H2Hadoop: Improving Hadoop Performance using the Metadata of Related Jobs"*. IEEE Transactions on Cloud Computing, 2016.
- Hamoud Alshammari, Jeongkyu Lee, Hassan Bajwa "Improves Current Hadoop MapReduce Workflow and Performance" (International Journal for Computer Applications, 2015).

Conferences Papers:

- H. Alshammari, H. Bajwa, and J. Lee, "Hadoop Based Enhanced Cloud Architecture for Bioinformatics Algorithms," IEEE – LISAT, USA, 2014.
- Hamoud Alshammari, Jeongkyu Lee, Hassan Bajwa "Enhancing Performance of Hadoop and MapReduce for Scientific Data using NoSQL Database" IEEE – LISTA, USA, 2015.

Posters:

1. Enhancing Hadoop MapReduce Performance for Bioinformatics Data (Hamoud Alshammari, Jeongkyu Lee, Hassan Bajwa, ASEE 2014)
2. Enhancing Hadoop MapReduce Performance for Scientific Data using NoSQL Database, (Hamoud Alshammari, Jeongkyu Lee, Hassan Bajwa, Faculty Research day at University of Bridgeport, 2015)