PERFORMANCE ANALYSIS OF THE TRANSACTIONS IN BANKING SYSTEM

[1]Janarthanan.R
[2]Ragu.G, [3]Monica, [4]Reshma.M,, [5]Yogeshwari
[1]Hod[2]Asst.Professor[3][4][5]UG Student, Department of Computer Science and Engineering
T.J.S Engineering College
[1] hodcse@tjsenggcollege.com, [2]40@yahoo.com, [3]monigoodmoni@ gmail.com,
[4]reshmamagi0920@gmail.com,, [5]yokehwarigowri92@ gmail.com,

Abstract
—Cloud Computing leverages Hadoop framework for processing BigData in parallel. Hadoop has certain limitations that
could be exploited to execute the job efficiently. These limitations are mostly because of data locality in the cluster, jobs and tasks
scheduling, and resource allocations in Hadoop. Efficient resource allocation remains a challenge in Cloud Computing MapReduce
platforms. We propose H2Hadoop, which is an enhanced Hadoop architecture that reduces the computation cost associated with
BigData analysis. The proposed architecture also addresses the issue of resource allocation in native Hadoop. H2Hadoop provides a
better solution for "text data", such as finding DNA sequence and the motif of a DNA sequence. Also, H2Hadoop provides an efficient
Data Mining approach for Cloud Computing environments. H2Hadoop architecture leverages on NameNode's ability to assign jobs to
theTaskTrakers (DataNodes) within the cluster. By adding control features to the NameNode, H2Hadoop can intelligently direct and
assign tasks to the DataNodes that contain the required data without sending the job to the whole cluster. Comparing with native
Hadoop, H2Hadoop reduces CPU time, number of read operations, and another Hadoop factors
.

Index Terms—

BigData, Cloud Computing, Hadoop, H2Hadoop, Hadoop Performance, MapReduce, Text Data

1 INTRODUCTION

parallel processing in Cloud Computing has emerged as an interdisciplinary research area due to the heterogeneous nature and large size of data. Translating sequential data to meaningful information requires substantial computational power and efficient algorithms
to identify the degree of similarities among multiple sequences [1]. Sequential pattern mining or data analysis
applications such as, DNA sequence aligning and motif finding usually require large and complex amounts of data
processing and computational capabilities [2]. Efficiently
targeting and scheduling of computational resources is required to solve such complex problems [3]. Although, some of the data sets are readable by humans,
it can be very complex to be understood and processed using traditional processing 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 data [5]. Before
we go any further, it is necessary to define certain definitions that are related to BigData and Hadoop.

1.1BigData Concepts

There are different ways of defining and comparing BigData with the traditional data such as data size, content,
collection and processing. Big data has been defined as large data sets that cannot be processed using traditional
processing techniques, such as Relational Database Management Systems, in a tolerable processing time [6].
BigData is either a relational database (Structured), such as
stock market data or non-relational database (Semistructured

EXISTING SYSTEM

Existing concept deals with providing backend by using mysql which contains lot of drawbacks i.e data limitation is that processing time is high when the data is huge and once data is lost we cannot recover

DRAWBACKS

We can process limitation of data.

We get results which take more time and maintenance cost is very high.

PROPOSED SYSTEM

Proposed concept deals with providing database by using hadoop tool

we can analyze no limitation of data and simple add number of machines to the cluster

we get results with less time, high throughput and maintenance cost is very less

we are using joins, partations and bucketing techniques in hadoop.

Drawbacks

We can process limitation of data.

We get results which take more time and maintenance cost is very high.

or Unstructured), such as social media data or
DNA data sets [7].
The 4V's of BigData are 1) Volume of the data, which means the data size. Some of companies' data storage is aboutZetabyte. 2) Velocity, which means the speed at which the data is generated. 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 the data is to these applications [8].
Different challenges in BigData have been discussed in previous research [9] and they are described as technical challenges such as the physical storage, that stores the BigData and reduce the redundancy. Also, there are many challenges such as the process of extracting the information,
cleaning data, data integration, data aggregation, and data representation. Since BigData has these issues, it needs such
an environment or framework to work through these challenges. Hadoop, which works with BigData sets, is a framework that most organizations use to process BigData
in order to overcome data challenges.
P

• *HamoudAlshammari, Department of Computer Science and Engineering,*
221 University !Ave Bridgeport, Connecticut.
e-mail: halshamm@my.bridgeport.edu !
• *Jeongkyu Lee, Associate Professor of Computer Science,*
221 University !Ave Bridgeport, Connecticut.
e-mail: jelee@bridgepoert.edu !
• *Hassan Bajwa, Associate Professor of Electrical Engineering,*
221 !University Ave Bridgeport, Connecticut.
e-mail: hbajwa@bridgeport.edu)

Fig. 1. Overall MapReduceWordCountMapReduce Job

1.2 Hadoop Overview

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 [10].
The two main components of Hadoop are (i) Hadoop Distributed File System (HDFS) that provides the data reliability (distributed storage) and (ii) MapReduce that provides the system analysis (distributed processing) [11]
[10]. Relying on the principle that "moving computation
towards data is cheaper than moving data towards computation" [12], Hadoop employs HDFS to store large
data files across the cluster.
MapReduce provides stream reading access, runs tasks

on a cluster of nodes, and provides a data managing system
for a distributed data storage system [13]. MapReduce algorithm has been used for applications such as generating
search indexes, document clustering, access log analysis, and different other kinds of data analysis [14].
"Write-once and read-many" is an approach that 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 assigned jobs [10]. 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 set to 3 times by default [15].
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 file system including saving the data and directing the jobs to the appropriateDataNodes that store related application data [16]. DataNodes facilitate Hadoop/MapReduce to process the jobs with streaming execution in a parallel processing environment [10, 17].
Running on the master node, JobTracker coordinates and deploys the applications to the DataNodes with TaskTracker services for execution and parallel processing
[15]. Each task is executed in an available slot in a DataNode, which is configured with a fixed number of map
slots, and another fixed number of reduced slots. The data inMapReduce for our purposes is in a text format, so both the input and output of data must also be in a text file format [10].
The master computer has two daemons, which are NameNode in terms of HDFS and JobTracker in terms of MapReduce. Similarly, the slaves also have two daemons, which are DataNodes in terms of HDFS and TaskTrackers in terms of MapReduce.

1.3What 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. During MapReduce, multiple Maps are processed in parallel followed by Reduce tasks also processed in parallel. Depending upon applications the numbers of maps can be different than that
of reduces. Storing data in HDFS has different forms [8] such as <Key, Value> concept to determine the given parameter (Key) and to retrieve the required result (Value)
at the end of the job.
For example, a "WordCount" job counts number of replication of each word in the data files. Figure 1 explains
MapReduce example "WordCount" as a common example
to apply MapReduce in such unstructured data like books.
As an input file, it consists of a sequence of characters that
are separated by space, so we can consider the space as a
delimiter that separates words. First step, Hadoop divides
the data to blocks in the Splitting phase. Then, the Mapping
phase does <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, the Reducing phase provides the addition of values to have one final value for each key. Finally, NameNode provides a final result that has all keys and their values as one final result
from the MapReduce job.

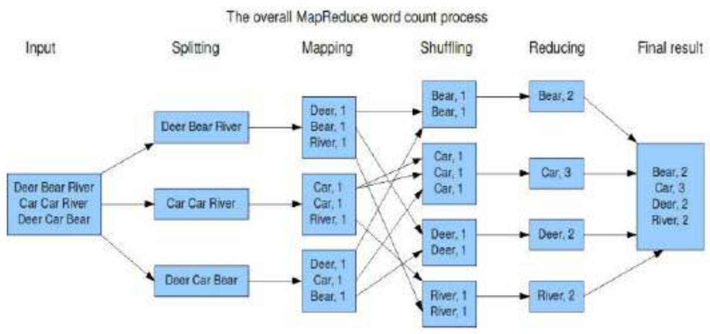


Fig. 1. Overall MapReduceWordCountMapReduce Job

The rest of the paper is divided as follows. In section II, we go over an overview of HadoopMapReduce performance and focus on the parameters that can be developed to improve the performance of Hadoop. In section III, we discuss the Hadoop workflow and its limitations in terms of the MapReduce algorithm performance. Section IV discusses the problem that this research tries to solve. Then in section V, we will propose our enhanced HadoopMapReduce workflow and compare the two architectures in terms of developing MapReduce performance. In section VI, the implementation and testing
phases are discussed, and the results are evaluated and discussed in section VII. In section VIII, we discuss some related works that have been proposed to improve the Hadoop performance. Finally in section IX, the conclusion
is provided.

2 OVERVIEW OF 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 [18] 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:
• Hadoop Parameters: which is a set of predefined configuration parameters that are in Hadoop setting files.
• Profile Statistics: which are a set of user-defined properties of input data and functions like Map, Reduce, or Combine.
• 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.
$NumberOfBlocks = DataSize / BlockSize$(1)
Where DataSize is the size of the input 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 [18].
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:
$IOCostRead = NumberOfBloks X HdfsReadCost$(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:
$IOCostWrite = NumberOfBloks X HdfsWriteCost$(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:
$CPUCostRead = NumberOfBlocks X InUncompeCPUCost$
$+ InputMapPairs X MapCPUCost$(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.
Readers can find more details about the Hadoop performanceanalyzing model in [18] which is published by
Duke university and considered as the most common paper
that discussed the Hadoop performance model.

2 NATIVE HADOOP WORKFLOW

In current HadoopMapReduce architecture, the client first sends a job to the cluster administrator, which is the
NameNode. The job can be sent either using Hadoop ecosystem (Query language such as Hive) or by writing a
job source code [19]. Before that, the data source files should be uploaded to the HDFS 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 DataNodes 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 where the results will be stored in.
Native Hadoop architecture follows the concept of "write-once and read-many," so there is no ability to make
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 the 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 solutions about data locality to address the issue of latency while reading data from DataNodes [20]. Hadoop falls short of query optimization and reliability of conventional database
systems.
In the existing HadoopMapReduce architecture, multiple jobs with the same data set work completely independent of each other. We also noticed that searching for the same sequence of characters. For example in any text
format data requires the same amount of time each time we
2016
2016

Fig. 2. Native HadoopMapReduce Workflow
Fig. 3. Native HadoopMapReduce Workflow Flowchart
execute the same job. Also, searching for the supersequence
of a sequence that has already been searched
requires the same amount of time.

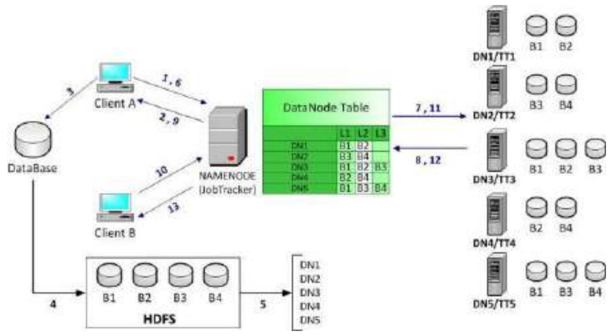


Fig. 2. Native HadoopMapReduce Workflow

3.1 Native HadoopMapReduce Workflow

MapReduce workflow in native Hadoop has been explained in figure 2 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
inHadoop.
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
job (job2) will go through step 1-5 even if the datasets are already available in HDFS. However,
if client "B" knows that the data exists in HDFS, it will send 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 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.

3.2 Native HadoopMapReduce Limitations

Many HadoopMapReduce jobs, especially tasks associated with the science data such as genomic data, deal
with the sequences similarities, super-sequences and subsequences
in DNA [21]. 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 sequence exists in a specific DataNode, then any superstring-sequence can only be found in the same DataNodes.

As shown in Figure 2, let’s suppose that Client A and Client B are searching for the same sequence in BigData
source files. 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. Process redundancy remains a major unsolved problem in native HadoopMapReduce infrastructure.

TABLE I
COMMON JOB BLOCKS TABLE COMPONENTS

3 RESEARCH PROLEM

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, a Hadoop cluster with three nodes is 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 than
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.
Since current Hadoop Framework does not support storing metadata of previous jobs, it ignores the location of
DataNode with sub-sequence and reads data from all DataNodes for every new job [21].
Shown in Figure 2, Client A and Client B are searching for similar sequences in BigData. Once Client A finds the sequence, Client B will repeat the search of BigData again to
find the same results. Since each job is independent, clients
do not share results. Any client looking for a super sequence with a sequence that has already been searched will have to go through the BigData search again. Thus the
cost to perform the same job will remain the same each time.

5 H2HADOOP

In existing Hadoop architecture, NameNode knows the location of the data blocks in HDFS. NameNode is responsible for assigning the jobs to a client and dividing that job into tasks. NameNode further assigns the tasks to theTasTrackers (DataNodes). Knowing which DataNode holds the blocks containing the required data, NameNode should be able to direct the jobs to the specific DataNodes without going through the whole cluster. In H2Hadoop, before assigning tasks to the DataNodes, we implemented a
pre-processing phase in the NameNode.
Our focus is on identifying and extracting features to build 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 of the cluster without going through
the whole data again. Explanation of the proposed solution
is as follows:

5.1 Common Job Blocks Table (CJBT)

Proposed HadoopMapReduce workflow (H2Hadoop) is the 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 which named Common
Job Blocks Table CJBT.
The proposed solution can only be used for text data. BigData, such as Genomic data and books can be processed
efficiently using the proposed framework. CJBT stores information about the jobs and the blocks associated with
specific data and features. This enables the related jobs to
get the results from specific blocks without checking the
entire cluster. Each 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 our research, we took an example
of genome BigData to show the functionality of enhanced
Hadoop architecture.
In order to understand the framework of Mapping and Reducing in the proposed platform, we searched for a DNA
sequence using H2Hadoop in HDFS. 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 [22]. Computationally intensive algorithms are used for sequence alignment. Scalable

parallel processing Hadoop framework has been proposed
and implemented for the sequence alignment of genomic
data [16, 23-25].
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
subsequence
is identified and updated in CJBT. Common
features 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 I), which are explained below:

entries.
Common Job

Name

Common

Feature

Block

Name

Sequence_Alignment GGGATTTA B1 B2 B3

TTTAGA B1 B4

Fining_Sequence

TTTAGCC B3 B6

GCCATTAA B1 B3 B4

AATCCAGG B3 B5

5.1.1 Common Job Name CJN

Common Job Name represents a shared name of a job
that each MapReduce client must use when submitting a
new job in order to get the benefit of the proposed
architecture. We define a library, which contains a list of
pre-coded jobs that 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 to identify and
match a job to a DataNode(s) containing block(s) in the
CJBT
.

5.1.2 Common Feature CF

Common Features are defined as the shared data
between jobs. H2Hadoop supports caching, 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.
Common Job
Name
Common
Feature
Block
Name
Sequence_Alignment GGGATTTA B1 B2 B3
TTTAGA B1 B4
Fining_Sequence
TTTAGCC B3 B6
GCCATTAA B1 B3 B4
AATCCAGG B3 B5
2016
2016

Fig. 4. H2Hadoop MapReduce Workflow
TABLE II
LIKELIHOOD OF RANDOM NUCLEOTIDES
JobTracker 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
common
feature of J1, it is logical to map the task and allocate
the
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 old one. However, feature selection should be
done carefully as the response time for the jobs can
increase
if 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 solution.
The length of common feature also plays on important role in the proposed solution. If the sequence is too short it
will 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 II). 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:
Dn = 3 x 109/ (4)n
Where n is the number of nucleotides in a random sequence.
# of Nucleotides likelihood of finding any random 9 – 15 nucleotides sequence in the
human genome: D(n)
genome 3 x 109
09 -nucleotides D9 = 11444
12 -nucleotides D12 =178
15 -nucleotides D15 =2.7
As shown in (TABLE II), the likelihood of any random 9 base pair (bp) of a long nucleotides sequence in a whole genome is quite large comparing with 12 base pair (bp), 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.

5.1.3 Block Name BN

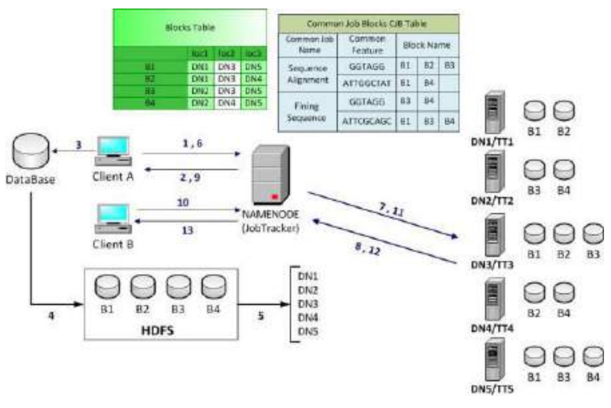
BlockName or BlockID is the location of the common features. It identifies the block(s) in a cluster where certain
information is stored. BlockName helps the NameNode direct jobs to specific DataNodes that store these blocks in
HDFS. CJBT has the list of all blocks that are related to the
results of the common feature. For example, if a sequence "TTTAGATCTAAAT" is only stored in B1 and B4, the NameNode will direct any job that has a particular sequence to B1 and B4. This CJBT is a dynamically configurable table and the BlockName entries are changing
as the common feature changes.
CJBT should not become too large because larger lookup table tends to decrease the system performance. The size of
CJBT can be limited by employing the 'leaky bucket' algorithm [26]. The 'leaky bucket' parameters can be adjusted to keep the size of CJBT constant. This can be discussed more in future work.

5.2 End-User Interface

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

5.3 H2Hadoop MapReduce Workflow

Enhanced Hadoop architecture doesn't differ from the nativeHadoop architecture so it will be enhancing only the
software level through build CJBT. Following chart (Figure
4) shows the proposed changes in NameNode, which works as a lookup table that contains metadata for the executed jobs in H2Hadoop.



Blocks Table

	loc1	loc2	loc3
B1	DN1	DN3	DN5
B2	DN1	DN3	DN4
B3	DN2	DN3	DN5
B4	DN2	DN4	DN5

Common Job Blocks CJB Table

Common Job Name	Common Feature	Block Name		
Sequence Alignment	GGTAGG	B1	B2	B3
	ATTGGCTAT	B1	B4	
Fining Sequence	GGTAGG	B3	B4	
	ATTCGCAGC	B1	B3	B4

MapReduce workflow in H2Hadoop has been explained
in figure 4 as follows:
Step 1 to Step 8: remain in the same workflow as native Hadoop. Except results from the first 7 steps are

stored in the CJBT.
Step 9: JobTracker sends the result to Client “A”. In this step, NameNode keeps the names of the blocks
that produced the results in the local lookup table
(CJBT) by the Common Job Name (Job1) that has
common feature as explained above.
Step 10: Client “B” sends a new MapReduce job “Job2” to
theJobTracker with the same common job name
and same common feature or super-sequence of
“Job1”.
Step 11: JobTracker sends “job2” to TaskTrackers who hold
the blocks, which have the first result of the
MapReduce “Job1” (DN2, DN4, DN5). In this step,
2016-
2016

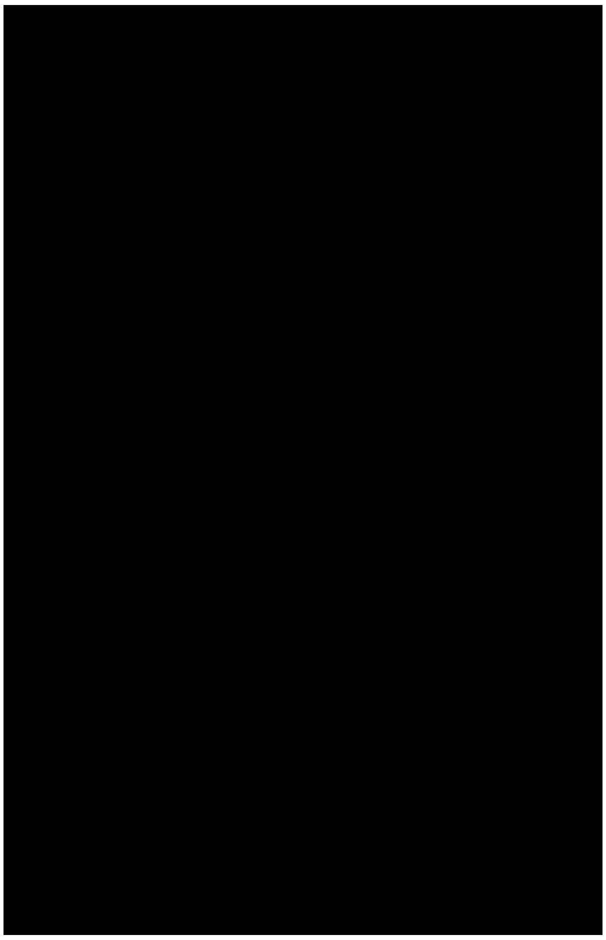


Fig. 5. H2Hadoop MapReduce Workflow Flowchart

theJobTracker starts with checking the CJBT first
to find if it is a new job which has the same
common name and common features of any
previous ones or not – In this case yes. Then the
JobTracker sends “Job2” only to TT2, TT4 and TT5.
We may assume here that the lookup table will be
updated with more details OR just remain as is
because every time we have a new job that may
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 H2Hadoop MapReduce framework. In addition, there should be a training phase before starting the process of MapReduce to have some metadata in the CJBT to receive the benefits of the new architecture.
From the flowchart that is explained in Figure 5, we can
see that there are two more conditions in H2Hadoop when
compared with native Hadoop that perform with a delay in
job processing. However, if we have a relationship between
jobs, H2Hadoop performance will be better than the native
Hadoop. The above-mentioned delay in H2Hadoop ultimately causes a short delay in time.
In H2Hadoop, 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.
Under these two conditions, H2Hadoop 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.

6 IMPLEMENTATION AND TESTING

In this section we will discuss the implementation plan for the proposed solution and expected results of H2Hadoop. We tested H2Hadoop under these specific circumstances, which include number of data files and the size of each file. The proposed solution could be implemented in two different ways. First, in cases where there are many source data files and each one is less than the default value of the block size. Second, in cases where there is a one or a couple of data source files and where most of the files are larger than the default block in size. In our implementation, we used DNA chromosome data and the data source size is about 24 files. Each file is less than the default block size in Hadoop. Various jobs were implemented using the above mentioned data. The implementation of the proposed solution goes in three parts:

6.1 Creating the Common Job Block Table (CJBT)

Using different techniques we are able to perform design and create the CJBT. One of them is using a NoSQL database such as HBase. HBase is a column-oriented database of which a main property is expanded horizontally [27].
The reason for using HBase is that it is an 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 research and enable the proposed solution works successfully. Another way is to create a key-value data structure such as dictionary in Python.

6.2 Designing User Interface (UI)

As we proposed earlier the user interface should contain user-friendly interface so that the user is receive the benefits of the enhanced design when choosing common data from lists. For example, when choosing the CJN from a
list of common job names that are related to the similar data
files.
Different forms of user interfaces can be designed based
on the user's 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 they will
2016
2016

TABLE III
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

Fig. 6. Number of read operations in Native Hadoop and H2Hadoop
for the same jobs.
Native Hadoop
H2Hadoop

use. 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. The commands that are used here are the same original Hadoops' commands.

6.3 Proposed Solution Environment

We can build a cluster for the proposed solution following some directions [28] 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 a shell interface of Linux, we will use it for the implementation of the proposed solution. We use the following applications and tools:

• We have one Master node, which is the NameNode and The JobTracker.

• We have 9 slave nodes that work as DataNodes and TaskTrackers in different locations.

• Linux OpenSUSE as an operating system on all nodes in the cluster. We used 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 HBase 0.98, which is the stable version of HBase at the time of implementing the cluster.

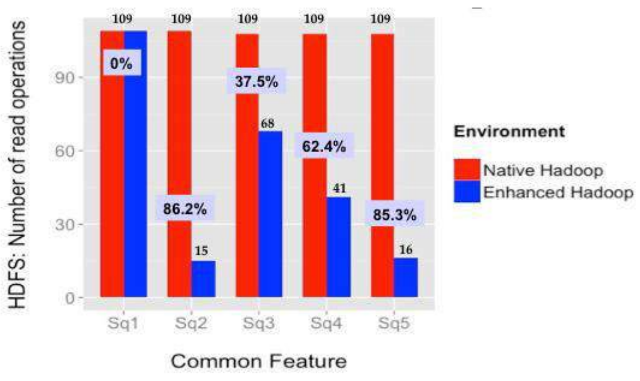
COMMON JOB BLOCK TABLE (DNA EXAMPLE)

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

•

6.4 Execute some experiments

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 prowtein process. The following examples are some sequences and their locations TABLE III (store the ChromosomeName in which chromosomes they occur): 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 III using HBase for easy access in the H2Hadoop environment.



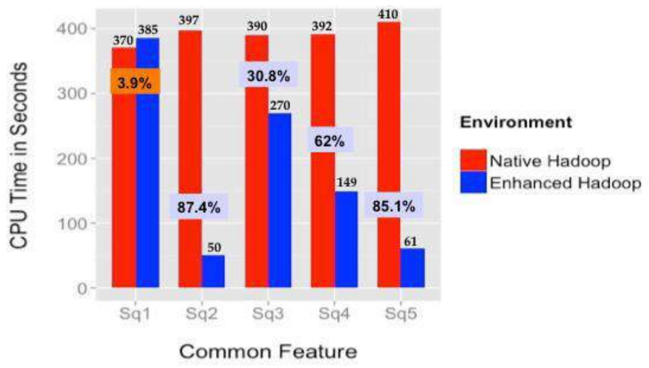
7 RESULTS AND EVALUATION

Up to this point, there are indications that we received positive 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 a direct effect on
the performance of Hadoop [29]. As expected, we also noticed that the performance of HadoopMapReduce depends upon the length of common features and the likelihood of finding the common features in the source files and DataNodes. If the common features exist in all source files, then H2Hadoop will not improve the performance as the job reads all files that contain the common feature.
From TABLE III, sequence1 is located in all chromosomes, which means it is located in all data blocks.
So, H2Hadoop will read the whole data files again if the
common feature is sequence1. In this case it gives no benefits of having H2Hadoop. However, all other sequences have better performance when we use them as
common feature using H2Hadoop rather than Native Hadoop since they are not present in all data files.
The above example gives us indications of positive results from the implementation in the number of blocks
that are read from HDFS. Figure 6 shows one of the results,
which is the number of read operations in native Hadoop
compared with H2Hadoop.
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

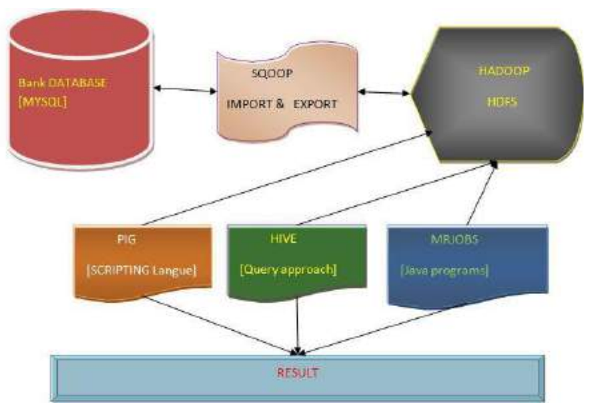
determine the number of blocks that should be read by the MapReduce job. As we mentioned before, by reducing the number of read operations we can improve the performance.
Figure 6 shows improvement in Hadoop performance by reducing the number of read operations from HDFS. In nativeHadoop, the number of read operations remains the same in every job because it reads all data files again during each job. While, in H2Hadoop there is difference in
number of read operations based on how frequent the sequence exists in the DNA. When we implemented native
Hadoop, the number of read operations was 109. By using H2Hadoop, the number of read operations was reduced to be 15, which increases the efficiency by 86.2%. On the other
hand, since sequence1 exists in every chromosome, the number of read operations remains the same 109 in H2Hadoop as native Hadoop.
One additional point that we should mention is the length of the sequence. Finding short sequences in length
2016
2016

Fig. 7. CPU processing time in Native Hadoop and H2Hadoop for
the same jobs.
Native Hadoop
H2Hadoop
Fig. 8. A list of factors that we can use to compare between
nativeHadoop and H2Hadoop for sequence2 results.
take less time than finding longer ones. However, the chance of having a common feature that is very long is minute as we explained in TABLE II.
Another HadoopMapReduce component is CPU processing time. Figure 7 shows the processing time of each
feature in DNA data files, which used for finding the sequence of jobs in both native Hadoop and H2Hadoop.
In H2Hadoop, we can see a huge difference between the CPU processing-time for H2Hadoop, which is less than
nativeHadoop since H2Hadoop does not read all data blocks from HDFS. For example, CPU processing-time in
nativeHadoop to process the job search for sequence2 is 397 seconds whereas it is 50 seconds in H2Hadoop.
Figure 7
shows that H2Hadoop reduces the CPU processing time by
87.4% compared to native Hadoop.
However, in sequence1 the CPU processing time in nativeHadoop is less than H2Hadoop. Since sequence 1 exists in all chromosomes, H2Hadoop reduces the efficiency by 3.9%. So, there is an overhead time in H2Hadoop, which is the process of looking for related jobs
in the lookup table (CJBT) in H2Hadoop. Although, this
might happen it rarely occurs based on our study showed
above in Table II. 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 H2Haddop.
There are different factors in native Hadoop we can study and then compare with Enhanced Hadoop (H2Hadoop). Figure 8 shows the processing results when
finding the job sequence in sequence2, which is (AAGACGGTGGTAAGG) in DNA data blocks.
We can say that all operations or factors that are related to output from MapReduce remain the same in both native
Hadoop and H2Hadoop. 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 H2Hadoop, 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.



SYSTEM ARCHITECHTURE



Data Preprocessing Module:

In this module we have to create Data set for bank dataset it contain set of table such that customer details, account details, transaction details overall marks details for last year

Data Migration Module with Sqoop

Sqoop is a command-line interface application for transferring data between relational databases and Hadoop

In this module we fetch the dataset into hadoop (HDFS) using sqoop Tool.

Data Analytic Module with Hive

Hive is a data ware house system for Hadoop. It runs SQL like queries called HQL (Hive query language) which gets internally converted to map reduce jobs

In this module we have to analysis the dataset using HIVE tool which will be stored in hadoop (HDFS).For analysis dataset HIVE using HQL

Data Analytic Module with Pig

Apache Pig is a high level data flow platform for execution Map Reduce programs of Hadoop. The language for Pig is pig Latin. Pig handles both structure and unstructured language

In this module also used for analyzing the Data set through Pig using Latin Script data flow language.

The Algorithm

Generally MapReduce paradigm is based on sending the computer to where the data resides!

MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.

Map stage : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

Reduce stage : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

8 RELATED WORK

Hadoop is considered as a new technology that provides processing services for BigData issues in cloud computing,
thus, research in this field is considered a hot topic. Many studies have discussed and developed different ways to improve the HadoopMapReduce performance from different considerations or aspects. Many studies have discussed optimizing Hadoop and MapReduce jobs such as
job scheduling and execution time to improve Hadoop performance. Whereas, there are many studies that have been discussed in relation to data locality in cloud computing.
One of the important features of Hadoop is the process of job scheduling [30] [31] and job execution time . Different
studies have provide some information improvements and have come up with positive results based on their assumptions [32] [33]. Others focus on the time of initialization and termination phases of MapReduce jobs [34].
System memory has many issues that could be addressed to improve the system performance. In Hadoop, Apache performs a centralized memory approach which is implemented to control the cashing and resources [35]. Apache Hadoop supports centralized data cashing. However, some studies utilize a distributed cashing approach to improve Hadoop performance [36] [37]. There
are different approaches that discuss memory issue. ShmStreaming [38] introduces a Shared memory Streaming
schema to provide lockless FIFO queue that connects Hadoop and external programs.
The location of input data has been determined in currentHadoop 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
2016
2016

10

strategy helps for various reasons, one of which is for 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. Different studies proposed solutions to improveHadoop performance by developing data locality
improvements [12] [39]. Others, focus on the type of data to
improveHadoop performance [16] [40]. In addition, a few
studies discuss different issues regarding the improvement
ofHadoop performance [41-45].

9 CONCLUSION

In this work we present Enhanced Hadoop framework (H2Hadoop), which allows a NameNode to identify the blocks in the cluster where certain information is stored.
We discussed the proposed workflow in H2Hadoop and compared the expected performance of H2Hadoop to nativeHadoop. In H2hadoop, we read less data, so we have some Hadoop factors such as number of read operations, which are reduced by the number of DataNodes carrying the source data blocks, which is identified prior to sending a job to TaskTracker. The maximum number of data blocks that the TaskTracker will
assign to the job is equal to the number of blocks that carries the source data related to a specific common job
.

REFERENCES

1. Ming, M., G. Jing, and C. Jun-jie. *Blast-Parallel: The parallelizing
implementation of sequence alignment algorithms based on
Hadoop platform*.in*Biomedical Engineering and Informatics
(BMEI), 2013 6th International Conference on*. 2013.
2. Schatz, M.C., B. Langmead, and S.L. Salzberg, *Cloud computing
and the DNA data race.* Nature biotechnology, 2010. 28(7): p. 691.
3. Schadt, E.E., et al., *Computational solutions to large-scale data
management and analysis.* Nature Reviews Genetics, 2010. 11(9):
p. 647-657.
4. Farrahi, K. and D. Gatica-Perez, *A probabilistic approach to*

mining mobile phone data sequences. Personal Ubiquitous
Comput., 2014. 18(1): p. 223-238.
5. Marx, V., *Biology: The big challenges of big data.* Nature, 2013.
498(7453): p. 255-260.
6. Lohr, S., *The age of big data.* New York Times, 2012. 11.
7. Changqing, J., et al. *Big Data Processing in Cloud Computing
Environments*.in*Pervasive Systems, Algorithms and Networks
(ISPAN), 2012 12th International Symposium on*. 2012.
8. Chen, M., S. Mao, and Y. Liu, *Big Data: A Survey.* Mobile
Networks and Applications, 2014. 19(2): p. 171-209.
9. Jagadish, H., et al., *Big data and its technical challenges.*
Communications of the ACM, 2014. 57(7): p. 86-94.
10. White, T., *Hadoop: The definitive guide*. 2012: " O'Reilly Media,
Inc.".
11. Patel, A.B., M. Birla, and U. Nair. *Addressing big data problem
usingHadoop and Map Reduce*. in*Engineering (NUiCONE), 2012
Nirma University International Conference on*. 2012.
12. Hammoud, M. and M.F. Sakr. *Locality-Aware Reduce Task
Scheduling for MapReduce*.in*Cloud Computing Technology and
Science (CloudCom), 2011 IEEE Third International Conference
on*. 2011.
13. Dean, J. and S. Ghemawat, *MapReduce: simplified data processing
on large clusters.* Communications of the ACM, 2008. 51(1): p.
107-113.
14. Li, F., et al., *Distributed data management using MapReduce.*
ACM Comput.Surv., 2014. 46(3): p. 1-42.
15. Xu, W., W. Luo, and N. Woodward. *Analysis and optimization of
data import with hadoop*. IEEE.
16. Buck, J.B., et al. *SciHadoop: Array-based query processing in
Hadoop*.in*High Performance Computing, Networking, Storage
and Analysis (SC), 2011 International Conference for*. 2011.
17. Condie, T., et al. *MapReduce Online*. in*NSDI*. 2010.
18. Herodotou, H., *Hadoop performance models.* arXiv preprint
arXiv:1106.0940, 2011.
19. Wu, S., et al. *Query optimization for massively parallel data
processing*. in*Proceedings of the 2nd ACM Symposium on Cloud
Computing*. 2011. ACM.
20. Palanisamy, B., et al. *Purlieus: locality-aware resource allocation
forMapReduce in a cloud*. in*Proceedings of 2011 International
Conference for High Performance Computing, Networking,
Storage and Analysis*.ACM.
21. Matsunaga, A., M. Tsugawa, and J. Fortes. *CloudBLAST:
Combining MapReduce and Virtualization on Distributed
Resources for Bioinformatics Applications*.in*eScience, 2008.
eScience '08. IEEE Fourth International Conference on*. 2008.
22. Cuff, J.A. and G.J. Barton, *Application of multiple sequence
alignment profiles to improve protein secondary structure
prediction.* Proteins: Structure, Function, and Bioinformatics,
2000. 40(3): p. 502-511.
23. Sadasivam, G.S. 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*. 2010. ACM.
24. Alshammari, H., H. Bajwa, and J. Lee, *Hadoop Based Enhanced
Cloud Architecture*, in *ASEE*. 2014: USA.
25. Erodula, K., 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*Information Technology: New Generations (ITNG),
2011 Eighth International Conference on*. 2011. IEEE.
26. Yin, N. 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*. 1991.
27. Vora, M.N. *Hadoop-HBase for large-scale data*. in*Computer*

Science and Network Technology (ICCSNT), 2011 International
Conference on. 2011. IEEE.
28. Noll, M.G., *Running hadoop on ubuntulinux (multi-node cluster).*
Apr-2013.[Online]. Available: http://www.michaelnoll.com/tutorials/running-hadoop-on-ubuntu-linuxmulti-nodecluster/ 2011.
29. Herodotou, H. and S. Babu, *Profiling, what-if analysis, and costbased*
optimization of MapReduce programs. Proceedings of the VLDB Endowment, 2011. 4(11): p. 1111-1122.
30. Tiwari, N., et al., *Classification Framework of MapReduce*
*Scheduling Algorithms.*ACM Comput.Surv., 2015. 47(3): p. 1-38.
31. Zaharia, M., et al., *Job scheduling for multi-user mapreduce*
clusters. EECS Department, University of California, Berkeley,
Tech. Rep. UCB/EECS-2009-55, 2009.
32. Gu, R., et al., *SHadoop: Improving MapReduce performance by*
optimizing job execution mechanism in Hadoop clusters. Journal of
Parallel and Distributed Computing, 2014. 74(3): p. 2166-2179.
33. Qi, C., L. Cheng, and X. Zhen, *Improving MapReduce Performance Using Smart Speculative Execution Strategy.*
Computers, IEEE Transactions on, 2014. 63(4): p. 954-967.
34. Jinshuang, Y., et al. *Performance Optimization for Short*
MapReduce Job Execution in Hadoop.in*Cloud and Green Computing (CGC), 2012 Second International Conference on*.
2012.
35. Apache, *Centralized Cache Management in HDFS.* Update date
2014.
36. Zhang, S., et al. *Accelerating MapReduce with distributed memory*
cache. in*Parallel and Distributed Systems (ICPADS), 2009 15th*
International Conference on. 2009. IEEE.
37. Zhang, J., et al., *A Distributed Cache for Hadoop Distributed File*
System in Real-Time Cloud Services, in *Proceedings of the 2012*
ACM/IEEE 13th International Conference on Grid Computing.
2012, IEEE Computer Society. p. 12-21.
38. Longbin, L., et al. *ShmStreaming: A Shared Memory Approach for*
Improving Hadoop Streaming Performance.in*Advanced Information Networking and Applications (AINA), 2013 IEEE 27th*
International Conference on. 2013.
39. Palanisamy, B., et al. *Purlieus: locality-aware resource allocation*
forMapReduce in a cloud. in*Proceedings of 2011 International*

Conference for High Performance Computing, Networking,
Storage and Analysis. 2011. ACM.
40. Xiao, Y. and H. Bo. *Bi-Hadoop: Extending Hadoop to Improve*
Support for Binary-Input Applications. in*Cluster, Cloud and Grid*
Computing (CCGrid), 2013 13th IEEE/ACM International
Symposium on. 2013.
41. Nishanth, S., et al. *CoHadoop++: A load balanced data colocation*
inHadoop Distributed File System. in*Advanced Computing (ICoAC), 2013 Fifth International Conference on*.
2013.
42. Jian, T., et al. *Improving ReduceTask data locality for sequential*
MapReduce jobs.in*INFOCOM, 2013 Proceedings IEEE*. 2013.
43. Xuhui, L., et al. *Implementing WebGIS on Hadoop: A case study of*
improving small file I/O performance on HDFS. in*Cluster*
Computing and Workshops, 2009.CLUSTER '09. IEEE International Conference on. 2009.
44. Xie, J., et al. *Improving mapreduce performance through data*
placement in heterogeneous hadoop clusters. in*Parallel &*

Distributed Processing, Workshops and Phd Forum (IPDPSW),
2010 IEEE International Symposium on. 2010. IEEE.
45. Elghandour, I. and A. Aboulnaga, *ReStore: reusing results of*
*MapReduce jobs.*Proc. VLDB Endow., 2012. 5(6): p. 586-597.

HamoudAlshammari (First Author)
received a BS in Computer Information
Systems from King Saud University,
Saudi Arabia in 2002. He received his
first MS degree in Business MBA from
Yarmok University, Jordan. Then, he
received the second MS degree in
Computer Science from University of Bridgeport, CT-USA.
He is doing his Ph.D. in Computer Science and Engineering
at University of Bridgeport, CT-USA. Alshammari is doing
his research in BigData and HadoopMapReduce
performance. He is also has interesting in data analysis. Mr.
Alshammari is a member in Upsilon Pi Epsilon Honor Society.
Dr.Jeongkyu Lee received a B.S. from
Sungkyunkwan University in Mathematic
Education and an M.S. from Sogang
University in Computer Science, both of
Seoul, Korea in 1996 and 2001,
respectively. He worked as a database
administrator for seven years with
companies including IBM. In fall 2002, he
entered the Doctoral program in Computer Science and
Engineering at the University of Texas at Arlington. After
he received Ph.D. degree in summer 2006, he joined the
Department of Computer Sciences and Engineering at
University of Bridgeport, CT as an assistant professor, and
he has been an associated professor since 2012.
His research interest is in the multimedia database
management and big data analytics. His work also includes
techniques for multimedia data mining, video processing,
multimedia ontology, and medical imaging. He is a
program committee of IEEE International Symposium on
Multimedia, ACM Symposium on Applied Computing
(SAC), and International Resources Management
Association (IRMA). He is a president of KOCSEA (Korean
Computer Scientists and Engineers Association in America).
Dr. Hassan Bajwa received his BSc
degree in Electrical Engineering from
NYU Polytechnic University of New York
in 1998. From 1998 to 2001 he worked for
Software Spectrum. He received his MS
from the City College of New York in
2003, and his Doctorate in Electrical
Engineering from City University of New
York in 2007.Currently he is an Associate Professor of
Electrical Engineering at the University of Bridgeport. His
research interests include modeling and simulation of
Nano-electronic architectures, low power sensor networks,
flexible electronics, bioelectronics, and Bioinformatics.
2016
2016.