Evaluating the Scalability of Hadoop in a real and virtual environment on a huge volume of unstructured data

by

Md. Siddiq Bin Nur

POST GRADUATE DIPLOMA IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology (IICT)
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY (BUET)

March 2017

The project report titled ―Evaluating the Scalability of Hadoop in a real and virtual environment on a huge volume of unstructured Data" Submitted by Md. Siddiq Bin Nur, Roll No : 0413311014, Session : April/2013, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Post Graduate Diploma in ICT held on 27 March, 2017.

CANDIDATE'S DECLARATION

It is hereby declared that this report or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Md. Siddiq Bin Nur
Roll No : 0413311014
Institute of Information and Communication Technology
BUET, Dhaka-1000

DEDICATED

To

My Father Md. Nurul Islam and Mother Farida Nahar

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATION

IDC	International Data Corporation
HDFS	Hadoop Distributed File System
DFS	Distributed File System
CPU	Central Processing Unit
MB	Megabyte
GB	Gigabyte
TB	Terabyte
PB	Petabytes
EB	Eatabytes
ZB	Zettabytes
HTTP	Hyper Text Transfer Protocol
I/O	Input/Output
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
RAM	Random Access Memory
SSH	Secure Shell
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
Amazon EC2	Amazon Elastic Compute Cloud
API	Application Programming Interface
SNA	Shared Nothing Architecture

ACKNOWLEDGEMENT

First of all, I would like to thank Almighty Allah for given me the strength and patience for carrying out this work and to complete this project.

I would like to express my sincere gratitude to my supervisor, Dr. Md. Saiful Islam, Institute of Information and Communication Technology (IICT), University of Engineering and Technology (BUET). He has assigned me an interesting and useful topic, which has a wide range of application in real world. He has provided all sorts of support regarding the project work, including learning instruction and lab material. His proper guidance, advice, continual encouragement and active participation in this process of this work help to complete this project.

I would like to convey my thanks to Assistant Professor Dr. Hossen Asiful Mustafa, Institute of Information and Communication Technology, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh for his guidance, valuable suggestions and constant encouragement throughout the whole period of the work, which inspired and guided me in each and every step of the project to complete it successfully in time.

I would also like to thank Assistant Professor Mohammad Imam Hasan Bin Asad of Institute of Information and Communication Technology, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh for encouragement during academic classes.

I am grateful to all the teachers, officers and staffs of IICT, BUET for giving me their kind support and information during the study

I would like to thank all my friends for their continuous support and inspiration throughout the period of this project.

I would like to acknowledge the efforts given by my family members especially my parents for their continuous support and inspiration, which helped me to complete the project successfully.

ABSTRACT

Businesses across all industries, academic institutions or research organizations are gathering and storing more and more unstructured data on a daily basis. Unstructured data is being constantly generated via call center logs, emails, documents on the web, blogs, tweets, customer comments, customer reviews, and so on. Unstructured data takes a lion‘s share in digital space and approximately occupies 80% by volume compared to only 20% for structured data. Until recently, the technology didn‘t evolve to support doing much with it except storing it or analyzing it manually. While the amount of unstructured data is increasing rapidly, businesses‘ ability to summarize, understand and make sense of such data for making better business decisions become challenging. But organizations are in dire need to process and exploit unstructured data to get edge in business. Some big data tools, primarily those based on Hadoop as well as MapReduce, are designed from the ground up to manage and analyze unstructured information. In this project, an attempt is made to determine the scalability of Hadoop cluster on huge volumes of textual unstructured data for word count. For this a Hadoop cluster is established through real environment and sample data sets are analyzed by using the cluster. It is found that it significantly reduce the processing time of desired output based on Hadoop cluster size. The results show that as cluster size increases the performance gives better output in terms of task completion time.

CHAPTER 1

INTRODUCTION

1.1 Big data

Long gone are the days when global enterprises, research organizations and governments were able to handle all the data they produced and managed in a logical, mostly structured form and use relational databases and Structured Query Languages (SQL) or similar techniques to query it. Companies, programs and network sensors produce huge amount of fresh data, around 2.5 EBs (exabytes) every single day, and it is estimated that over 90% of existing data has been generated during the last two years [1-2]. Some predictions say the annual global data production in 2020 will be about 30 times faster than it was in 2010 - 35 Zettabytes (ZB)^2 per year [3]. Figure 1.1 shows the predicted production of annual data rate.

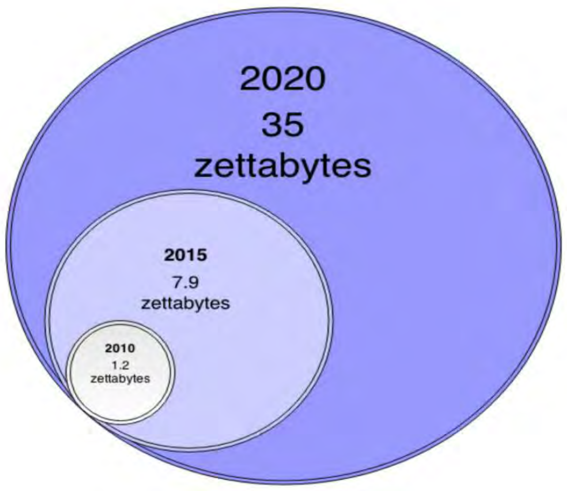

Figure 1.1: Predicted annual data production rate

Data comes from many sources like:

- ✓ The New York Stock Exchange generated 1 terabyte per day
- ✓ Facebook host 1 petabyte storage
- ✓ Ancestry.com stored 2.5 petabytes
- ✓ The Internet Archive stores 2 petabytes and generate 20 terabytes per month
- ✓ The Large Hadron Collider near Geneva, Switzerland, will produce about 15 petabytes of data per year.

1.2 Structured and unstructured data

Structured data usually consists of data that is of known format, such as numbers, dates and strings (names, addresses, clusters of words), i.e. it refers to data with a high level of organization [4]. Structured data is defined and (mostly) machine-readable, it's stored in well-defined mathematical structure and it has rules and standards. Structured data is usually used in systems that use RDBMS (Relational Database Management System) and SQL for storing and querying the data.

Essentially all the data that is not stored following a pre-defined rules or structures can be classified as unstructured data. This could be for example any binary data or textual information, that a computer program could not use. Even data that has some level of structure is treated as unstructured, if the structure does not reflect a useful schema [5]. Table 1.1 shows the comparison of structured and unstructured data.

There is an area, however, where structured and unstructured data overlap. For example unstructured data can be stored in a column of a table within a relational database. Structured data can be saved to file and stored in HDFS (Hadoop Distributed File System). This gray area is called "semi- structured data" and many times the data falls within this category instead of being clearly well defined or not defined at all. As an example, video is usually thought to be unstructured data but an image is structured. An animated image, such as a GIF is still an image, but it serves as a very low quality (and very short) audio-less video.

Structured data	**Unstructured data**
Digital images	Social media
Names, URLs, C++	Semantics
DNA	Behavior
Radio frequency spectrum	Weather
XML	
Natural languages	

Table 1.1: Examples of structured, unstructured and semi-structured data

Large-scale data is by default unstructured. It is collected from different events automatically and usually stored as-is. Most of the large-scale frameworks have high-level interfaces and underlying techniques to deal with structured data as well, but in comparison to RDBMS batch-type systems they lose in terms of performance when dealing with smaller structured-data datasets. This fact is demonstrated in the implementation, where the used datasets are relatively small and quite well structured.

Traditional approaches to information management have mostly focused on dealing with structured data. However, the majority of data produced today is in unstructured format; it is estimated that around 85-90% of all the data today is unstructured [6]. Adding this to the sheer amount of information companies are dealing with today, the management of an information system can become a pretty demanding task. Issues such as performance, scalability, replication and consistency introduce challenges that certainly already existed earlier, but previously the scope has been different. Figure 1.2 shows the different stages of data.

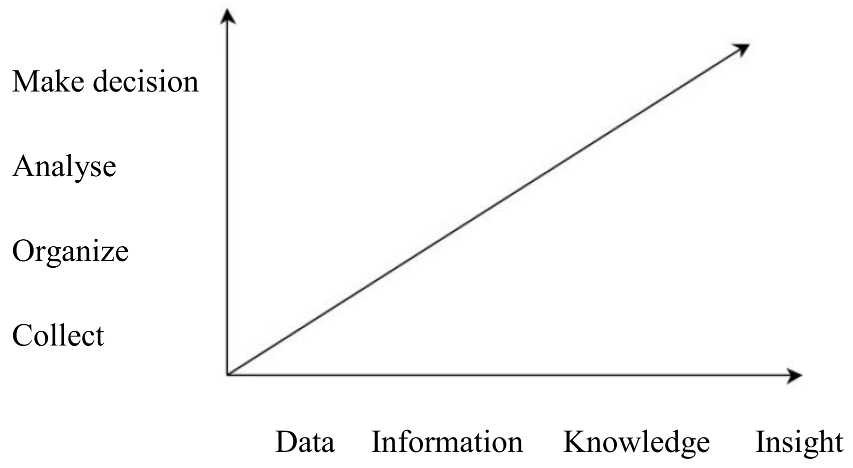

Figure 1.2: Different stages of data

Data on a large scale is not a new thing. Fields such as physics, meteorology, genomics and banking have been dealing with heaps of data and information for decades. It is only quite recently, though, that new techniques and software frameworks have made big data conceivable for the masses.

In the world of RDBMS and centralized web servers, the division between components and their role in a system was straightforward and logical. When handling unstructured datasets that are possibly many thousand or million times bigger than what we are used to when dealing with traditional, structured data, lines get a bit more blurred. Hadoop and its modules and tools take care of every step of the process, from analyzing to storing and processing and management.

1.3 History of Hadoop

Hadoop is an open source platform that makes the processing of very large sets of data relatively easy and simple. It‘s not a problem whether the data is structured, unstructed or hybrid. It evolved from the sole need to store, process and manage massive amounts of data that previous systems were not able to handle. It includes a lot of different sub-projects, or modules, of which the most common and important ones are briefly explained below. All of the projects and different modules are open-source.

Hadoop is based on Apache Lucene 6 and Apache Nutch 7, both of which are created by Doug Cutting, also the creator of Hadoop. It started as a sub-project of Lucene as an open source web search engine (Nutch). Soon after Google released Google File System (GFS) and MapReduce, Hadoop added both of them to its implentation, giving the base for the Hadoop Distributed File System (HDFS) and Hadoop's implementation of MapReduce.

1.4 Read and write data simultaneously

There are some issues to read and write data in parallel

- ✓ hardware failure,
- ✓ most analysis tasks need to combine data. Combine data from different sources can be handled by Distributed systems, but make it accurately is still challenging.

From the above case Hadoop provides a reliable shared storage and analysis system. The storage is provided by HDFS and analysis by MapReduce.

The Mapreduce approach may seem like brute-force. It‘s also a batch query processor and the ability to run an ad hoc query against your whole dataset and get the results in a reasonable time is transformative. It gives us the opportunity to innovate with data.

Seek time is increasing slowly than transfer rate. Seeking is the process of moving the disk‘s head to a particular place on the disk to read or write data. It characterizes the latency of a disk operation, where the transfer rate corresponds to a disk‘s bandwidth.

So many ways MapReduce is a complement of an RDBMS. The major differences are given Table 1.2 MapReduce is the best match for analyzing the full dataset in a batch fashion. An RDBMS is the good fit for point queries or updates. MapReduce suits application where the data is written once and read many times, whereas a relational database is good for datasets that are continually updated.

	Traditional RDBMS	**MapReduce**
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

Table 1.2 : RDBMS compared to MapReduce

MapReduce is a linearly scalable programming model. The programmer writes 2 functions

- ✓ a map function and
- ✓ a reduce function.

Each of which defines a mapping from one set of key-value pairs to another. These 2 functions are oblivious to the size of the data or the cluster that they are operating on, so they can be used unchanged for a small dataset and for a massive one. More importantly if you double the size of the input data, a job will run twice as slow. But if you also double the size of the cluster a job will run as fast as the original one. This is not generally true of SQL queries.

1.5 Objectives

With the increasing popularity of commodity computing Hadoop software framework is becoming widely used for processing large data on commodity computers. The main goal of this research project is to process large volumes of data using the Hadoop framework. To achieve the goal the following objectives are identified:

i) Understand Hadoop framework and learn to deploy a Hadoop cluster in commodity computer.

ii) Textual analysis of big volumes of data and determining the real execution time of specific word counting on various nodes based on different volumes of data.

iii) Evaluating the performance of the data processing under the proposed Hadoop and conventional framework.

1.6 Structure of the report

In Chapter 1, a big picture of big data is given and finally the objectives of the project is described. The technology for the processing of big data is discussed in Chapter 2 in detail. Implementation of the project is presented in Chapter 3. Finally conclusion and future work are described in Chapter 4.

CHAPTER 2

MAPREDUCE AND HADOOP

2.1 Introduction

MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages; in this chapter, we shall look at the same program expressed in Java, Ruby, Python, and C++. Most importantly, MapReduce programs are inherently parallel; thus putting very large scale data analysis into the hands of anyone with enough machines at their disposal. MapReduce comes with its own large datasets.

2.2 MapReduce process and function

MapReduce works by breaking the processing into two phases:

- ✓ the map phase and
- ✓ the reduce phase.

Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions:

- ✓ the map function and
- ✓ the reduce function.

Release 0.20.0 of Hadoop includes a new Java MapReduce application programming interface (API), sometimes referred to as ―Context Objects,” designed to make the API easier to evolve in the future. The new API is type-incompatible with the old; so applications need to be rewritten to take advantage of it.

There are several notable differences between the two APIs:

- ✓ The new API favors abstract classes over interfaces, since these are easier to evolve. For example, you can add a method (with a default implementation) to an abstract class without breaking old implementations of the class. In the new API, the Mapper and Reducer interfaces are now abstract classes.

- ✓ The new API is in the org.apache.hadoop.mapreduce package (and subpackages). The old API is found in org.apache.hadoop.mapred.

- ✓ The new API makes extensive use of context objects that allows the user code to communicate with the MapReduce system. The MapContext, for example, essentially unifies the role of the JobConf, the OutputCollector, and the Reporter.

- ✓ The new API supports both a ―push” and a ―pull” style of iteration. In both APIs, key-value record pairs are pushed to the mapper, but in addition, the new API allows a mapper to pull records from within the map() method. The same goes for the reducer. An example of how the ―pull” style can be useful is processing records in batches, rather than one by one.

- ✓ Configuration (see ―Configuration Management” on Installing Apache Hadoop section) has been unified. The old API has a special JobConf object for job configuration, which is an extension of Hadoop‘s vanilla Configuration object. In the new API, this distinction is dropped, so job configuration is done through a configuration.

- ✓ Job control is performed through the Job class, rather than JobClient, which no longer exists in the new API.

2.3 Some Terminology of Hadoop

- ✓ High Availability (HA) - an application‘s ability to restore itself to full operating capabilities when faced with intermittent or permanent failure because of any dependency from network through operating system to application, itself.

- ✓ NameNode (NN) - the master service or interface called by all Hadoop filesystem operations. The NameNode manages files which exist in a Hadoop system as well as where the data for those files reside.

- ✓ JobTracker (JT) - the master service or interface called by all MapReduce clients attempting to execute code and work inside a Hadoop cluster.

- ✓ Worker - A VM or physical machine running the HadoopTaskTracker to execute user code and the DataNode to store user data. Hadoop is designed to transparently handle failure of these nodes.

- ✓ Master – A Hadoop service that is critical to Hadoop‘s functionality, such as the NameNode or JobTracker. For a Hadoop cluster to be HA, these services must be made HA.

- ✓ Cold Standby – is an HA strategy where a replacement service is started only after a service configured for HA is determined to have failed.

- ✓ Hot Standby - is an HA strategy where a service configured for HA has a second version kept in lock-step with the active server. Clients can failover to this hot standby automatically w/o external hardware or software intervention.

2.4 Types of nodes that control the job execution process

There are two types of nodes that control the job execution process:

- ✓ a jobtracker and
- ✓ a number of tasktrackers.

The jobtracker coordinates all the jobs which running on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a tasks fails, the jobtracker can reschedule it on a different tasktracker.

Hadoop divides the input to a MapReduce job into fixed size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user defined map function for each record in the split. Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if we are processing the splits in parallel, the processing is better load balanced if the splits are small, because a faster machine will be able to process proportionally more splits over the course of the job than a slower machine. Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing increases as the splits become more fine grained.

On the other words, if splits are become too small, then the overhead of managing the splits and of map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of a HDFS block, 64 MB by default, although this can be changed for the cluster (for all newly created files), or specified when each file is created.

Hadoop does its best to run the map task on a node where the input data resides in HDFS. This is called the data locality optimization. It should now be clear why the optimal split size is the same as the block size: it is the largest size of input that can be guaranteed to be stored on a single node. If the split spanned two blocks, it would be unlikely that any HDFS node stored both blocks, so some of the split would have to be transferred across the network to the node running the map task, which is clearly less efficient than running the whole map task using local data.

2.5 Map tasks and Reduce Tasks

That Map tasks write their output to local disk not Hadoop Distributed File System (HDFS), The main reason are :

- ✓ Map output is intermediate output,
- ✓ Processed by reduce tasks to produce the final output.
- ✓ Once the job is complete the map output can be thrown away

Storing Map output in HDFS with replication would be overkill. If the node running the map task fails before the map output has been consumed by the reduce task, the Hadoop will automatically rerun the map task on another node to recreate the map output. Reduce tasks doesn‘t have the advantage of data locality – the input to a single reduce task is normally the output from all mappers.

2.6 Single reduce task

The whole data flow with a single reduce task is illustrated in Figure 2.1

- ✓ The dotted boxes indicate nodes,
- ✓ the light arrows show data transfers on a node, and
- ✓ the heavy arrows show data transfers between nodes.

The number of reduce tasks is not governed by the size of the input, but is specified independently.

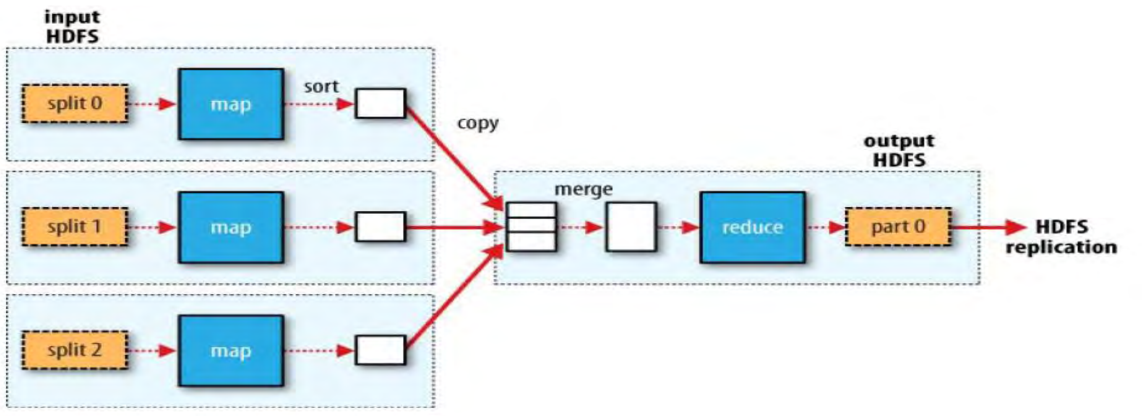

Figure 2.1: MapReduce data flow with a single reduce task

2.7 Multiple reduce tasks

When there are more than one reducers, the map tasks partition their output each creating one partition for each reduce task. There can be many keys in each partition, but the records for every key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default practitioner —which buckets keys using a hash function—works very well. The data flow for the general case of multiple reduce tasks is illustrated in Figure 2.2, This diagram makes it clear

- ✓ Why the data flow between map and reduce tasks is colloquially known as ―the shuffle," as each reduce task is fed by many map tasks.

- ✓ The shuffle is more complicated than this diagram suggests, and tuning it can have a big impact on job execution time.

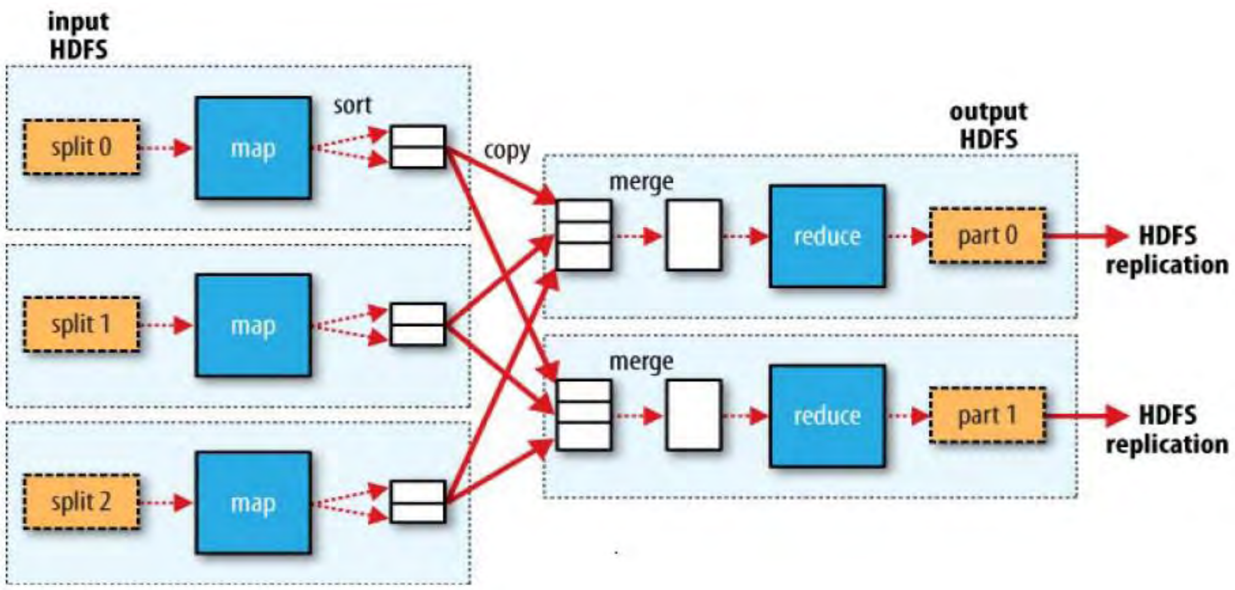

Figure 2.2: MapReduce data flow with multiple reduce tasks

2.8 Zero reduce tasks

Finally, it's also possible to have zero reduce tasks.

- ✓ This can be appropriate when you don't need the shuffle since the processing can be carried out entirely in parallel

- ✓ In this case, the only off-node data transfer is when the map tasks write to HDFS, Figure 2.3

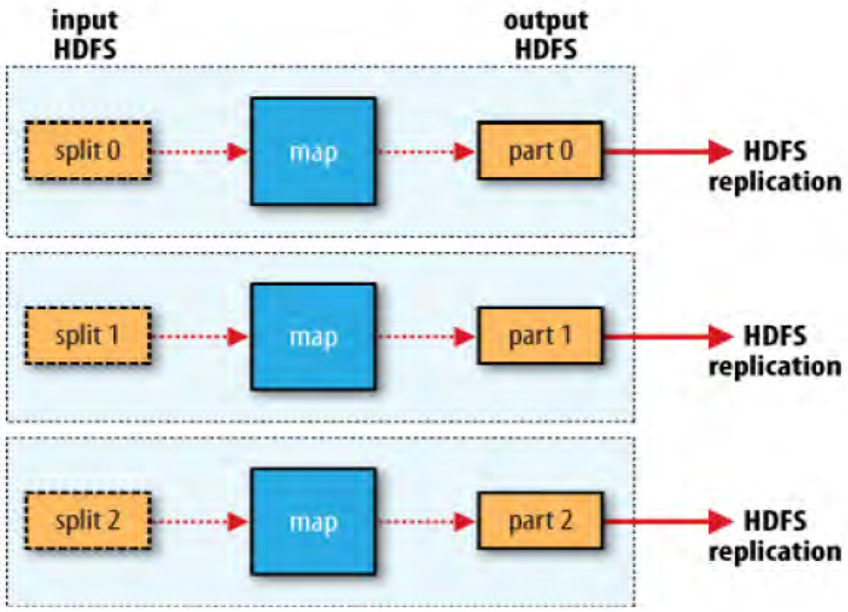

Figure 2.3: MapReduce data flow with no reduce tasks

2.9 Combiner Functions and Hadoop Streaming

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a combiner function to be run on the map output—the combiner function's output forms the input to the reduce function. Since the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call it for a particular map output record, if at all. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer.

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. Hadoop Streaming uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program.

Streaming is naturally suited for text processing (although as of version 0.21.0 it can handle binary streams, too), and when used in text mode, it has a line-oriented view of data.

- ✓ Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output.

- ✓ A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format—a tab-separated key-value pair—passed over standard input. The reduce function reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

2.10 The Hadoop Distributed File system (HDFS)

When a dataset becomes larger the storage capacity of a single physical machine, it becomes necessary to partition it across a number of separate machines. Distributed file systems is file systems that manage the storage across a network of machines. Mainly they are network based, all the complications of network programming kick in, thus making distributed file systems more complex than regular disk file systems.

HDFS is Hadoop's flag ship file system, but Hadoop actually has a general-purpose file system abstraction, so we'll see along the way how Hadoop integrates with other storage systems (such as the local file system)

HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters on commodity hardware. Let's see this statement in more detail:

a) Very large files

―Very large" in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.

b) Streaming data access

HDFS is built around the idea that the most efficient data processing pattern is a write once, read many times pattern. A dataset is typically generated or copied from source, then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

c) Commodity hardware

Hadoop doesn't require expensive, highly reliable hardware to run on. It's designed to run on clusters of commodity hardware (commonly available hardware available from multiple vendors) for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure. It is also worth examining the applications for which using HDFS does not work so well. While this may change in the future, these are areas where HDFS is not a good fit today.

d) Low-latency data access

Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency.

e) Lots of small files

Since the name node holds file system metadata in memory, the limit to the number of files in a file system is governed by the amount of memory on the name node. As a rule of thumb, each file, directory, and block takes about 150 bytes.

2.11 HDFS architecture

When a client writes data in a HDFS file, it gets replicated as a part of the write operation in a form of data pipeline. Data is first written in a local file, which eventually grows to a full block of data. It is depicted in Figure 2.4

When the block is ready, or full, the client gets the list of assigned DataNodes for replicas of the block from the NameNode. The first DataNode on the list receives the client's data blocks and saves them to its local filesystem. Then the DataNode sends the blocks to the next DataNode on the list and this goes on for as long as there are receiving DataNodes on the list. The last DataNode saves the data, but does not forward it anymore.

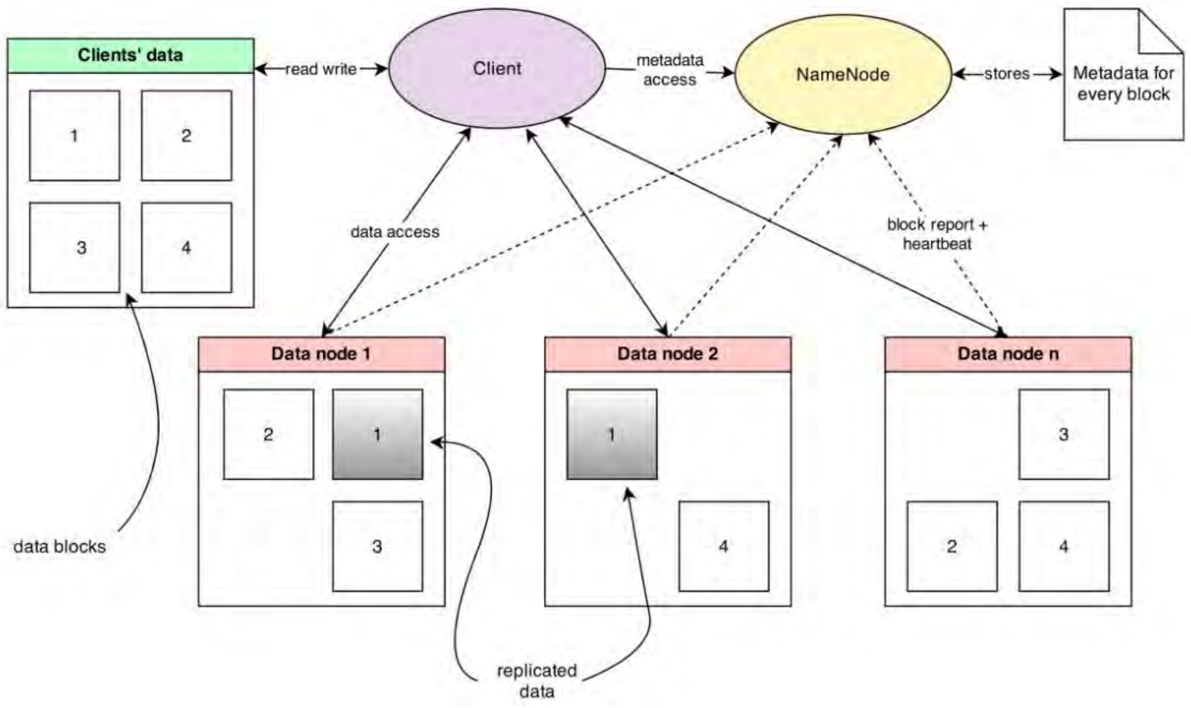

Figure 2.4:HDFS architecture

HDFS is also aware of how different machines are distributed between the computer racks. This ensures good performance, since performance among machines within a rack tend to be better than the performance among the machines on separate racks due to network latency. Also, it guarantees that the data is not lost in case a whole rack gets destroyed. HDFS has to be fine-tuned to work fluently. Some of the settings, such as block size and the replication factor, can be specified for a file separately during its creation. Otherwise values specified in Hadoop configuration are used.

2.12 HDFS Concepts: Blocks

A disk has a block size, which is the minimum amount of data that it can read or write, while disk blocks are normally 512 bytes. File systems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. File system blocks are typically a few kilobytes in size. This is generally transparent to the file system user who is simply reading or writing a file—of whatever length.

File system maintenance tools: *df* and *fsck* that operate on the file system block level. HDFS too has the concept of a block, but it is a much larger unit—64 MB by default. Like in a file system for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a file system for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block‘s worth of underlying storage.

Having a block abstraction for a distributed file system brings several benefits given below:

- ✓ The first benefit is the most perceived: a file can be larger than any single disk in the network. There‘s nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster. In fact, it would be possible, if unusual; to store a single file on an HDFS cluster whose blocks filled all the disks in the cluster.

- ✓ Second, making the unit of abstraction a block rather than a file simplifies the storage subsystem. Simplicity is something to strive for all in all systems, but is important for a distributed system in which the failure modes are so varied. The storage sub system deals with blocks, simplifying storage management (since blocks are a fixed size, it is easy to calculate how many can be stored on a given disk), and eliminating metadata concerns

- ✓ Third, blocks fit well with replication for providing fault tolerance and availability. To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines (typically three). If a block becomes unavailable, a copy can be read from another location in a way that is transparent to the client. A block that is no longer available due to corruption or machine failure can be replicated from their alternative locations to other live machines to bring the replication factor back to the normal level. Similarly, some applications may choose to set a high replication factor for the blocks in a popular file to spread the read load on the cluster.

2.13 Namenodes and Datanodes

Namenodes:

A HDFS cluster has two types of node operating in a master-worker pattern:

- ✓ a name-node (the master) and
- ✓ a number of data nodes (workers).

The namenode manages the file system namespace. It maintains the file system tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files:

- ✓ the namespace image and
- ✓ the edit log.

The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.

A client accesses the file system on behalf of the user by communicating with the namenode and datanodes. The client presents a POSIX-like file system interface, so the user code does not need to know about the namenode and datanode to function.

Datanodes:
Datanodes are the work like horses of the file system. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing.

Without the namenode, the file system cannot be used. In fact, if the machine running the namenode were obliterated, all the files on the file system would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes. For this reason, it is important to make the namenode resilient to failure, and Hadoop provides two mechanisms for this.

- ✓ The first way is to back up the files that make up the persistent state of the file system metadata. Hadoop can be configured so that the namenode writes its persistent state to multiple file systems. These writes are synchronous and atomic. The usual configuration choice is to write to local disk as well as a remote NFS mount.

- ✓ It is also possible to run a secondary namenode, which despite its name does not act as a namenode. Its main role is to periodically merge the namespace imagewith the edit log to prevent the edit log from becoming too large. The secondary namenode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the namenode to perform the merge. It keeps a copy of the merged name-space image, which can be used in the event of the namenode failing.

- ✓ However, the state of the secondary namenode lags that of the primary, so in the event of total failure of the primary data, loss is almost guaranteed. The usual course of action in this case is to copy the namenode‘s metadata files that are on NFS to the secondary and run it as the new primary.

2.14 Basic File system Operations

The file system is ready to be used, and we can do all of the usual file system operations such as:

- ✓ reading files,
- ✓ creating directories,
- ✓ moving files,
- ✓ deleting data, and
- ✓ listing directories.

2.15 Hadoop File systems

Hadoop has an abstract notion of file system, of which HDFS is just one implementation. The Java abstract class org.apache.hadoop.fs. File System represents a filesystem in Hadoop, and there are several concrete implementations, which are described in Table 2.1

Filesystem	**URI scheme**	**Java implementation (under org.apache.hadoop)**	**Description**
Local	file	fs.LocalFileSystem	A filesystem for a locally connected disk with client-side checksums. Use RawLocalFileSys tem for a local filesystem with no checksums. See ―LocalFileSystem" on page 76.
HDFS	hdfs	hdfs.DistributedFileSystem	Hadoop‘s distributed filesystem. HDFS is designed to work efficiently in conjunction with Map-Reduce.
HFTP	hftp	hdfs.HftpFileSystem	A filesystem providing read-only access to HDFS over HTTP. (Despite its name, HFTP has no connection with FTP.) Often used with distcp (―Parallel Copying with distcp" on page 70) to copy data between HDFS clusters running different versions.
HSFTP	hsftp	hdfs.HsftpFileSystem	A filesystem providing read-only access to HDFS over HTTPS. (Again, this has no connection with FTP.)
HAR	har	fs.HarFileSystem	A filesystem layered on another filesystem for archiving files. Hadoop Archives are typically used for archiving files in HDFS to reduce the namenode‘s memory usage. See ―Hadoop Archives" on page 71.
KFS (Cloud-Store)	kfs	fs.kfs.KosmosFileSystem	CloudStore (formerly Kosmos filesystem) is a distributed filesystem like HDFS or Google‘s GFS, written in C++. Find more information about it at http://kosmosfs .sourceforge.net/.
FTP	ftp	fs.ftp.FTPFileSystem	A filesystem backed by an FTP server.

S3 (native)	s3n	fs.s3native.NativeS3FileSystem	A filesystem backed by Amazon S3. See http://wiki.apache.org/hadoop/AmazonS3.
S3 (block-based)	s3	fs.s3.S3FileSystem	A filesystem backed by Amazon S3, which stores files in blocks (much like HDFS) to overcome S3's 5 GB file size limit.

Table 2.1: Hadoop Filesystems

Hadoop provides many interfaces to its file systems, and it generally uses the URI scheme to pick the correct file system instance to communicate with.

2.16 HDFS Interfaces

There are two interfaces that are specific to HDFS:

HTTP

HDFS defines a read only interface for retrieving directory listings and data over HTTP. Directory listings are served by the namenode's embedded web server(which runs on port 50070) in XML format, while file data is streamed fromdatanodes by their web servers (running on port 50075).

Use HTTP

This protocol is not tied to a specific HDFS version, making it possible to write clients that can use HTTP to read data from HDFS clusters that run different versions of Hadoop. HftpFile System is a such a client: it is a Hadoop file system that talks to HDFS over HTTP (hsftpFileSystem is the HTTPS variant).

FTP

Although not complete at the time of this writing (https://issues.apache.org/jira/browse/HADOOP-3199), there is an FTP interface to HDFS, which permits the use of the FTP protocol to interact with HDFS.

Use FTP

This interface is a convenient way to transfer data into and out of HDFS using existing FTP clients. The FTP interface to HDFS is not to be confused with FTP File System, which exposes any FTP server as a Hadoop filesystem.

2.17 File Read

To get an idea of how data flows between the client interacting with HDFS, the namenode and the datanode, consider Figure 2.5, which shows the main sequence of eventswhen reading a file.

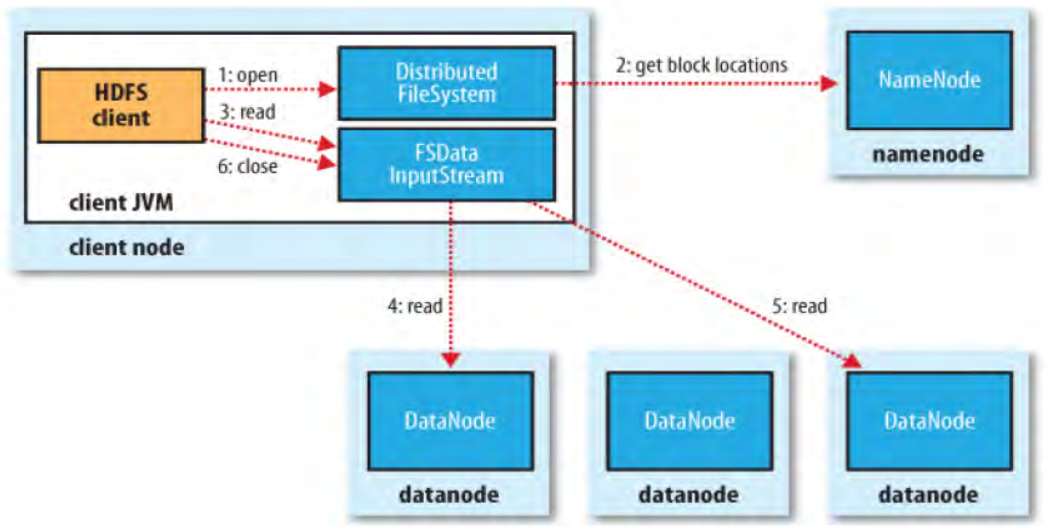

Figure 2.5: A client reading data from HDFS

- ✓ At first the client opens the file it wishes to read by calling open() on the FileSystem object, which for HDFS is an instance of DistributedFileSystem (step 1 in Figure 2.5)
- ✓ Distributed File System calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file (step 2).
- ✓ For each block, the namenode returns the addresses of the datanodes that have a copy of that block.
- ✓ Furthermore, the datanodes are sorted according to their proximity to the client
- ✓ If the client is itself a datanode (in the case of a MapReduce task, for instance), then it will read from the local datanode. The Distributed File System returns a FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the datanode and namenode I/O.
- ✓ The client then calls read( ) on the stream (step 3). DFSInputStream, which has stored the datanode addresses for the first few blocks in the file, then connects to the first (closest) datanode for the first block in the file.
- ✓ Data is streamed from the datanode back to the client, which calls read( ) repeatedly on the stream (step 4).
- ✓ When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block (step 5). This happens transparently to the client, which from its point of view is just reading a continuous stream.

- ✓ Blocks are read in order with the DFSInputStream opening new connections to datanodes as the client reads through the stream. It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls close( ) on the FSDataInputStream (step 6).

2.18 File Write

Now we‘ll look at how files are written to HDFS. Although quite detailed, it is instructive to understand the data flow since it clarifies HDFS‘s coherency model. The case we‘re going to consider is the case of creating a new file, consider Figure 2.6, writing data to it, and then closing the file.

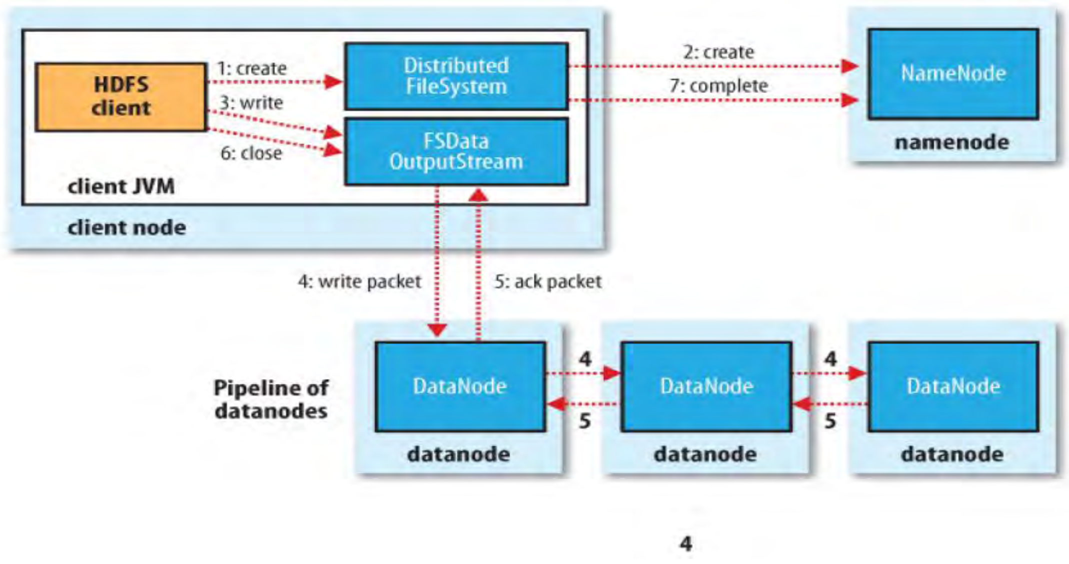

Figure 2.6: A client writing data to HDFS

- ✓ The client creates the file by calling create() on DistributedFileSystem (step 1 in Figure 2.6)
- ✓ DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem‘s namespace, with no blocks associated with it (step 2).
- ✓ The namenode performs various checks to make sure the file doesn‘t already exist, and that the client has the right permissions to create the file. If these checks pass, the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an IOException. The DistributedFileSystem returns a FSDataOutputStream for the client to start writing data to. Just as in the read case,

FSDataOutputStream wraps a DFSOutput Stream, which handles communication with the datanodes and namenode.

- ✓ As the client writes data (step 3), DFSOutputStream splits it into packets, which it writesto an internal queue, called the data queue. The data queue is consumed by the DataStreamer, whose responsibility it is to ask the namenode to allocate new blocks bypicking a list of suitable datanodes to store the replicas. The list of datanodes forms apipeline—we‘ll assume the replication level is 3, so there are three nodes in the pipeline.The DataStreamer streams the packets to the first datanode in the pipeline, which storesthe packet and forwards it to the second datanode in the pipeline.
- ✓ Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipe line (step 4).
- ✓ DFSOutputStream also maintains an internal queue of packets that are waiting to beacknowledged by datanodes, called the ack queue. A packet is removed from the ackqueue only when it has been acknowledged by all the datanodes in the pipeline (step 5).
- ✓ When the client has finished writing data it calls close() on the stream (step 6).
- ✓ This action flushes all the remaining packets to the datanode pipeline and waits for ac-knowledgments before contacting the namenode to signal that the file is complete (step7). The namenode already knows which blocks the file is made up of (via DataStreamer asking for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully.

2.19 Summary

We have discussed the Hadoop and MapReduce internal architecture and their detail working processes. Here the key points are:

- ✓ MapReduce is ideal for operating on very large, flat (unstructured) datasets and perform trivially parallel operations on them
- ✓ Hadoop jobs go through a map stage and a reduce stage where the mapper transforms the raw input data into key-value pairs where multiple values for the same key may occur
- ✓ The reducer transforms all of the key-value pairs sharing a common key into a single key with a single value.

CHAPTER 3

IMPLEMENTATION

In this project, we have created a virtual environment using Hadoop technology to process large amount of unstructured data. Here the processing is to count the number of various words occur in a text. In the next sections, the computing environment and result are given.

3.1 Running Hadoop in a Virtual Environment

Hadoop was set up on a cluster of virtual machines in VMware Workstation. The computer had 12 GB of RAM with a Q6700 Core 2 Quad Core processor and was running Windows Vista.

3.1.1 Virtual Machine Configuration

Operating System	Ubuntu 15.04 Server
Processor	3 GHz
RAM	2 GB
Hard Drive	500 GB

Table 3.1: Configuration of each virtual machine

3.1.2 Setting up the Environment

To create the virtual environment a Ubuntu 15.04 virtual machine was set up and Hadoop was installed. The virtual machine was then cloned 3 times to create the remaining machines in the environment.

3.2 Running Hadoop in a Real Environment

This section outlines how Hadoop was run in a real environment.

3.2.1 Real Machine Configuration

Operating System	Ubuntu 15.04 Server
Processor	3 GHz
RAM	2 GB
Hard Drive	500 GB

Table 3.2: Configuration of each real machines

3.2.2 Setting up the Environment

Installation is straightforward. Java 6 is a prerequisite. Please refer to the installation instructions in Chapter 4.

3.3 Creating a Performance Metric for Measuring the Scalability Hadoop

Hadoop‘s example MapReduce Java program was used to measure the performance of each cluster. The wordcount program counts the occurrence of each word in the input. To create a dataset for the word count program to use, totaling 977.1 MB. The main metric used to judge performance was the time it took each cluster size to run word count on the collection of unstructured datasets.

3.4 Hadoop properties

System logfiles produced by Hadoop are stored in $HADOOP_INSTALL/logs by default. This can be changed using the HADOOP_LOG_DIR setting in hadoop-env.sh. It‘s a good idea to change this so that logfiles are kept out of the directory that Hadoop is installed in, since this keeps logfiles in one place even after the installation directory changes after an upgrade. A common choice is /var/log/hadoop, set by including the following line in hadoop-env.sh:

```
export HADOOP_LOG_DIR=/var/log/hadoop
```

3.5 MapReduce

MapReduce jobs are controlled by a software daemon known as the JobTracker. The JobTracker resides on a master node

- ✓ Clients submit MapReduce jobs to the JobTracker
- ✓ The JobTracker assigns Map and Reduce tasks to other nodes on the cluster
- ✓ These nodes each run a software daemon known as the TaskTracker
- ✓ The TaskTracker is responsible for actually instantiating the Map or Reduce task, and reporting progress back to the JobTracker.

3.6 The Five Hadoop Daemon

Hadoop is comprised of five separate daemons

i) NameNode

- ✓ Holds the metadata for HDFS

ii) Secondary NameNode

- ✓ Performs housekeeping functions for the NameNode
- ✓ Is not a backup or hot standby for the NameNode!

iii) DataNode

- ✓ Stores actual HDFS data blocks

iv) JobTracker

- ✓ Manages MapReduce jobs, distributes individual tasks to machines

v) TaskTracker

- ✓ Instantiates and monitors individual Map and Reduce tasks

3.7 The Scalability of Hadoop

As shown in Figure 3.1, The results from the experiment showed that adding more nodes to the cluster greatly decreased the time required to run the word count program in both the virtual and real environments. The graph of the real and virtual runtime both begin to show signs of diminishing returns. This suggests that at some point adding more nodes to the cluster will not improve the runtime. In the context of what Hadoop was designed for, the clusters and data set used in the experiment are both considered small. Hadoop was built to handle much larger data sets running on clusters with many more nodes.

Given the relatively under power of the machines used in the real cluster the results were fairly significant. Using 3 nodes to run the word count program nearly reduced the runtime by 78% when compared to using 1 node. Assuming that this trend could be achieved in other MapReduce programs, a job that would take 30 days to compute on a single machine could be executed in about 7 days on the cluster that was configured. The additions of more machines in the cluster could lead to an even greater reduction in runtime.

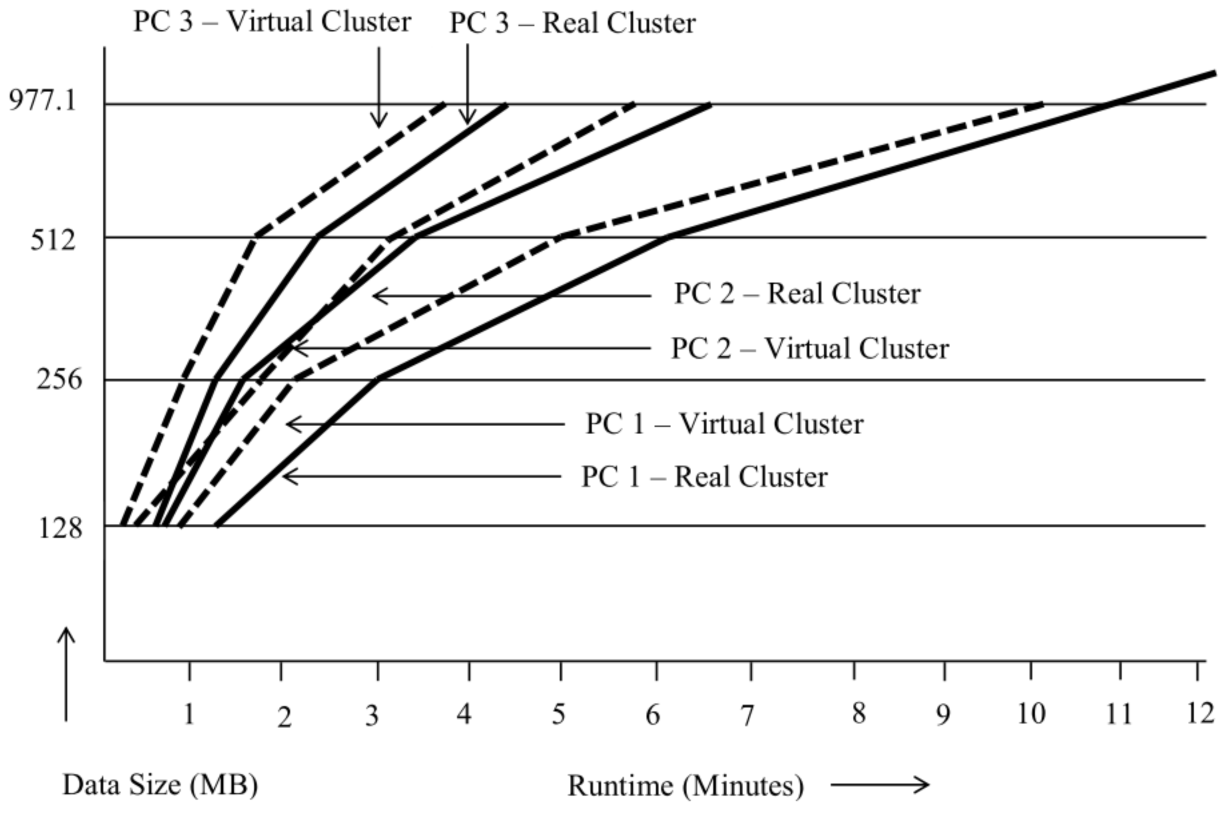

Figure 3.1: Runtime of word count with increasing cluster sizes (Data vs Time)

As shown in Figure 3.2, The curve for the virtual clusters has an interesting shape. The results showed that as the cluster size increased the runtime continued to decrease. However, the results of diminishing returns can be clearly seen as a result of the unavailability of free resources. Running 3 virtual machines put a considerable load on the host computer running the virtualization software and pushed the CPU utilization to 100%. Perhaps the most interesting results from the study is that adding more virtual machines decreased the time required to run word count on the data set. This indicates that the use of virtualization helped better utilize the resources of the host computer.

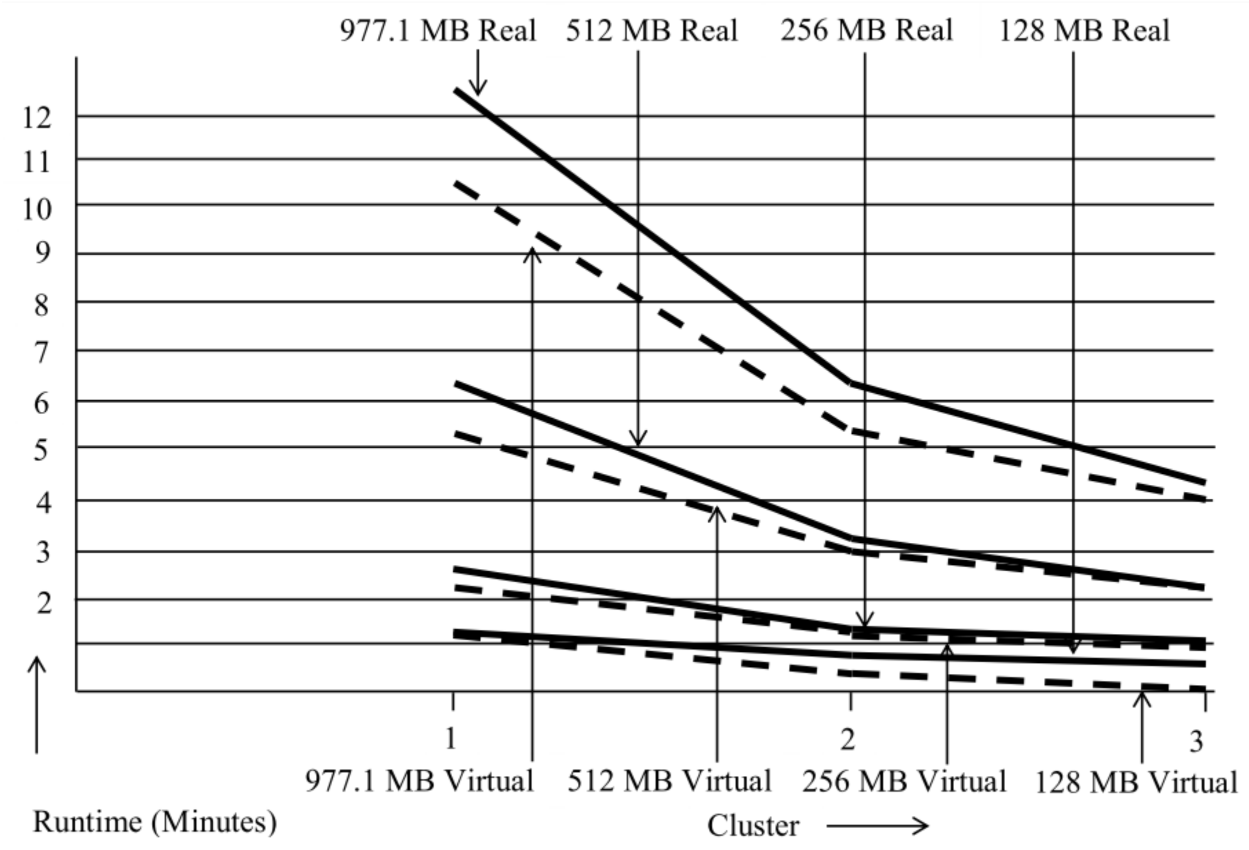

Figure 3.2: Runtime of word count with increasing cluster sizes (Time vs Cluster)

Data Size (MB)	PC 1 (Time)	PC 2 (Time)	PC 3 (Time)
977.1	12.37	6.26	4.37
512	6.10	3.21	2.21
256	2.51	1.40	1.13
128	1.18	0.51	0.41

Table 3.3: Results from running word count on various cluster sizes (Real cluster)

Data Size (MB)	PC 1 (Time)	PC 2 (Time)	PC 3 (Time)
977.1	10.17	5.56	4.14
512	5.03	3.11	2.13
256	2.17	1.29	1.01
128	1.03	0.21	0.11

Table 3.4: Results from running word count on various cluster sizes (Virtual cluster)

3.8 Summary

In this chapter, we have discussed the creation of computing environment and result. During setup and configuration, it is unique experience for us. We have spent hours and days on configuration the cluster, from which about one third is spent going through learning material, make it quite clear that a Hadoop cluster might be trivial to set up, but getting it stable and running without problems involves thorough knowledge of different parts of the system, like root permissions and full administrative rights to the cluster. Finally we ran out system successfully.

CHAPTER 4

CONCLUSION

4.1 Summary

The work of this particular project gives directions and ideas as to what kind of solutions and systems could be used for the processing of large volume of unstructured data. Usually data intensive industry get experiences for more unstructured data on daily basis as its volume near about 80% and manually handle this data is so complex. For making positive feedback and policy decision from this type data is essential and challenging. Hadoop is the best match to analyze the unstructured data. In this project, we have successfully determined the scalability of Hadoop on huge volumes of textual information.

The results obtained from this project clearly indicate that for time efficient and scalable data processing Hadoop technology can be used. It will be helpful for those organizations who derive and manages business values from big volumes of data. For example, health care industry can use Hadoop for analyzing volumes of e-health records, clinical outcomes and treatment results. It can also be useful for image analysis, financial analysis and social network analysis which involves big data and complex algorithms.

4.2 Future work

There is a lot of future work related to this project. The potential research can be broken up into two main areas – experimentation and exploration.

In the experimentation section, a more comprehensive performance study can be done. Hadoop carries many parameters that need to customize as a result, it could potentially increase the performance of the MapReduce process. Experiments could also be conducted with bigger clusters and more demanding MapReduce tasks that require much bigger data sets to get result.

The following provides the key areas of future work:

- ✓ Effect on the performance of the software by implementing the Map Reduce programming model using Java programming language and not using the python programming language and the Hadoop streaming utility of Hadoop.

- ✓ It could demonstrate on how Hadoop‘s MapReduce framework could be extended to work with big volume of textual data for textual processing.

- ✓ It could execute log file implementation for MapReduce Framework for difference volume of data.

- ✓ MapReduce program may be implemented on Windows System.

4.3 Installing Apache Hadoop

This section gives direction about how to install Hadoop from Apache Hadoop distribution. And let you know about the background knowledge to cover the things you need to consider when setting up Hadoop.

To **avoide** the complexity of installing and maintaining the same software on each node, it is also normal to use an automated installation method like Red Hat Linux‘s Kickstart or Debian‘s Fully Automatic Installation. These tools allow you to automate the operating system installation by recording the answers to questions that are asked during the installation process, as well as which packages to install. Crucially, they also provide hooks to run scripts at the end of the process, which are invaluable for doing final system tweaks and customization that is not covered by the standard installer.

The following sections describe the customizations that are needed to run Hadoop.
These should all be added to the installation script.

Installing Java

Java 6 or later version is required to run Hadoop. However the latest stable Sun JDK is the preferred option, although Java distributions from other vendors may work too. The following command confirms that Java was installed correctly:

```
% java -version
java version "1.6.0_12"
Java(TM) SE Runtime Environment (build 1.6.0_12-b04)
Java HotSpot(TM) 64-Bit Server VM (build 11.2-b01, mixed mode)
```

Creating a Hadoop User

It‘s good practice to create a dedicated Hadoop user account to separate the Hadoop installation from other services running on the same machine.

Some cluster administrators choose to make this user‘s home directory an NFS-mounted drive, to aid with SSH key distribution (see the following discussion). The NFS server is typically outside the Hadoop cluster. If you use NFS, it is worth considering autofs, which allows you to mount the NFS filesystem on demand, when the system accesses it. Autofs provides some protection against the NFS server failing, and also allows you to use replicated filesystems for failover. There are other NFS gotchas to watch out for, such as synchronizing UIDs and GIDs.

For help setting up NFS on Linux, refer to the HOWTO at http://nfs.sourceforge.net/nfs-howto/index.html.

Installing Hadoop

Download Hadoop from the Apache Hadoop releases page (http://hadoop.apache.org/core/releases.html), and unpack the contents of the distribution in a sensible location, such as /usr/local (/opt is another standard choice). Note that Hadoop is not installed in the hadoop user‘s home directory, as that may be an NFS-mounted directory.

```
% cd /usr/local
% sudo tar xzf hadoop-x.y.z.tar.gz
```

We also need to change the owner of the Hadoop files to be the hadoop user and group:

```
% sudo chown -R hadoop:hadoop hadoop-x.y.z
```

Update $HOME /.bashrc

.bashrc file will be located in every users home directory. You should edit the file located in /home/hadoop/ Edit the .bashrc file as follows,

```
#HADOOP VARIABLES START
export JAVA_HOME= /usr/lib/jvm/java-6-sun
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib"

# Some convenient aliases and functions for running Hadoop-related commands
unalias fs &> /dev/null
alias fs="hadoop fs"
```

```
unalias hls &> /dev/null
alias hls="fs -ls"


#HADOOP VARIABLES END
```

Testing the Installation

Once you‘ve created the installation file, you are ready to test it by installing it on the machines in your cluster. This will probably take a few iterations as you discover kinks in the install. When it‘s working, you can proceed to configure Hadoop and give it a test run. This process is documented in the following sections.

SSH Configuration

The Hadoop control scripts rely on SSH to perform cluster-wide operations. For example, there is a script for stopping and starting all the daemons in the cluster. Note that the control scripts are optional—cluster-wide operations can be performed by other mechanisms, too (such as a distributed shell).

To work seamlessly, SSH needs to be set up to allow password-less login for the hadoop user from machines in the cluster. The simplest way to achieve this is to generate a public/private key pair, and it will be shared across the cluster using NFS.

First, generate an RSA key pair by typing the following in the hadoop user account:

```
% ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

Even though we want password-less logins, keys without passphrases are not considered good practice, so we specify a passphrase when prompted for one. We shall use ssh-agent to avoid the need to enter a password for each connection.

The private key is in the file specified by the -f option, ~/.ssh/id_rsa, and the public key is stored in a file with the same name with .pub appended, ~/.ssh/id_rsa.pub.

Next we need to make sure that the public key is in the ~/.ssh/authorized_keys file on all the machines in the cluster that we want to connect to. If the hadoop user‘s home directory is an NFS filesystem, as described earlier, then the keys can be shared across the cluster by typing:

```
% cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

If the home directory is not shared using NFS, then the public keys will need to be
shared by some other means.

Test that you can SSH from the master to a worker machine by making sure ssh-agent is running, and then run ssh-add to store your passphrase. You should be able to ssh to a worker without entering the passphrase again.

Hadoop Configuration

There are a handful of files for controlling the configuration of a Hadoop installation; the most important ones are listed in Figure 4.1

Filename	**Format**	**Description**
hadoop-env.sh	Bash script	Environment variables that are used in the scripts to run Hadoop.
core-site.xml	Hadoop configura- tion XML	Configuration settings for Hadoop Core, such as I/O settings that are common to HDFS and MapReduce.
hdfs-site.xml	Hadoop configura- tion XML	Configuration settings for HDFS daemons: the namenode, the secondary name- node, and the datanodes.
mapred-site.xml	Hadoop configura- tion XML	Configuration settings for MapReduce daemons: the jobtracker, and the tasktrackers.
Masters	Plain text	A list of machines (one per line) that each run a secondary namenode.
Slaves	Plain text	A list of machines (one per line) that each run a datanode and a tasktracker.
hadoop-met rics.properties	Java Properties	Properties for controlling how metrics are published in Hadoop
log4j.properties	Java Properties	Properties for system logfiles, the namenode audit log, and the task log for the tasktracker child process

Figure 4.1 : Hadoop configuration files

Configuration Management

Hadoop does not have a single, global location for configuration information. Instead, each Hadoop node in the cluster has its own set of configuration files, and it is up to administrators to ensure that they are kept in sync across the system. Hadoop provides a rudimentary facility for synchronizing configuration using rsync, alternatively there are parallel shell tools that can help do this, like dsh or pdsh.

Hadoop is designed so that it is possible to have a single set of configuration files that are used for all master and worker machines. The great advantage of this is simplicity, both conceptually (since there is only one configuration to deal with) and operationally (as the Hadoop scripts are sufficient to manage a single configuration setup).

For a cluster of any size, it can be a challenge to keep all of the machines in sync: consider what happens if the machine is unavailable when you push out an update—who en-sures it gets the update when it becomes available? This is a big problem and can lead to divergent installations, so even if you use the Hadoop control scripts for managing Hadoop, it may be a good idea to use configuration management tools for maintaining the cluster. These tools are also excellent for doing regular maintenance, such as patch-ing security holes and updating system packages.

Environment Settings

In this section, we consider how to set the variables in hadoop-env.sh.

Adding the above statement in the hadoop-env.sh file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

```
export JAVA_HOME= /usr/lib/jvm/java-6-sun
```

Java

The location of the Java implementation to use is determined by the JAVA_HOME setting in hadoop-env.sh, or from the JAVA_HOME shell environment variable, if not set in hadoop-env.sh. It‗s a good idea to set the value in hadoop-env.sh, so that it is clearly defined in one place, and to ensure that the whole cluster is using the same version of Java.

The JAVE HOME environment variable can be set using the a setenv statement in the .cshrc file (under a user‗s home directory) if using a C shell.

```
setenv JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Important Hadoop Daemon Properties

Hadoop has a bewildering number of configuration properties. In this section, we address the ones that you need to define (or at least understand why the default is appropriate) for any real-world working cluster. These properties are set in the Hadoop site files: core-site.xml, hdfs-site.xml, and mapred-site.xml. Figure 4.2: Important Hadoop Daemon Properties shows a typical example set of files. Notice that most are marked as final, in order to prevent them from being overridden by job configurations.

```
<?xml version="1.0"?>
<!-- core-site.xml -->
```

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://namenode/</value>
    <final>true</final>
  </property>
</configuration>

<?xml version="1.0"?>
<!-- hdfs-site.xml -->
<configuration>

 <property>
    <name>dfs.name.dir</name>
    <value>/disk1/hdfs/name,/remote/hdfs/name</value>
    <final>true</final>
  </property>
  <property>
    <name>dfs.data.dir</name>
  <value>/disk1/hdfs/data,/disk2/hdfs/data</value>
    <final>true</final>
  </property>

  <property>
    <name>fs.checkpoint.dir</name>
    <value>/disk1/hdfs/namesecondary,/disk2/hdfs/namesecondary</value>
    <final>true</final>
  </property>
</configuration>
<?xml version="1.0"?>
```

```
<!-- mapred-site.xml -->
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>jobtracker:8021</value>
    <final>true</final>
  </property>

  <property>
    <name>mapred.local.dir</name>
    <value>/disk1/mapred/local,/disk2/mapred/local</value>
    <final>true</final>
  </property>

  <property>
    <name>mapred.system.dir</name>
    <value>/tmp/hadoop/mapred/system</value>
    <final>true</final>
  </property>

  <property>
    <name>mapred.tasktracker.map.tasks.maximum</name>
    <value>7</value>
    <final>true</final>
  </property>

  <property>
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
    <value>7</value>
    <final>true</final>
  </property>


  <property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx400m</value>
    <!-- Not marked as final so jobs can include JVM debugging options -->
  </property>
</configuration>
```

Figure 4.2: Important Hadoop Daemon Properties

HDFS

To run HDFS, you need to designate one machine as a namenode. In this case, the property **fs.default.name** is a HDFS filesystem URI, whose host is the namenode‘s hostname or IP address, and port is the port that the namenode will listen on for RPCs. If no port is specified, the default of 8020 is used.

MapReduce

To run MapReduce, you need to designate one machine as a jobtracker, which on small clusters may be the same machine as the namenode. To do this, set the mapred.job.tracker property to the hostname or IP address and port that the jobtracker will listen on. Note that this property is not a URI, but a host-port pair, separated by a colon. The port number 8021 is a common choice.

Hadoop Daemon Addresses and Ports

Hadoop daemons generally run both an RPC server for communication between daemons, and a HTTP server to provide web pages for human consumption. Each server is configured by setting the network address and port number to listen on. By specifying the network address as 0.0.0.0, Hadoop will bind to all addresses on the machine. Alternatively, you can specify a single address to bind to. A port number of 0 instructs the server to start on a free port: this is generally discouraged, since it is incompatible with setting cluster-wide firewall policies.

Once you have a Hadoop cluster up and running, you need to give users access to it. This involves creating a home directory for each user, and setting ownership permissions on it:

```
% hadoop fs -mkdir /user/username
% hadoop fs -chown username:username /user/username
```

This is a good time to set space limits on the directory. The following sets a 1 TB limit on the given user directory:

```
% hadoop dfsadmin -setSpaceQuota 1t /user/username
```

Configuring Hadoop

The main settings for Hadoop reside in the hadoop-site.xml file under the conf directory. Initially the file is empty and needs to be configured. There are many values that can be specified, if a specific value is not specified the default value is used (the default values reside in the hadoop-default.xml file in the conf directory). The hadoop.tmp.dir parameter specifies the temporty directory that Hadoop uses to save dfs files. The fs.default.name parameter is set to the address of the master server in the cluster that is running the NameNode service. The mapred.job.tracker parameter is set to the server that is running the JobTracker service. The dfs.block.size parameter specifies the block size that the HDFS uses for file storage. The following is the configuration file from this experiment.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/opt/devinej/hadoop/tmp/\${user.name}</value>
  <description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://aldenv167:54310</value>
  <description>The name of the default file system. A URI whose
  scheme and authority determine the FileSystem implementation. The
  uri‘s scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class. The uri‘s authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>

<property>
  <name>mapred.job.tracker</name>
  <value>aldenv167:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at. If "local", then jobs are run in-process as a single map
  and reduce task.
</description>
</property>
<property>
  <name>dfs.replication</name>
  <value>2</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
</description>
</property>

<property>
  <name>dfs.block.size</name>
  <value>2097152</value>
  <description>Sets the block size to 2 MB since most of the files used in
  the empirical study are between 250KB-2MB
</description>
</property>
</configuration>
```

After editing the hadoop-site.xml file, the masters and slaves files need to be edit. Each file resides in the conf folder. The masters file contains a list of master server ip address or host names. This is usually just set to one machine for simple installations; the best machine for this would be the machine that is set up with Hadoop first. The slaves files is edited next. The slaves file should contain the host name or ip address of each machine that will run Hadoop. An example slaves file is provided below.

```
141.195.226.167

141.195.226.168

141.195.226.169

141.195.226.170

141.195.226.171

141.195.226.172

141.195.226.173

141.195.226.174
```

Deploying the Hadoop Installation to the Servers in the Cluster

The installation directory needs to be copied to each of the other servers in the cluster. This can be easily done by using the scp command. A sample of the command is given below. This will copy all of the installation and configuration files to the specified remote machine.

```
scp -r /hadoop_install_directory remote_computer:/hadoop_install_directory
```

Through the course of the experimentation it was necessary to write a short script to deploy the hadoop config files to all the computer in the cluster. It could very easily by modified to deploy the entire Hadoop install folder. The script can be found below.

```
#!/bin/sh

install_dir=/opt/devinej/hadoop


#Copy the configuration files to each of the machines in the cluster

for((i =167; i<=174; i++))

do
```

```
scp $install_dir/conf/masters aldenv$i:$install_dir/conf
scp $install_dir/conf/slaves aldenv$i:$install_dir/conf
scp $install_dir/conf/hadoop-site.xml aldenv$i:/$install_dir/conf
done
```

Formatting the Namenode

On the master server the namenode must be formated before the first run of Hadoop. Formatting the namenode prepares the temp folder specified in the config file for stor- ing data as part of the HDFS. If there is already data in the HDFS it will be erased. The command below is a sample on how to format the namenode.

```
install_dir/bin/hadoop dfs namenode -format
```

Starting the DFS Service

The DFS service is the main distributed file system service. The script starts the dfs service on each server in the cluster; it resides in bin directory and is called start-dfs.sh. The script will reads in the list of servers from the slaves file in the conf directory. The following command demonstrates the execution of the services.

```
hadoop_install_directory/bin/start-dfs.sh
```

Once the dfs service is started, typing the jps command on each slave server in the cluster should show DataNode. On the master server typing in jps should show DataNode, NameNode, and SecondaryNameNode. This indicates that the dfs service is running correctly.

Starting the MapReduce Service

The MapReduce service supports the use of MapReduce. The script to execute the service on each node in the cluster is called start-mapred.sh and resides in the bin directory. The command to execute the command is shown below.

```
hadoop_install_directory/bin/start-mapred.sh
```

Once the mapred service, typing the jps command on each slave server in the cluster should show TaskTaracker (in addition to DataNode). On the master server typing in jps should show JobTracker, (in addition to DataNode, NameNode, and SecondaryNameNode). This indicates that the MapReduce Service is correctly running. Note that to properly use Hadoop the mapred service must be running.

Accessing the HDFS

The HDFS can be accessed though the hadoop executable. A variety of typical Linux comands can be passed. The HDFS can also be accessed through the web interface at http://master_node:50070. A sample HDFS command is given below that will list everything in the top directory of the HDFS.

```
install_dir/bin/hadoop dfs -ls
```

Sending a MapReduce Job to Cluster

The hadoop executable allows for MapReduce job to be sent to the cluster. The syntax is provided below. The status of the job can be viewed at http://master_node:50030

```
install_dir/bin/hadoop jar program.jar org.myorg.ProgramName /input /output
```

REFERENCES

[1] ―Every day we create 2.5 quintillion bytes of data", s*toragenewsletter.com*. URL: *http://www.storagenewsletter.com/rubriques/market-reportsresearch/ibm-cmo-study/*, Press Release 2015, [Last accessed on 3.3.2017.]

[2] Dragland Å. ―Big Data – better or worse", Sintef Research news, 2013, URL: *http://www.sintef.no/home/Press-Room/Research-News/Big-Data--for-better-or-worse/*, [Last accessed on 3.3.2017.]

[3] CSC.com. URL: *http://www.csc.com/insights/flxwd/78931-big_data_universe_beginning_to_explode*, [Last accessed on 3. 3. 2017.]

[4] ―Bright Planet, Structured vs Unstructured data", URL: http://www.brightplanet.com/2012/06/structured-vs-unstructured-data/, 2012, [Last accessed on 3.3.2017.]

[5] Manuja M. and Garg D, ―Semantic Web Mining of Un-structured Data: Challenges and Opportunities", International Journal of Engineering (IJE), vol. 5, no. 3, pp. 51-57, 2014

[6] Mohanty S., Jagadeesh M. and Srivatsa H, ―Big Data Imperatives: Enterprise Big Data Warehouse, BI Implementations and Analytics", Apress, 2013

[7] White T, ―Hadoop: The Definitive Guide", O'Reilly Media; 2009

[8] Apache Hadoop 1.0 High Availability Solution on VMware vSphere ™

[9] Lublisnky B., Smith K.T. and Yakubovich A , Professional Hadoop Solutions, Wros Press.

[10] Devine, J. "Evaluating the Scalability of Hadoop in a Real and Virtual Environment." (2008).

[11] Björgvinsson, Andri Mar. "Distributed cluster pruning in Hadoop." (2010).

[12] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[13] Hadoop - Apache Software Foundation project home page, *http://hadoop.apache.org/*, [Last accessed on 3.3.2017]

[14] Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A, ―HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads", (VLDB ‗09), Lyon, France: VLDB Endowment, 2009.

[15] Cloudera Hadoop Training: MapReduce and HDFS, *http://vimeo.com/ 3584536*, 2009.

APPENDIX A

A test run

After writing a MapReduce job, it‘s normal to try it out on a small dataset to flush out any immediate problems with the code.

Single Node: Hadoop runs using the local filesystem with a local job runner, the output has been slightly reformatted to fit the page -

```
15/12/23 16:57:42 INFO input.FileInputFormat: Total input paths to process : 160
15/12/23 16:57:42 INFO util.NativeCodeLoader: Loaded the native-hadoop library
15/12/23 16:57:42 WARN snappy.LoadSnappy: Snappy native library not loaded
15/12/23 16:57:43 INFO mapred.JobClient: Running job: job_201512231655_0001
15/12/23 16:57:44 INFO mapred.JobClient:  map 0% reduce 0%
15/12/23 16:58:02 INFO mapred.JobClient:  map 1% reduce 0%
15/12/23 16:58:11 INFO mapred.JobClient:  map 2% reduce 0%
15/12/23 16:58:20 INFO mapred.JobClient:  map 3% reduce 0%15/12/23 16:58:29 INFO mapred.JobClient:  map
5% reduce 0%
15/12/23 16:58:38 INFO mapred.JobClient:  map 6% reduce 1%
15/12/23 16:58:47 INFO mapred.JobClient:  map 7% reduce 2%
15/12/23 16:58:56 INFO mapred.JobClient:  map 8% reduce 2%
15/12/23 16:59:05 INFO mapred.JobClient:  map 10% reduce 2%
15/12/23 16:59:08 INFO mapred.JobClient:  map 10% reduce 3%
15/12/23 16:59:14 INFO mapred.JobClient:  map 11% reduce 3%
15/12/23 16:59:23 INFO mapred.JobClient:  map 12% reduce 3%
15/12/23 16:59:32 INFO mapred.JobClient:  map 13% reduce 4%
15/12/23 16:59:41 INFO mapred.JobClient:  map 14% reduce 4%
15/12/23 16:59:50 INFO mapred.JobClient:  map 16% reduce 4%
15/12/23 16:59:53 INFO mapred.JobClient:  map 16% reduce 5%
15/12/23 16:59:59 INFO mapred.JobClient:  map 17% reduce 5%
15/12/23 17:00:08 INFO mapred.JobClient:  map 18% reduce 5%
15/12/23 17:00:17 INFO mapred.JobClient:  map 19% reduce 5%
15/12/23 17:00:20 INFO mapred.JobClient:  map 19% reduce 6%
15/12/23 17:00:26 INFO mapred.JobClient:  map 21% reduce 6%
15/12/23 17:00:36 INFO mapred.JobClient:  map 22% reduce 7%
15/12/23 17:00:45 INFO mapred.JobClient:  map 23% reduce 7%
15/12/23 17:00:54 INFO mapred.JobClient:  map 24% reduce 7%
15/12/23 17:01:03 INFO mapred.JobClient:  map 26% reduce 7%
15/12/23 17:01:06 INFO mapred.JobClient:  map 26% reduce 8%
15/12/23 17:01:12 INFO mapred.JobClient:  map 27% reduce 8%
15/12/23 17:01:21 INFO mapred.JobClient:  map 28% reduce 9%
15/12/23 17:01:30 INFO mapred.JobClient:  map 29% reduce 9%
15/12/23 17:01:36 INFO mapred.JobClient:  map 29% reduce 10%
15/12/23 17:01:39 INFO mapred.JobClient:  map 31% reduce 10%
15/12/23 17:01:48 INFO mapred.JobClient:  map 32% reduce 10%
```

```
15/12/23 17:01:57 INFO mapred.JobClient:  map 33% reduce 10%
15/12/23 17:02:06 INFO mapred.JobClient:  map 34% reduce 11%
15/12/23 17:02:15 INFO mapred.JobClient:  map 36% reduce 11%
15/12/23 17:02:21 INFO mapred.JobClient:  map 36% reduce 12%
15/12/23 17:02:24 INFO mapred.JobClient:  map 37% reduce 12%
15/12/23 17:02:33 INFO mapred.JobClient:  map 38% reduce 12%
15/12/23 17:02:42 INFO mapred.JobClient:  map 39% reduce 12%
15/12/23 17:02:51 INFO mapred.JobClient:  map 41% reduce 13%
15/12/23 17:03:00 INFO mapred.JobClient:  map 42% reduce 13%
15/12/23 17:03:06 INFO mapred.JobClient:  map 42% reduce 14%
15/12/23 17:03:09 INFO mapred.JobClient:  map 43% reduce 14%
15/12/23 17:03:18 INFO mapred.JobClient:  map 44% reduce 14%
15/12/23 17:03:27 INFO mapred.JobClient:  map 46% reduce 14%
15/12/23 17:03:30 INFO mapred.JobClient:  map 46% reduce 15%
15/12/23 17:03:36 INFO mapred.JobClient:  map 47% reduce 15%
15/12/23 17:03:45 INFO mapred.JobClient:  map 48% reduce 15%
15/12/23 17:03:51 INFO mapred.JobClient:  map 48% reduce 16%
15/12/23 17:03:54 INFO mapred.JobClient:  map 49% reduce 16%
15/12/23 17:04:03 INFO mapred.JobClient:  map 51% reduce 16%
15/12/23 17:04:12 INFO mapred.JobClient:  map 52% reduce 16%
15/12/23 17:04:15 INFO mapred.JobClient:  map 52% reduce 17%15/12/23 17:04:21 INFO mapred.JobClient:  map
53% reduce 17%
15/12/23 17:04:30 INFO mapred.JobClient:  map 54% reduce 17%
15/12/23 17:04:36 INFO mapred.JobClient:  map 54% reduce 18%
15/12/23 17:04:39 INFO mapred.JobClient:  map 56% reduce 18%
15/12/23 17:04:48 INFO mapred.JobClient:  map 57% reduce 18%
15/12/23 17:04:57 INFO mapred.JobClient:  map 58% reduce 18%
15/12/23 17:05:00 INFO mapred.JobClient:  map 58% reduce 19%
15/12/23 17:05:06 INFO mapred.JobClient:  map 60% reduce 19%
15/12/23 17:05:15 INFO mapred.JobClient:  map 61% reduce 20%
15/12/23 17:05:24 INFO mapred.JobClient:  map 62% reduce 20%
15/12/23 17:05:33 INFO mapred.JobClient:  map 63% reduce 20%
15/12/23 17:05:42 INFO mapred.JobClient:  map 65% reduce 20%
15/12/23 17:05:45 INFO mapred.JobClient:  map 65% reduce 21%
15/12/23 17:05:51 INFO mapred.JobClient:  map 66% reduce 21%
15/12/23 17:06:00 INFO mapred.JobClient:  map 67% reduce 22%
15/12/23 17:06:09 INFO mapred.JobClient:  map 68% reduce 22%
15/12/23 17:06:18 INFO mapred.JobClient:  map 70% reduce 22%
15/12/23 17:06:28 INFO mapred.JobClient:  map 71% reduce 22%
15/12/23 17:06:31 INFO mapred.JobClient:  map 71% reduce 23%
15/12/23 17:06:37 INFO mapred.JobClient:  map 72% reduce 23%
15/12/23 17:06:46 INFO mapred.JobClient:  map 73% reduce 24%
15/12/23 17:06:55 INFO mapred.JobClient:  map 75% reduce 24%
15/12/23 17:07:01 INFO mapred.JobClient:  map 75% reduce 25%
15/12/23 17:07:04 INFO mapred.JobClient:  map 76% reduce 25%
15/12/23 17:07:13 INFO mapred.JobClient:  map 77% reduce 25%
15/12/23 17:07:22 INFO mapred.JobClient:  map 78% reduce 25%
15/12/23 17:07:31 INFO mapred.JobClient:  map 80% reduce 26%
15/12/23 17:07:40 INFO mapred.JobClient:  map 81% reduce 26%
```

```
15/12/23 17:07:46 INFO mapred.JobClient:  map 81% reduce 27%
15/12/23 17:07:49 INFO mapred.JobClient:  map 82% reduce 27%
15/12/23 17:07:58 INFO mapred.JobClient:  map 83% reduce 27%
15/12/23 17:08:07 INFO mapred.JobClient:  map 85% reduce 27%
15/12/23 17:08:16 INFO mapred.JobClient:  map 86% reduce 28%
15/12/23 17:08:25 INFO mapred.JobClient:  map 87% reduce 28%
15/12/23 17:08:34 INFO mapred.JobClient:  map 88% reduce 28%
15/12/23 17:08:37 INFO mapred.JobClient:  map 88% reduce 29%
15/12/23 17:08:43 INFO mapred.JobClient:  map 90% reduce 29%
15/12/23 17:08:49 INFO mapred.JobClient:  map 90% reduce 30%
15/12/23 17:08:52 INFO mapred.JobClient:  map 91% reduce 30%
15/12/23 17:09:01 INFO mapred.JobClient:  map 92% reduce 30%
15/12/23 17:09:10 INFO mapred.JobClient:  map 93% reduce 30%
15/12/23 17:09:16 INFO mapred.JobClient:  map 93% reduce 31%
15/12/23 17:09:19 INFO mapred.JobClient:  map 95% reduce 31%
15/12/23 17:09:28 INFO mapred.JobClient:  map 96% reduce 31%
15/12/23 17:09:31 INFO mapred.JobClient:  map 96% reduce 32%
15/12/23 17:09:37 INFO mapred.JobClient:  map 97% reduce 32%
15/12/23 17:09:46 INFO mapred.JobClient:  map 98% reduce 32%
15/12/23 17:09:55 INFO mapred.JobClient:  map 100% reduce 32%
15/12/23 17:10:01 INFO mapred.JobClient:  map 100% reduce 33%15/12/23 17:10:07 INFO mapred.JobClient:
map 100% reduce 94%
15/12/23 17:10:16 INFO mapred.JobClient:  map 100% reduce 100%
15/12/23 17:10:21 INFO mapred.JobClient: Job complete: job_201512231655_0001
15/12/23 17:10:21 INFO mapred.JobClient: Counters: 29
15/12/23 17:10:21 INFO mapred.JobClient:   Job Counters
15/12/23 17:10:21 INFO mapred.JobClient:     Launched reduce tasks=1
15/12/23 17:10:21 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=1274373
15/12/23 17:10:21 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
15/12/23 17:10:21 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots
(ms)=0
15/12/23 17:10:21 INFO mapred.JobClient:     Launched map tasks=160
15/12/23 17:10:21 INFO mapred.JobClient:     Data-local map tasks=160
15/12/23 17:10:21 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=703846
15/12/23 17:10:21 INFO mapred.JobClient:   File Output Format Counters
15/12/23 17:10:21 INFO mapred.JobClient:     Bytes Written=79951
15/12/23 17:10:21 INFO mapred.JobClient:   FileSystemCounters
15/12/23 17:10:21 INFO mapred.JobClient:     FILE_BYTES_READ=66551206
15/12/23 17:10:21 INFO mapred.JobClient:     HDFS_BYTES_READ=977095665
15/12/23 17:10:21 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=83344050
15/12/23 17:10:21 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=79951
15/12/23 17:10:21 INFO mapred.JobClient:   File Input Format Counters
15/12/23 17:10:21 INFO mapred.JobClient:     Bytes Read=977071200
15/12/23 17:10:21 INFO mapred.JobClient:   Map-Reduce Framework
15/12/23 17:10:21 INFO mapred.JobClient:     Map output materialized bytes=13311040
15/12/23 17:10:21 INFO mapred.JobClient:     Map input records=24532800
15/12/23 17:10:21 INFO mapred.JobClient:     Reduce shuffle bytes=13227846
15/12/23 17:10:21 INFO mapred.JobClient:     Spilled Records=5236640
```

```
15/12/23 17:10:21 INFO mapred.JobClient:     Map output bytes=1511458400
15/12/23 17:10:21 INFO mapred.JobClient:     Total committed heap usage (bytes)=20391591936
15/12/23 17:10:21 INFO mapred.JobClient:     CPU time spent (ms)=908850
15/12/23 17:10:21 INFO mapred.JobClient:     Combine input records=137091040
15/12/23 17:10:21 INFO mapred.JobClient:     SPLIT_RAW_BYTES=24465
15/12/23 17:10:21 INFO mapred.JobClient:     Reduce input records=872800
15/12/23 17:10:21 INFO mapred.JobClient:     Reduce input groups=5455
15/12/23 17:10:21 INFO mapred.JobClient:     Combine output records=4363840
15/12/23 17:10:21 INFO mapred.JobClient:     Physical memory (bytes) snapshot=29376294912
15/12/23 17:10:21 INFO mapred.JobClient:     Reduce output records=5455
15/12/23 17:10:21 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=262792732672
15/12/23 17:10:21 INFO mapred.JobClient:     Map output records=133600000
```

Multi Node: Hadoop runs using the local filesystem with a local job runner, the output has been slightly reformatted to fit the page -

```
15/12/24 12:22:40 INFO input.FileInputFormat: Total input paths to process : 160
15/12/24 12:22:40 INFO util.NativeCodeLoader: Loaded the native-hadoop library
15/12/24 12:22:40 WARN snappy.LoadSnappy: Snappy native library not loaded
15/12/24 12:22:41 INFO mapred.JobClient: Running job: job_201512241219_0001
15/12/24 12:22:42 INFO mapred.JobClient:  map 0% reduce 0%
15/12/24 12:23:00 INFO mapred.JobClient:  map 2% reduce 0%
15/12/24 12:23:09 INFO mapred.JobClient:  map 5% reduce 0%
15/12/24 12:23:18 INFO mapred.JobClient:  map 7% reduce 1%
15/12/24 12:23:26 INFO mapred.JobClient:  map 8% reduce 1%
15/12/24 12:23:27 INFO mapred.JobClient:  map 10% reduce 2%
15/12/24 12:23:33 INFO mapred.JobClient:  map 10% reduce 3%
15/12/24 12:23:35 INFO mapred.JobClient:  map 11% reduce 3%
15/12/24 12:23:36 INFO mapred.JobClient:  map 12% reduce 3%
15/12/24 12:23:44 INFO mapred.JobClient:  map 13% reduce 3%
15/12/24 12:23:45 INFO mapred.JobClient:  map 14% reduce 3%
15/12/24 12:23:48 INFO mapred.JobClient:  map 14% reduce 5%
15/12/24 12:23:53 INFO mapred.JobClient:  map 16% reduce 5%
15/12/24 12:23:54 INFO mapred.JobClient:  map 17% reduce 5%
15/12/24 12:24:02 INFO mapred.JobClient:  map 18% reduce 5%
15/12/24 12:24:03 INFO mapred.JobClient:  map 19% reduce 5%
15/12/24 12:24:11 INFO mapred.JobClient:  map 21% reduce 5%
15/12/24 12:24:12 INFO mapred.JobClient:  map 22% reduce 6%
15/12/24 12:24:18 INFO mapred.JobClient:  map 22% reduce 7%
15/12/24 12:24:20 INFO mapred.JobClient:  map 23% reduce 7%
15/12/24 12:24:21 INFO mapred.JobClient:  map 24% reduce 7%
15/12/24 12:24:29 INFO mapred.JobClient:  map 26% reduce 7%
15/12/24 12:24:30 INFO mapred.JobClient:  map 27% reduce 7%
15/12/24 12:24:33 INFO mapred.JobClient:  map 27% reduce 9%
15/12/24 12:24:38 INFO mapred.JobClient:  map 28% reduce 9%
15/12/24 12:24:39 INFO mapred.JobClient:  map 29% reduce 9%
15/12/24 12:24:47 INFO mapred.JobClient:  map 31% reduce 9%
```

```
15/12/24 12:24:48 INFO mapred.JobClient:  map 32% reduce 10%
15/12/24 12:24:56 INFO mapred.JobClient:  map 33% reduce 10%
15/12/24 12:24:57 INFO mapred.JobClient:  map 34% reduce 10%
15/12/24 12:25:03 INFO mapred.JobClient:  map 34% reduce 11%
15/12/24 12:25:05 INFO mapred.JobClient:  map 36% reduce 11%
15/12/24 12:25:06 INFO mapred.JobClient:  map 37% reduce 11%
15/12/24 12:25:14 INFO mapred.JobClient:  map 38% reduce 11%
15/12/24 12:25:15 INFO mapred.JobClient:  map 39% reduce 11%
15/12/24 12:25:18 INFO mapred.JobClient:  map 39% reduce 13%15/12/24 12:25:23 INFO mapred.JobClient:  map
41% reduce 13%
15/12/24 12:25:24 INFO mapred.JobClient:  map 42% reduce 13%
15/12/24 12:25:32 INFO mapred.JobClient:  map 43% reduce 13%
15/12/24 12:25:33 INFO mapred.JobClient:  map 44% reduce 14%
15/12/24 12:25:41 INFO mapred.JobClient:  map 46% reduce 14%
15/12/24 12:25:42 INFO mapred.JobClient:  map 47% reduce 15%
15/12/24 12:25:50 INFO mapred.JobClient:  map 48% reduce 15%
15/12/24 12:25:51 INFO mapred.JobClient:  map 49% reduce 15%
15/12/24 12:25:59 INFO mapred.JobClient:  map 51% reduce 15%
15/12/24 12:26:00 INFO mapred.JobClient:  map 52% reduce 15%
15/12/24 12:26:03 INFO mapred.JobClient:  map 52% reduce 17%
15/12/24 12:26:08 INFO mapred.JobClient:  map 53% reduce 17%
15/12/24 12:26:09 INFO mapred.JobClient:  map 54% reduce 17%
15/12/24 12:26:17 INFO mapred.JobClient:  map 57% reduce 18%
15/12/24 12:26:26 INFO mapred.JobClient:  map 60% reduce 19%
15/12/24 12:26:32 INFO mapred.JobClient:  map 60% reduce 20%
15/12/24 12:26:35 INFO mapred.JobClient:  map 62% reduce 20%
15/12/24 12:26:44 INFO mapred.JobClient:  map 65% reduce 20%
15/12/24 12:26:47 INFO mapred.JobClient:  map 65% reduce 21%
15/12/24 12:26:53 INFO mapred.JobClient:  map 67% reduce 21%
15/12/24 12:27:02 INFO mapred.JobClient:  map 70% reduce 21%
15/12/24 12:27:05 INFO mapred.JobClient:  map 70% reduce 22%
15/12/24 12:27:11 INFO mapred.JobClient:  map 72% reduce 22%
15/12/24 12:27:14 INFO mapred.JobClient:  map 72% reduce 23%
15/12/24 12:27:20 INFO mapred.JobClient:  map 75% reduce 24%
15/12/24 12:27:29 INFO mapred.JobClient:  map 77% reduce 25%
15/12/24 12:27:38 INFO mapred.JobClient:  map 80% reduce 25%
15/12/24 12:27:47 INFO mapred.JobClient:  map 82% reduce 25%
15/12/24 12:27:50 INFO mapred.JobClient:  map 82% reduce 26%
15/12/24 12:27:56 INFO mapred.JobClient:  map 85% reduce 26%
15/12/24 12:27:59 INFO mapred.JobClient:  map 85% reduce 27%
15/12/24 12:28:05 INFO mapred.JobClient:  map 87% reduce 28%
15/12/24 12:28:14 INFO mapred.JobClient:  map 90% reduce 29%
15/12/24 12:28:21 INFO mapred.JobClient:  map 90% reduce 30%
15/12/24 12:28:24 INFO mapred.JobClient:  map 92% reduce 30%
15/12/24 12:28:33 INFO mapred.JobClient:  map 95% reduce 30%
15/12/24 12:28:42 INFO mapred.JobClient:  map 97% reduce 30%
15/12/24 12:28:45 INFO mapred.JobClient:  map 97% reduce 31%
15/12/24 12:28:51 INFO mapred.JobClient:  map 100% reduce 32%
15/12/24 12:29:03 INFO mapred.JobClient:  map 100% reduce 100%
```

```
15/12/24 12:29:08 INFO mapred.JobClient: Job complete: job_201512241219_0001
15/12/24 12:29:08 INFO mapred.JobClient: Counters: 29
15/12/24 12:29:08 INFO mapred.JobClient:   Job Counters
15/12/24 12:29:08 INFO mapred.JobClient:     Launched reduce tasks=1
15/12/24 12:29:08 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=1249319
15/12/24 12:29:08 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving
slots (ms)=0
15/12/24 12:29:08 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots
(ms)=015/12/24 12:29:08 INFO mapred.JobClient:     Launched map tasks=161
15/12/24 12:29:08 INFO mapred.JobClient:     Data-local map tasks=161
15/12/24 12:29:08 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=352161
15/12/24 12:29:08 INFO mapred.JobClient:   File Output Format Counters
15/12/24 12:29:08 INFO mapred.JobClient:     Bytes Written=79951
15/12/24 12:29:08 INFO mapred.JobClient:   FileSystemCounters
15/12/24 12:29:08 INFO mapred.JobClient:     FILE_BYTES_READ=66551206
15/12/24 12:29:08 INFO mapred.JobClient:     HDFS_BYTES_READ=977094763
15/12/24 12:29:08 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=83346465
15/12/24 12:29:08 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=79951
15/12/24 12:29:08 INFO mapred.JobClient:   File Input Format Counters
15/12/24 12:29:08 INFO mapred.JobClient:     Bytes Read=977071200
15/12/24 12:29:08 INFO mapred.JobClient:   Map-Reduce Framework
15/12/24 12:29:08 INFO mapred.JobClient:     Map output materialized bytes=13311040
15/12/24 12:29:08 INFO mapred.JobClient:     Map input records=24532800
15/12/24 12:29:08 INFO mapred.JobClient:     Reduce shuffle bytes=13227846
15/12/24 12:29:08 INFO mapred.JobClient:     Spilled Records=5236640
15/12/24 12:29:08 INFO mapred.JobClient:     Map output bytes=1511458400
15/12/24 12:29:08 INFO mapred.JobClient:     Total committed heap usage (bytes)=20391591936
15/12/24 12:29:08 INFO mapred.JobClient:     CPU time spent (ms)=878310
15/12/24 12:29:08 INFO mapred.JobClient:     Combine input records=137091040
15/12/24 12:29:08 INFO mapred.JobClient:     SPLIT_RAW_BYTES=23563
15/12/24 12:29:08 INFO mapred.JobClient:     Reduce input records=872800
15/12/24 12:29:08 INFO mapred.JobClient:     Reduce input groups=5455
15/12/24 12:29:08 INFO mapred.JobClient:     Combine output records=4363840
15/12/24 12:29:08 INFO mapred.JobClient:     Physical memory (bytes) snapshot=29465100288
15/12/24 12:29:08 INFO mapred.JobClient:     Reduce output records=5455
15/12/24 12:29:08 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=262792732672
15/12/24 12:29:08 INFO mapred.JobClient:     Map output records=133600000
```

APPENDIX B

A Word Frequency

```
import java.io.IOException;

import java.util.StringTokenizer;



import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;



public class WordCount {



  public static class TokenizerMapper

       extends Mapper<Object, Text, Text, IntWritable>{



    private final static IntWritable one = new IntWritable(1);

    private Text word = new Text();



    public void map(Object key, Text value, Context context
               ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

```
 public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```