INDEXING STRATEGY FOR BIG DATA PROCESSING: A CASE STUDY OF PINGER

FATIMA BINTA ADAMU

MASTER OF SCIENCE INFORMATION TECHNOLOGY
UNIVERSITI UTARA MALAYSIA
2015

Perakuan Kerja Tesis/Disertasi

(To be substituted with signed document for this page)

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Dengan adanya jumlah data yang besar yang dikumpul secara berterusan dan dikongsikan oleh organisasi, maka telah menjadi keperluan untuk memenuhi pemprosesan yang baru muncul dan mendapatkan semula keperluan yang berkaitan dengan jumlah data yang besar ini. Hal ini boleh dicapai melalui mengindeks set data dan mengurangkan pengiraan berat biasa kepada strategi pengindeksan yang terkini semasa pemprosesan jumlah set data yang sangat besar. Kajian ini mencadangkan satu strategi Pengindeksan yang dinamakan sebagai Big Data INDexing Stratergy (BIND), yang menggunakan konsep pengkomputeran selari berprestasi tinggi. BIND menyokong pengedaran data selari dan menjalankan pemprosesan dalam bentuk MapReduce. Bagi membina strategi BIND, konsep penjadualan tugas lan foster untuk pemprosesan selari digunakan. Cadangan strategi pengindeksan yang pertama kali diuji ke atas persekitaran kelompok 2-nod mempunyai pelbagai saiz set data digunakan untuk mengambil perhatian samada prestasinya bertambah baik atau merosot mengikut pertambahan saiznya. Selepas itu, ia telah diuji pada kelompok 3-nod untuk diambil perhatian prestasinya apabila bilangan sumber pengiraan ditambah. Keputusan menunjukkan BIND mengurangkan pemprosesan dan query time berbanding dengan strategi semasa. Hasil kajian mempunyai implikasi yang signifikan dalam keberkesanannya menguruskan Big Data dan memudahkan penyimpanan data serta dapatan semula maklumat untuk pengguna dan organisasi yang menguruskan Big Data ini.

Kata kunci: Big Data; Pemprosesan; Pengindeksan; MapReduce; Dapatan Semula Maklumat

Abstract

With the huge amount of data continuously accumulated and shared by individuals and organizations, it has become necessary to meet the emerging processing and retrieval requirements associated with these large volumes of complex data. This could be achieved by indexing the data sets and reducing heavy computational overhead accustomed to most current indexing strategies during processing of very large amount of data sets. This study proposed a novel Indexing strategy called Big Data INDexing Strategy (BIND), using a concept of high performance parallel computing. BIND supports parallel distribution of data and performs processing in a MapReduce fashion. To develop BIND strategy, Ian foster's task-scheduling concept for parallel processing is applied. The proposed indexing strategy was first tested on a 2-node cluster environment where varying sizes of datasets were used to note if the performance improves or declines as the size of the data increases. Subsequently, it was tested on a 3-node cluster to note the performance when the number of computation resources are increased. The results demonstrate that BIND minimizes the processing and query time as compared to the current strategy. The findings have significant implication in efficiently managing Big Data and facilitating data storage and information retrieval for users and organizations that manage Big Data.

Keywords: Big Data; Processing; Indexing; MapReduce; Information Retrieval

Acknowledgements

In the name of ALLAH, Most Gracious, Most Merciful:

"*Work; so Allah will see your work and (so will) His Messenger and the believers*;"

(The Holy Quran - AtTawbah 9:105)

All thanks and praises to Allah (SWT) for the blessings of life and for guiding me through studies and life.

My sincere appreciation goes to my able mentor and supervisor, Dr. Adib Habbal, for stirring my interest to Big Data and for diligently going through this work to make it a success.

To my dearest Parents, you are my everyday teachers. Thank you is simply not enough for your prayers, your support, your guidance, your patience, and everything you stand for me. May Allah continue to bestow wisdom upon you and shower you with immense blessings.

To all my instructors who guided me through studies and life, may God continue to guide you and reward you with more beneficial knowledge. My profound appreciation goes to the team of InterNetWorks Research Laboratory, UUM; the SLAC Laboratory, Stanford University, US; Professor Bebo White, Dr. Les Cottrell, Dr. Anjum Naveed, and the entire PingER team. Most of all, I say a big thank you to the School of Computing, Universiti Utara Malaysia.

To my family, friends, and well wishers, there is no greater support than you. Thank you for all the love and encouragement you shower on me.

My unrelented prayers and appreciation goes to my adviser, my guardian, my guru,

my counselor, my consultant, my confidant, and my number one friend - my husband, Ibrahim Abdullahi. May Allah continue to guide and reward you.

And to my dearest little one, my darling Hanan, who went through a masters degree before the age of two for mama, I LOVE YOU. May Allah grant you a blessed life and aljannat firdaus.

Dedication

To my family.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BIND	-	Big Data INDexing
GPS	-	Global Positioning System
RFID	-	Radio Frequency Identification System
IT	-	Information Technology
MB	-	Megabyte
ZB	-	Zettabyte
TB	-	Terabyte
EX	-	Exabyte
SQL	-	Structured Query Language
RDBMS	-	Relational Database Management System
IDC	-	International Data Corporation
PingER	-	Ping End-to-end Reporting
ABDC	-	Analytics and Big Data Committee
SDAV	-	Scalable Data Management, Analysis and Visualization Institute
MA	-	Measurement Agents
PC	-	Personal Computer
IEPM	-	Internet End-to-end Performance Measurement
SLAC	-	Standard Linear Accelerator Center
HDFS	-	Hadoop Distributed File System
CSV	-	Comma Separated Values
JT	-	JobTracker
RM	-	ResourceManager
TT	-	TaskTracker
NM	-	NodeManager
DN	-	DataNode
NN	-	NameNode
SNN	-	SecondaryNameNode
RAM	-	Random Access Memory
CPU	-	Central Processing Unit

NNS	-	Nearest Neighbor Search
AI	-	Artificial Intelligence
NAI	-	Non-Artificial Intelligence
LSI	-	Latent Semantic Indexing
HMM		Hidden Markov Model
SVD		Singular Value Decomposition
RDF		Resource Description Framework
GiST		Generalized Search Tree
GIN		Generalized Inverted Index
PT		Posting Tree
ET		Entries Tree
PL		Posting List
RTT		Round Trip Time
ICMP		Internet Control Message Protocol
QoS		Quality of Service
AIS		Asynchronous Index Strategy
ETL		Extract, Transform, Load
LFS		Local File System
K-V		Key-Value
K-L(V)		Key-List (Value)
UI		User Interface
SSH		Secure Shell
JAR		Java Archive
IDE		Integrated Development Environment
HQL		Hive Query Language
IS		Indexing Strategy
TTL		Time-To-Leave
IR		Information Retrieval

CHAPTER ONE
INTRODUCTION

1.1 OVERVIEW

Big Data is a term used to describe very large data sets that are of different forms or structure (complex), generated at a very high speed, and cannot be managed by traditional database management systems [6]. This definition explains the three (3) main characteristics associated with Big Data: volume, variety and velocity (3Vs), and the value that can be extracted from it is seen as a fourth characteristic (4V's) [7]. Big Data is sourced from so many end devices such as Personal Computers (PC), smart phones, Global Positioning System (GPS) devices [8], sensors, Radio Frequency Identification (RFID) devices, etc. Also, online applications such as social networks and applications that involve video streaming are great sources that generate Big Data.

According to Zhou et al. in [9], the total size of data generated will surpass 7.9 Zettabytes (ZB) by the end of 2015, and predicted to reach 35ZB in 2020. Significant interest have been taken in this area lately, because of the ubiquitous and pervasive nature of data, and the emergence of mobile and cloud computing which results in the generation of huge amounts of data. The application of Big Data analytic has been of great value in terms of decision making in areas concerned with development [10, 11], security [12], and health care [13, 14, 15]. Big Data has also been used in the prediction of natural disaster, energy consumption, economic productivity, social network (for customer or consumer interest), Information Technology (IT), business[16], and other domains. Cisco related that organizations such as Facebook, Yahoo, Google, Twitter, etc., deal with Big Data on a daily basis, and have realized the need to analyze historical data for decision making [17]. The use of so many wired and wireless devices has generated voluminous data that are to be collected, transferred, processed,

stored and retrieved as soon as they are produced, or from time to time. Data can be stored, processed, retrieved, and used as an anchor for decision making. These operations are becoming difficult to perform because data keeps increasing in volume and complexity as time passes by [6].

Over the past decades, the storage of data has evolved from the manual gathering or collection of data, to the automation of data collection [18, 19]. This is as a result of the huge size of data generated nowadays, and the difficulty that comes with manual collection, processing, and retrieval of information. Tools such as Hadoop, Spark [20, 14], Sqoop, Storm[5, 14], Flume, Kafka, Scribe, and S4, have been developed to handle this type of data [6]. Hadoop [4] appears to be one of the most popular tools to manage Big Data for further analysis. It enables very large volumes of data to be processed and stored. It also enables users to analyze the data in ways not feasible using Structured Query Language (SQL) based approaches [4]. Hadoop is chosen over Relational Database Management System (RDBMS) [21], because it is more scalable – good for growing data – unlike RDBMS. The growing data volume should be met with efficient processing and storage method. Retrieval of data could become difficult if processing is not properly done.

The challenges of information retrieval are brought about by the large amount of complex data that is generated and collected. According to Li et al., data storage has doubled every 40 months since 1980 [6]. As of 2012, 2.5 Exabytes (EB) of data were created [6]. Also, International Data Corporation (IDC) predicts that the global Big Data will multiply by 50 times in the next decade [22]. This should be met with efficient processing strategies. In line with this, several studies are being conducted on the processing of Big Data [6] [23] [5]. Some committees and institutes such as Analytic and Big Data Committee (ABDC), Scalable Data Management, Analysis and

Visualization Institute (SDAV) – US, Alan Turing Institute – Britain, and Ping End-to-end Reporting (PingER) – Malaysia, to mention a few, have put up research projects for study on handling Big Data. Live data from PingER Measurement Agents (MA) was used to support and achieve the objectives in this study. PingER is the Internet End-to-end Performance Measurement (IEPM) project, that monitors the end-to-end performance of the network or Internet links. PingER is led by the Standard Linear Accelerator Center (SLAC), presently known as SLAC National Accelerator Laboratory. PingER MAs monitors Internet performance, and store the results for future analysis. The data gathered from the PingER MAs are stored, to be used for the observation of Internet performance within Malaysia, South East Asia, and (between) Malaysia and the rest of the world. The insight gained from analyzing the data, can be used in managing Internet usage, detecting system failure, arresting network performance issues, and then taking precautions before they get out of hand. Therefore, processing of these data (Ping logs) has to be done in a way that yields efficient retrieval for analysis and decision making. Hence, the design of an indexing strategy for Big Data storage and retrieval, is of paramount importance to PingER.

1.2 PROBLEM STATEMENT

In this information age, large amount of unstructured data is generated from various online applications through so many end devices such as PCs, smart phones, GPS devices, sensors, RFID devices, and lots more. Data is increasingly spread across multiple and different machines in different formats. This is evident in the amount of data extracted from emails, web pages, social media, etc. on a daily basis. Gigabytes to terabytes of new data are generated to be processed and stored on a daily basis [24]. How to process these vast amounts of fast growing data has become a challenge faced by most enterprises [25]. It has become necessary to meet the emerging storage and retrieval requirements associated with these large volumes of complex data [6].

Storage and retrieval of information from Big Data, can be facilitated by efficient processing. The ability to process Big Data by creating appropriate indexing strategies to facilitate storage and retrieval of information, is a problem [26, 27, 28, 25, 9, 5, 29, 30]. Moreover, a large scale of data necessitates an efficient indexing strategy to achieve easy information retrieval. In line with this, there is a need to present a Big Data indexing strategy to ensure storage of data in a manner that allows for efficient information retrieval. Storage of Big Data in an indexed form, allows for efficient retrieval. Hence, indexing approaches have become a major research issue of information retrieval [5, 31]. However, organizations are still challenged with how to process and retrieve information from huge amounts of unstructured data [30, 28]. The study by Gollub *et al* [30] presented a Big Data processing strategy to mitigate the problem. Their strategy was based on the Inverted Indexing technique and uses the MapReduce framework. It also uses the default RecordReader which processes one record at a time, thereby, concentrating more on the MapReduce phases. Given a set of library documents, the strategy integrates already existing library systems and uses the MapReduce framework to create a hierarchy of keyqueries which covers the selected set of library documents. The advantage of this approach is that it solves the problem of obsolation of digital contents or shelves. An automatic constant update of user's keyqueries yield efficient query results. Furthermore, keyqueries become easier on the datasets since they were able to index the datasets with the queries as key and the set of library documents as values. The problem with this approach is that it incures computational overhead since it concentrates on the MapReduce phase only and relies on the default InputFormat and RecordReader, processing one record per line for each individual mapper. Also, the mappers operate simultaneously, forcing mappers who have completed task execution to wait for other mappers before further allocation of task. This slows down the process of task execution (processing time), and consequently slows down the time it takes for the information to be available for

retrieval and use. Therefore, there is a need for further research on Big Data processing [6, 5].

1.3 RESEARCH QUESTIONS

1. Can the design of an indexing strategy facilitate information retrieval in Big Data?
2. How can PingER log data be processed to facilitate information retrieval?
3. What is the impact of the proposed indexing strategy on the processing time and the query time?

1.4 RESEARCH OBJECTIVES

1. To design an indexing strategy called BIND strategy for efficient Big Data processing in order to facilitate storage and information retrieval using the MapReduce framework.
2. To implement and verify the proposed indexing strategy using PingER log data.
3. To evaluate the proposed indexing strategy in terms of processing time and query time.

1.5 SIGNIFICANT OF THE STUDY

This study is of relevance to various domains such as agriculture, health care, marketing, security, fraud detection, business, network performance monitoring, and other fields that require an efficient means of Big Data processing for easy storage and information retrieval, especially for analysis and decision making.

Logs from PingER MA, are basically stored in flat files. As the logs keep accumulating and increasing in volume, the storage and retrieval of data could become a cum-

bersome exercise. Hence, this study is of significance to PingER.

1.6 RESEARCH OUTCOMES

The outcome of this study is as follows:

1. The design of a Big Data indexing strategy, that processes data to enable storage in an indexed form, hence, facilitating information retrieval.
2. The implementation and evaluation of the proposed strategy.
3. A strategy that minimizes Big Data processing and query time.

1.7 RESEARCH SCOPE

The study focused on Big Data processing for efficient storage and retrieval. To achieve the objectives, live data from PingER MA was used. The data are logs accumulated form monitoring internet performance by PingER MA, which are stored in their original forms in flat files. Hadoop MapReduce and Hadoop Distributed File System (HDFS), will be used in Big Data processing and storage respectively. Meanwhile, optimization of Big Data management tools, and Big Data analytic, are out of this study's scope.

1.8 ORGANIZATION OF THE STUDY

The study is organized as follows:

Chapter 1 introduces Big Data, its characteristics and sources. It also contains the problem statement, objectives, significance, outcome and scope of the study. The general concept of Big Data was covered. Chapter 2 reviews the past literature related to the study. It also explains the key terms used in the study. The various indexing

strategies where explained, as well as the strategy most suitable for the study. While Chapter 3 explains the methodology used in achieving the objectives, Chapter 4 describes the implementation of the proposed strategy and the tools utilized in the study. It also discusses the design of the proposed strategy. Chapter 5 evaluates the results and summarises the work. Areas for future study where proposed as well.

CHAPTER TWO
LITERATURE REVIEW

This chapter reviews the main literatures associated with this study. Key terms, as well as previous studies conducted are carefully selected. Each of the studies selected, define its contribution, analysis, approaches and methods explored. Section 2.1 explains Big Data; its definition, main characteristics, and brief history. It also explains the categories of Big Data, and the sources of Big Data. Section 2.2 discusses the characteristics mentioned, as argued in previous studies. It also shows how each characteristic affect Big Data storage and information retrieval. Section 2.3 explains how Big Data can be managed before processing. This includes Big Data extraction, transformation, and loading. Section 2.4 describes Big Data processing. It also outlines the various tools for processing and storing Big Data. It further explains the tools that will be used in this study. Section 2.5 explains Big Data indexing and various types of indexing strategies. Section 2.6 explains Big Data storage and information retrieval. Section 2.7 describes PingER, and Section 2.8 is about flat file, which is the initial storage that holds the data sets to be used for this study. Section 2.9 relates previous studies on Big Data storage and retrieval, and Section 2.10 summarizes the chapter.

2.1 BIG DATA OVERVIEW

Big Data is a term for very large and complex data sets, generated at a very high speed and that cannot be managed by traditional data management tools. The main characteristics of Big Data are its volume, velocity, variety and value (4V's) [7]. The volume denotes the large size that Big Data comes in; Variety means the different forms or complexity of the data; velocity refers to the speed at which data are created or generated; and value stands for the value that could be extracted or the insight gained after analytics are done. These characteristics of Big Data explain that value or

relevant information can be acquired from analyzing a large amount of accumulated or stored data as shown in Figure 2.1. Figure 2.1 demonstrates that as the growth of data increases with time, so does the growth of relevant information that can be retrieved after analysis.

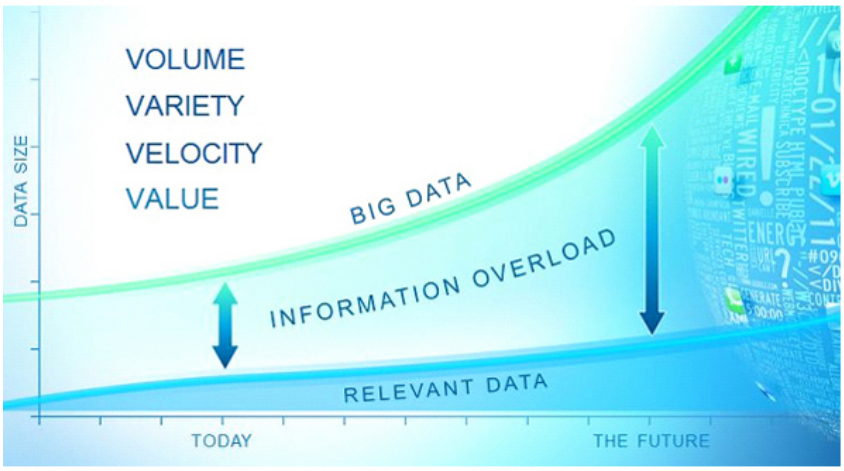

Figure 2.1: Big Data Value [1]

In 1941, large volume of data or the growth or growing rate of data was quantified and referred to as 'information explosion'. In 1944, it was estimated that the American University libraries doubled in size every sixteen years. Given this, it was speculated that in 2040, libraries will have approximately 200,000,000 volumes occupying over 6,000 miles of shelves [32]. This emphasizes the growth of data.

Today, Big Data has gained significance and researches are being carried out on it to realize its potentials, and to improve business and life as a whole. It is evident that Big Data has exhibited lots of potential in research communities and industries [15]. Due to this, People are keen to discover additional values from accumulated data [7]. This results in yearning for knowledge on how to process the data to facilitate storage and retrieval of information.

Big Data is categorized into three types: structured, unstructured and semi-structured [24]. Structured data are mostly data found in the Relational Database Management System (RDBMS). They fit into the management principles of RDBMS on how they will be stored (defining the data types, data field and data length) and accessed (defining constraints). Structured data could be queried and managed using Structured Query Language (SQL). However, with the development and growth of the Internet, organizations, and subsequently data, some data types could not fit into traditional RDBMS. These data types are categorized under unstructured and semi-structured data. Examples of unstructured data are images, audio, video, web pages, blogs and emails.

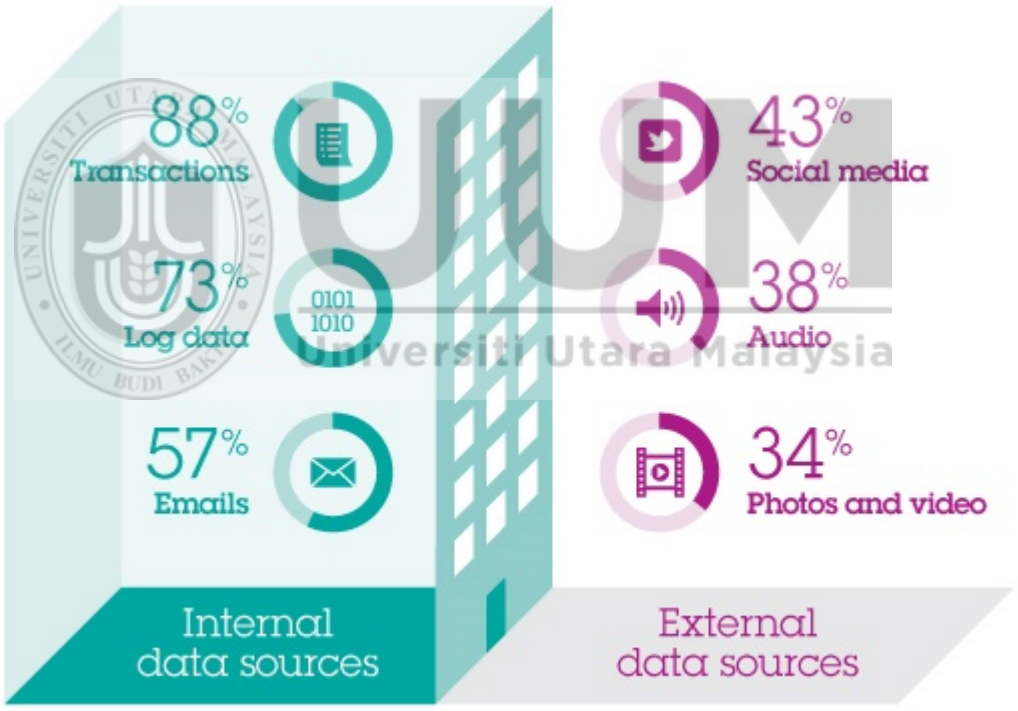

Figure 2.2: Sources of Big Data [2]

Semi-structured data is between structured and unstructured. Examples are emails, videos and word documents where sender and recipient's name, date and time (structured data) is added to the unstructured data, which is the content of the mail, video,

etc. Semi-structured data is what is mostly found on the Internet today. Big Data is sourced from end devices such as PCs, mobile devices, RFID devices, sensors, GPS devices, etc., and also online applications and social media such as social networks, forums, blogs, and applications that involve video streaming as shown in Figure 2.2. Figure 2.2 shows that log data from sensors, monitoring tools, and daily transactions, constitute the major sources of Big Data.

2.2 CHARACTERISTICS OF BIG DATA

The characteristics mentioned are further discussed as presented and argued in previous studies. The summary of the characteristics of Big Data is shown in Figure 2.3.

2.2.1 Volume

The term "Big Data" itself, suggest data of great volume. It involves a huge amount of data [33] that is impossible for a single or very few traditional machines to process and store in a long period of time [14]. According to Alnafoosi et al. in [24], over 250 Exabytes of compressed data globally stored, is increasing at a rate of 23 percent since 2007. Already stored data are used in creating more data in organizations. This depicts the volume and growth of data in our world today, which should be met with efficient processing techniques in order to facilitate storage and information retrieval [9].

2.2.2 Variety

According to Garlasu et al. [33], variety is a characteristic of Big Data that implies that Big Data consists of different types of data. That is, Big Data comes in a structured, semi-structured, and unstructured form consisting of text or numeric information from different users and organizations, images, audios or videos. With this complexity, Big Data cannot be processed and stored using traditional RDBMS [34]. More sophisti-

cated tools such as Hadoop, have been designed for the management of Big Data.

2.2.3 Velocity

Data continues to grow at a high speed because of continuous information dissemination, reuse of stored records (historical records from archives), information derived from analysis, and ongoing transactions in organizations and on the Internet. Data must be rapidly captured, processed and stored or used since it is produced in large quantity and at great speed [33]. The rate at which data is created or generated, captured, processed, stored, retrieved and analyzed (or used), is the characteristic of Big Data known as velocity. Simply put, it is the speed of change of Big Data.

2.2.4 Value

Value is the special V [15]. This attributes to its meaning which is the insight or desired outcome realized after Big Data has been stored, processed, and analyzed. Poor processing of data, can damage highly valuable data [15]. Due to this, processing of data must be done efficiently to ease information retrieval in order to achieve maximum value from analyzing Big Data..

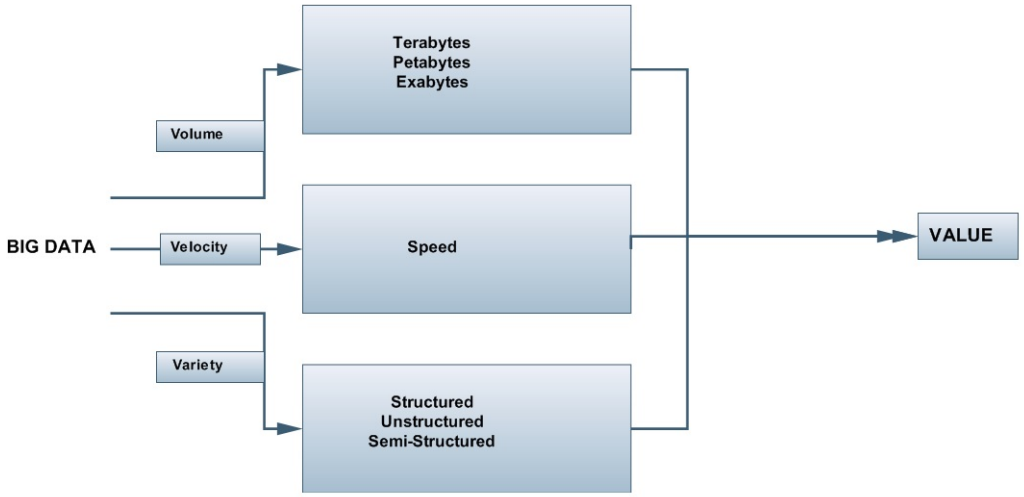

Figure 2.3: Big Data characteristics

2.3 MANAGING BIG DATA

Managing Big Data entails proper handling of data in order to achieve maximum benefit or value from it. This includes functions that must be performed on a dataset before useful information can be gotten or retrieved from it. The first is Big Data integration which consist of extraction, transformation, and loading of Big Data. The next is Big Data processing which involves indexing and MapReduce, and the final is Big Data storage.

2.3.1 Big Data Integration

Big Data integration is the process of retrieving heterogeneous data and combining them into a unified structure [4]. Big Data integration is unlike traditional data integration, because Big Data sources are wide and data are heterogeneous in structure. Big Data is extracted, combined, transformed, and stored for business or personal use [35]. Proper storage and consequently retrieval, results from proper integration of Big Data. Data collection and integration is a mandatory step in Big Data management due to the volume and variety of data sources [31]. Big Data integration consist of extraction, transformation, and loading of Big Data.

2.3.1.1 *Big Data Extraction*

Extraction of Big Data is the first phase of Big Data integration. This is the process of removing or taking a copy of Big Data from its original source. It is the process of generating or sourcing Big Data in its complex form before it is processed. Big Data can be extracted in flat file formats such as txt, csv, and xls [36].

2.3.1.2 *Big Data Transformation*

Big Data transformation involves the activities of preprocessing data to meet the functional requirements. Such activities include removing duplicates, merging, data sorting and filtering based on some requirements [36], grouping data, splitting data, compressing data, etc.

2.3.1.3 *Big Data Loading*

This phase involves the transfer of Big Data to a data storage system that supports Big Data processing and storage. From here, data will be processed according to requirements and stored. This will facilitate retrieval of information whenever the datasets are queried.

2.4 BIG DATA PROCESSING

In general, data processing is the control of data in order to generate useful information from it. Big Data processing involves maneuvering of data to gain useful information or to acquire the value in it. According to Mo et al. [37], Big Data is in strong need for processing and storage. Big Data needs to be processed before storage, to facilitate searching, information retrieval, and analysis. Searching for information from a huge amount of complex data can be a strenuous exercise if data is not processed properly [30]. In [38], Mariya stated that researchers have identified that Big Data processing techniques has to be considered to maintain searching and data access stabile.

Big Data systems must process and store gathered heterogeneous datasets, and provide performance assurance in terms of scalability, privacy, and fast retrieval [31]. Big Data has to be processed and stored efficiently to ensure easy retrieval. The issue of data processing is of concern to ensure data growth, management of existing data, and analysis of data [24]. Facebook needs to process, store, and analyze user generated

data that amounts to over 30 Petabyte [31]. Big Data is all about the value in it which can only be gained from proper processing of data, and useful information retrieval from data.

Various tools have been developed for processing and managing Big Data. Tools such as Hadoop, Spark, Sqoop, Storm, Flume, Kafka, Scribe, and S4, have been developed to handle large and heterogeneous datasets, where traditional RDBMS cannot [39]. Hadoop [7, 8] appears to be one of the most popular tools to manage Big Data for retrieval and analysis. It enables very large volumes of data to be processed and stored. It also allows for analysis of Big Data which is not possible using SQL based approaches [4]. Hadoop is scalable, which makes it an ideal tool for managing and storing growing data. Big Data volume and complexity should be met with efficient processing and storage method. Retrieval of data could become difficult if appropriate tools are not used for management (processing and storage).

2.4.1 Hadoop

Organizations are normally faced with limitations when using traditional RDBMS to manage and store large data sets. The major challenges faced in managing large data sets are scalability, analytics, security, search or information retrieval, storage, and transfer [40]. These limitations could have a big effect on business, finance, health, and other domains that utilize Big Data.

New technologies and solutions were needed to handle this type of data. A new solution was found by Google known as MapReduce – which uses a parallel processing model. Most Big Data processing tools are built upon Google's MapReduce [26]. Apache Hadoop is based on MapReduce and Hadoop Distributed File System (HDFS). Other solutions have been discovered, but Hadoop is the most widely accepted and

used.

Researches are being conducted on Big Data processing and storage using Apache Hadoop, HDFS, and MapReduce programming framework[20]. The study by Yu et al. [34] show that Hadoop is a solution to the Big Data management problems, by considering a prototype of application scenarios in Big Data.

Hadoop is an open-source, extensible program that is scalable and reliable with the ability to handle data growth in distributed systems. Hadoop enables thousands of independent computers and users to work together in a multiprogramming fashion [33]. Structurally, Hadoop comprises of multi-node cluster set-up, HDFS from Google's distributed file system, and MapReduce programming framework. Hadoop cluster usually comprises of several nodes which serves as the slaves (also known as WorkerNodes) and a single MasterNode [41]. While the JobTracker (JT) or ResourceManager, NameNode (NN), TaskTracker (TT) or NodeManager, and DataNodes (DN) all belong to the MasterNode, the TaskTracker and DataNodes represent the WorkerNodes or SlaveNodes. The JobTracker coordinates the MapReduce service to the nodes, as shown in Figure 2.4. From the study in [41], the TaskTracker collates nodes tasks, map, shuffle and reduce operations which are issued by the JobTracker. The NameNode has a Secondary NameNode (SNN) for backup in case of data loss or file corruption. HDFS on the other hand, manages the storage operations in Apache Hadoop.

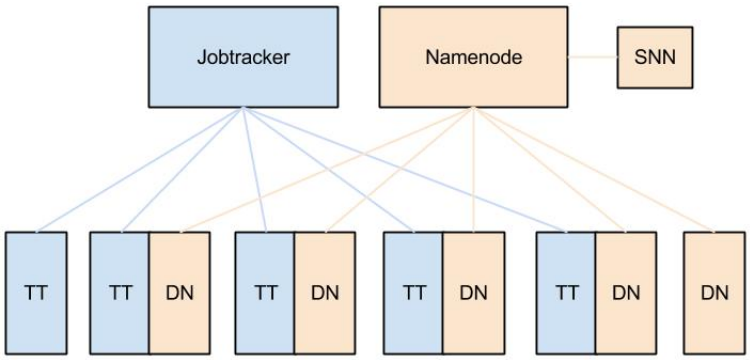

Figure 2.4: Hadoop Distributed File System Daemons [3]

2.4.2 MapReduce

This is a software framework launched by Google in 2004 to manage distributed set of data located at clustered computers [42]. In MapReduce programming, a cluster of nodes running in parallel handle computation of large amounts of data. According to Mao et al. in [42], MapReduce was originally developed by Google to scan accumulated logs.

Zhang *et al.* [20] explained that in MapReduce, a map function and reduce function is specified by the user. A key-value pair is specified that is processed by the map function. This generates or produces a set of key-value pairs. These conditions perform the intermediary function of creating partitions. The reduce function merges all the intermediate values having the same key or intermediate key. The operations are carefully handled by the nodes that serve as the slaves. Once a node processes its chunk of data, its resulted data is sent back to the MasterNode. MapReduce as described in [43], uses two approaches, the Map Step and the Reduce Step, which provide the distinct function of the operations in MapReduce.

In Map step, the input is partitioned into smaller parts or sub-problems which are distributed for the WorkerNodes or slaves. A WorkerNode can also partition the work

further into smaller chunks forming a tree structure of multi-levels. The smaller sub-problems are then shuffled and sorted out by map. The Reduce step collects the entire results generated after processing and combines them to form the output. An example of the MapReduce function is seen in Figure 2.5. In this example, the sentences are assumed to be the data input from three different servers. In the map phase, the words are split and word count taken for each sentence. In the reduce phase, the words are first shuffled with similar words grouped together. Finally, the sentences are reduced by taking the total count for each word and eliminating duplicate words.

By virtue of its scalability, fault-tolerance, load balancing, scheduling and parallelization, the MapReduce framework has become the most popular choice and the de facto standard for Big Data analysis [30]. However, the performance and implementation of a MapReduce application is affected by the characteristics of the data to be processed and the design of the algorithm [26]. The characteristic of the data to be processed include the type of data e.g emails, videos, audio, log files, etc., and what should be performed on the data e.g detecting and removing/correcting errors in data, indexing, etc. Hence, a poor algorithm results in an inefficient sorting of the keys during the map phase, which could in turn, lead to poor formation of indexes. Therefore, a good indexing strategy depends on an efficient MapReduce function. Different techniques have been established to improve the performance of MapReduce based on the software, hardware, and the framework level optimization [25]. One of the techniques used in improving the performance of MapReduce, is by tuning parameters for MapReduce execution [26]. This focuses on the design of the algorithm and the characteristics of the data to be processed. Some researchers have developed techniques, tools, and algorithms for high performance distributed and parallel computing. Ian Foster [44] is one of such researchers and is well known for his development of techniques, tools, and algorithms for high-performance distributed and parallel computing.

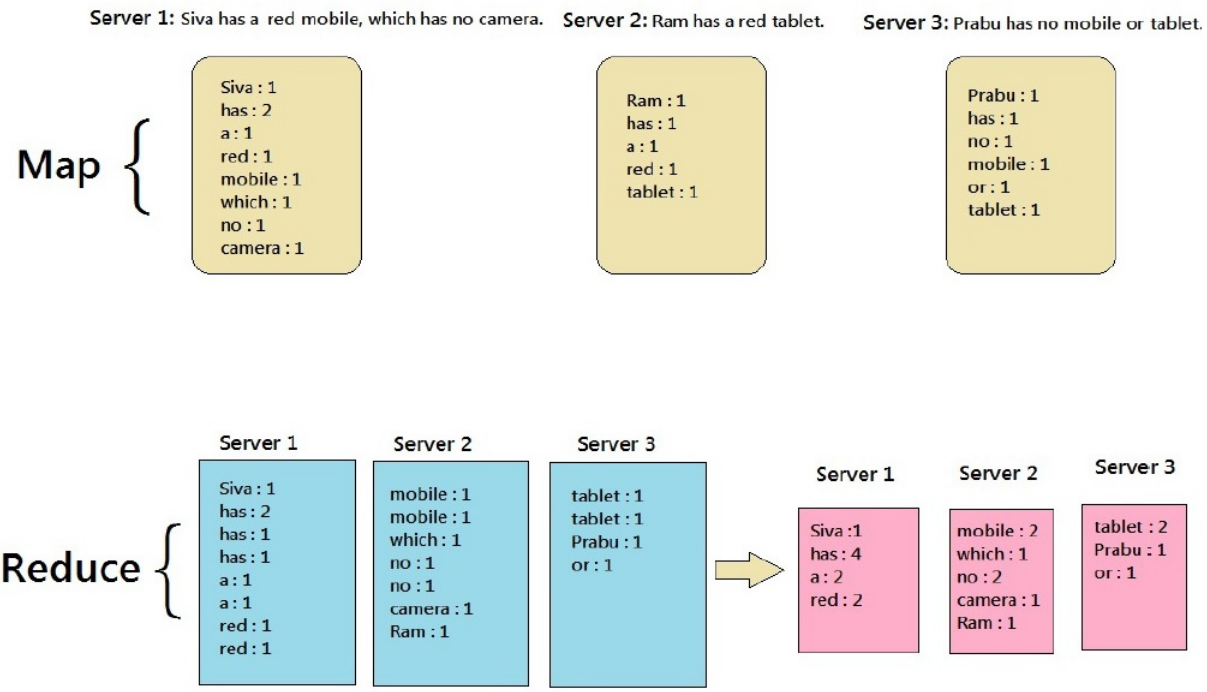

Figure 2.5: An example of MapReduce function [4]

2.4.3 Hadoop Distributed File System (HDFS)

HDFS is a Java based distributed file system designed to cover clusters of commodity hardware to provide a reliable and scalable data storage. It stores data across over thousands of servers [33, 41]. HDFS is designed to hold large amounts of data.

According to the study in [33], HDFS is managed by a NameNode server to hold the index of the file system, and secondary NameNode which can produce snapshots of memory structures of the NameNode in case of loss of data and corruption of file systems, as shown in Figure 2.4. The metadata (permissions, file names and locations, and the replication factor) for the DataNodes is stored in the NameNode. File system operations such as opening, renaming, moving and closing of files/folders are also responsibilities of the NameNode. Signals (heartbeats) are received from the DataNodes by the NameNodes. This is done to keep track or to know the states of the DataNodes.

Yu *et al.* in [45], used Hadoop HDFS and cloud computing to develop a distributed data storage and management service that supports massive traffic data. This depicts the capability of HDFS in terms of Big Data storage. A reliable storage solution that supports Big Data processing, is a necessity when managing Big Data. HDFS is very scalable, and supports Big Data processing and storage [45].

2.5 BIG DATA INDEXING

In general, indexes or indices are a list of tags, names, subjects, etc. of a group of items which references where the items occur. With this, Big Data indexes can be said to be a list of tags, names, subjects, etc. of a dataset which references where data can be found. An indexing strategy is the design of an access method to a searched item, or simply put, an index. It also describes how data is organized in a storage system to facilitate information retrieval. The idea of Big Data indexing is to fragment the datasets according to criteria that will be used frequently in query [27]. The fragments are indexed with each containing values satisfying some query predicates. This is aimed at storing the data in a more organized manner, thereby easing information retrieval. The growth of data and accumulation of complex data collections have become a challenge for information retrieval [27]. This necessitates the design of an indexing strategy for the processing of Big Data.

Complex data are collected with metedata that describes their contents. Such datasets can be queried using the metadata of the contents. Instead of searching the whole database (which can be time consuming), a more efficient approach is to search the appropriate group(s) relating to the query. This results in a decrease in information retrieval time, since the search process considers only the content of a specific group(s). To facilitate information retrieval, a suitable indexing strategy have to be applied to the datasets during processing. This also comes with the advantage of having an or-

ganized storage system to ease search and information retrieval. An indexing strategy thereby depends on a solution that utilizes a massively parallel computer or machine that interconnects lots of RAM, CPUs, and disk units. The benefits of this are high throughput for data processing, increase access time for queries, data replication that results in increased availability and reliability, and scalability of the structure. The design of an access method or the type of indexing strategy to be used in processing a specific dataset depends on the type of queries that will be performed on the dataset, such as similarity queries (nearest neighbor search), range queries, point query, keyword queries, and ad-hoc query, as explained below:

1. Point Query: Point query is also known as equality query. It is used to find data items similar to the searched item or key.
2. Multi - key Query: Multi - key queries are queries on a set of indexed keys, for example, arrays.
3. Range Query: Range queries carry minimum and maximum values. They are used to query multi-dimensional or spatial data (e.g geographical data or coordinates.
4. Ad-hoc Query: These are user defined queries on a data set based on custom or arbitrary fields or indexes.
5. Nearest Neighbor Search (NNS): Also known as similarity or proximity search is used in finding the closest item to the searched key.
6. Keyword Query: In a keyword query, the keys (or terms) entered by the user are literally used in retrieving documents or files containing those keys.

Therefore, the designer must be aware of the type of data to be indexed (e.g. logs, email, audio, video, images, etc.) and the type of query that will be performed on the indexes. A brief description of the types of data mentioned in this study, will

simplify understanding of the indexing strategies and the type of data and query each supports. There are two categories of data in the language of research; quantitative and qualitative [46]. Quantitative is the usual form of defining data, which is mostly, the numerical form. Examples are integers, float, Var, and VarChar. Qualitative on the other hand, is not the numerical form of defining data, but can easily be converted into quantitative. It's a way to appropriately describe data as it is used in a specific research context. Examples of qualitative data are text, images, audio, video, etc. Qualitative data will be used in the study. Examples of such are:

1. Multimedia data: Multimedia data consist of video, images, maps, audio, animation, etc.
2. Log data: These are textual data which includes very large collections of alphabets, numbers, and characters, recorded or compiled over time from sensors, monitoring systems, weblogs, transactions, etc. Examples of the types of data collected includes application logs, server logs, event logs, weather and temperature measurements, network pings, vibrations, light intensity, sound frequencies, pressure, and lots more. It also includes multimedia data or any kind of data that is accumulated and translated into textual form.
3. Spatial data: Spatial data includes optical characters, coordinates or geographical data such as maps, longitude and latitude position, etc.

Many indexing strategies have been proposed [47, 26, 27, 28, 25, 9, 5, 29, 30, 48]. According to Gani *et al* [49], indexing strategies can be categorized into Artificial Intelligence (AI) approach, and Non-Artificial Intelligence (NAI) approach.

2.5.1 Artificial Intelligence Approach

Artificial Intelligence (AI) indexing approaches are so called because of their ability to detect unknown behavior in Big Data. They establish relationships between data items by observing patterns and categorizing items or objects with similar traits. Latent Semantic Indexing (LSI) [50, 5, 51] and Hidden Markov Model (HMM) [52, 53] are two popular AI indexing approaches.

2.5.1.1 *Latent Semantic Indexing*

Latent Semantic Indexing, LSI for short, is an indexing strategy (retrieval/access method) that identifies patterns between the terms in an unstructured data set (specifically, text). It uses a mathematical approach known as Singular Value Decomposition (SVD) for the pattern or relationship identification. Hence, LSI is not subjected to any language. The main characteristic of LSI is the ability to elicit the conceptual (semantic) content of data sets and to establish relationships between terms with similar contexts as illustrated in Figure 2.6. In Figure 2.6, the audiences discuss the program "House of cards" on social media and forums. LSI is used here, to categorize or index comments made by the audience into audience favors, audience expectations, and the shortcomings of the season (by extracting the meaning of each comment). This makes it easier for the director to make decisions towards the improvement of the next season, and so on. LSI makes use of Resource Description Framework (RDF), which is a standard for web resource description. The RDF can describe author, title, date, time, price, definition, and a lot more information of a web page. RDF also uses tags, by adding information about parts of speech such as noun, verb, adjective, etc. to the context of each word. Along with establishing meaningful relationships between texts, LSI overcomes the problem that comes with keyword queries (synonyms and polysemy) [50], mostly encountered while working with inverted indexes. These problems often result in mismatches during information retrieval and can be bad for

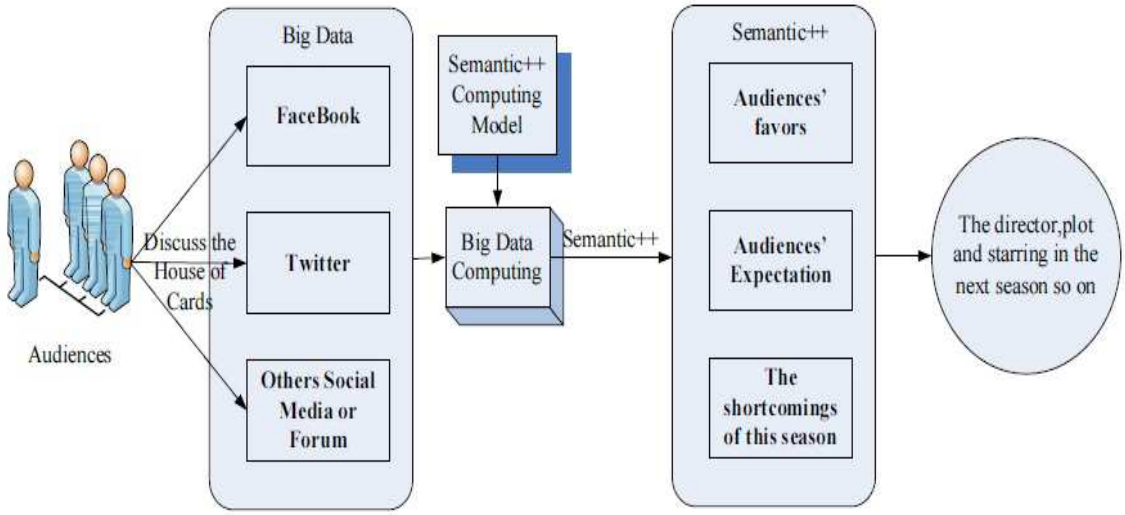

Figure 2.6: An example of Latent Semantic Indexing [5]

decision making. In LSI strategy, text or documents are assigned to categories according to their contextual similarities. During categorization, the contexts of the set of text to be categorized are compared to the contexts of example document, and categories are assigned based on matching documents. Also, documents can be grouped together based on their contextual similarities, without comparing with example documents. The challenges mostly faced while working with LSI is scalability and performance [51]. LSI strategy demands very high computational performance as well as memory to index Big Data. LSI supports keyword queries on textual data which can be in the form of web contents (images, audio, etc.), documents, emails, or any item that can be converted into text.

2.5.1.2 *Hidden Markov Model*

The Hidden Markov Model (HMM) indexing approach is an access method developed from the Markov model. A Markov model is made up of states which are connected by transitions,where future states are solely dependent on the present state and independent of historical states. Similar to the LSI strategy, the HMM uses pattern recognition and relationship between data. In the HMM indexing approach, data or characteristics which the states depend on during query, are categorized and stored in advance. The

query results are usually prediction of future states of an item, based on the current or present state. The present state is used to predict the future states using the dependent data or characteristics of the states. For example, in the study by Matsui *et al.* [52], HMM was used to classify and store motion data used by robots. The classification was based on the acceleration information,consisting of the position information and the pure force. Hence, the prediction of the next series (sequence) of motions was dependent on the position informant and the pure force. The motion data is stored and classified in advance, before the quick search for motion is conducted. Also the study by Widodo et al. [53] used HMM and LSI to classify or index documents. In their work, words are expanded in every documents, and stored in advance. Expanding the documents prior to indexing gives more room for prediction and quicker search on items.

Although AI indexing approaches have an edge over NAI because the meaning of the data items are considered, the former generally takes more time in information retrieval and are sometimes considered inefficient as compared to NAI indexing approaches [49]. Hence, the study is focused more on NAI indexing strategies.

2.5.2 Non – Artificial Intelligence Approach

In Non-Artificial Intelligence (NAI) indexing approach, the formation of indexes does not depend on the meaning of the data item or the relationship between texts. Rather, indexes are formed based on items most queried or searched for in a particular data set. NAI indexing approaches are the tree-based indexing strategy (B-tree [54, 55, 56], R-tree [57][58], and X-tree [59]), inverted indexing approach [30, 25], hash [29], and custom (GiST [60] and GIN [61]) indexing strategies.

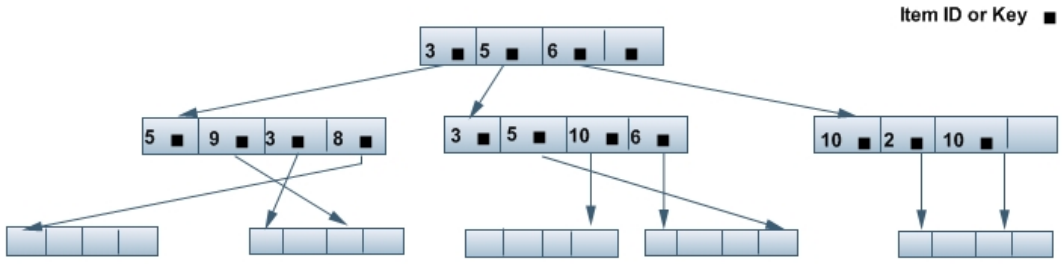

Figure 2.7: An illustration of a B-tree

2.5.2.1 *The Tree-based indexing strategies*

The Tree indexing structures are the B-tree, R-tree, M-tree, X-tree, etc. [29]. In the Tree indexing strategy, retrieval of data is done in a sorted order , following branch-relations of the data item. This satisfies nearest neighbour queries. According to researches, the Tree indexing strategies are being displaced by other indexing strategies because they are generally outperformed (in terms of speed of information retrieval) by simple sequential scans [29]. The Tree-based indexing strategies are explained as follows:

1. The B-Tree: A B-tree works like the Binary tree search, but in a more complex manner. This is because the nodes of B-tree have many branches, unlike the binary tree which has two branches per node. So a B-tree is more complicated than a binary tree [62]. B-tree indexes satisfy range queries and similarity queries also known as Nearest Neighbor Search (NNS), using comparison - operators (<, < =, =, >, > =). In the B-tree, the keys and all records are normally stored in leaves, but copies (of the key) are stored in internal nodes as illustrated in Figure 2.7. Also, the leaves might include pointers to the next node, showing the path to the searched item.

   Researches have shown that this strategy is not always fast when searching Big Data and can waste storage spaces since the nodes are not always full [49, 54].

A B-tree scales linearly, but is only suitable for one dimensional access method unlike other tree-based access methods or indexing strategies such as the R-tree. Also, the B-tree algorithm consumes huge computing resources when performing indexing on Big Data [54]. Other variations of the B-tree are the B+tree [55], B*tree, KDB-tree [56], and so on.

2. The R-Tree: This is an indexing strategy used for spatial or range queries.It is mostly applied in geospatial systems with each entry having X and Y coordinates with minimum and maximum values [57, 63]. The advantage of using an R-tree over a B-tree is that, R-tree satisfies multi-dimensional or range queries, whereas B-tree does not. Given a query range, using R-tree makes finding answers to queries quick [64]. An example is finding all the hostels within a given campus, or finding all hotels within a given kilometer from a certain location. The idea is to group data items according to there distance from each other (Figure 2.8 and Figure 2.9), and assign minimum and maximum bound to them. Each record at the leaf node, describes a single item (with minimum and maximum values). Each internal node describes a collection of items or objects.

   Though the R-tree is preferred over the B-tree in the case of indexing spatial data, the R-tree does not find the exact answer as query results. It merely limits the search space. Also, it consumes memory space because coordinates are stored along with the data [65]. Variants of the R-tree are R*tree, R+-tree [58], etc.

3. The X-Tree: This type of indexing strategy, based on the R-tree, satisfies range query. The X-tree is similar to the Rtree and operates just like the R-tree. Although, unlike the R-tree which satisfies 2-3 dimensional range queries, the X-tree satisfies queries of many dimensions [29, 59]. This implies that the X-tree is a more complicated version of the R-tree. The advantage of the X-tree over the R-tree is that it covers more dimensions, otherwise, the X-tree also consumes

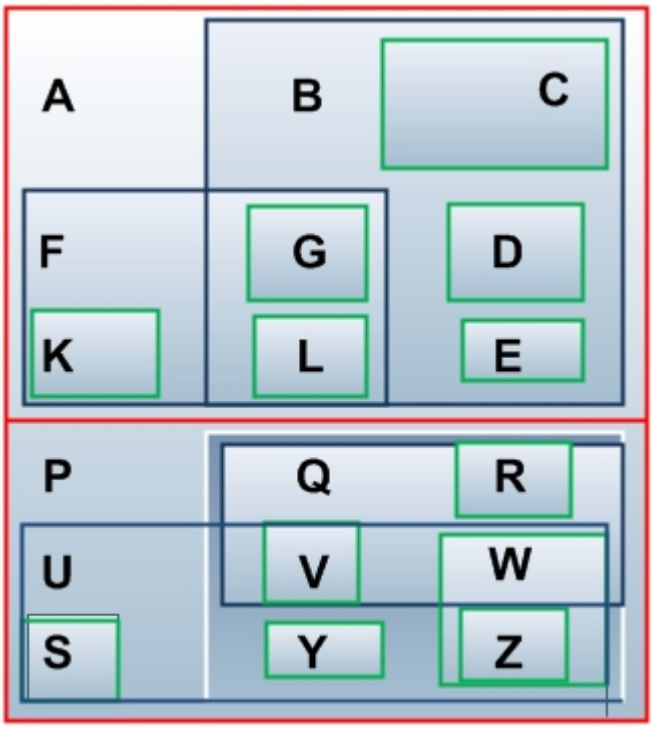

Figure 2.8: Range Grouping in R-tree

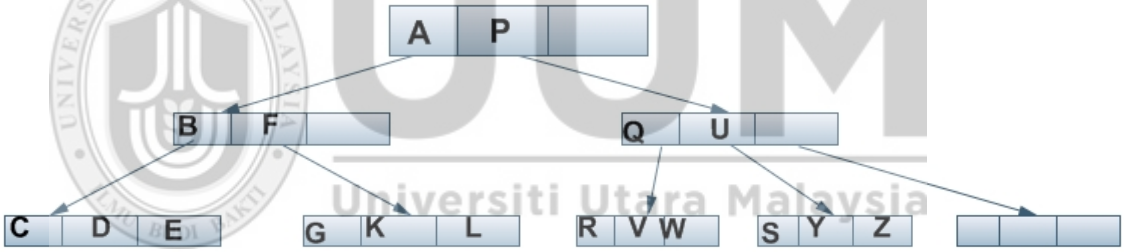

Figure 2.9: Nodes in R-tree

memory space due to storage of coordinates.

2.5.2.2 *Hash Indexing Strategy*

Hash allows for equality comparison. Hash indexing accelerates information retrieval by detecting duplicates in a large dataset [66]. An example of an hash indexing strategy implementation is explained in the study by Giangreco et al (2014) [29]. Giangreco et al designed a strategy that takes hand sketched diagrams and retrieves similar images from a large collection of images, based on hash indexing strategy. Hash is used in Big Data indexing to index and retrieve data items (in a dataset) that are similar to the

searched item such as in password checking systems, DNA sequence match, etc. It uses a hashed key (which is computed by the hash function and usually shorter than the original value) to store and retrieve indexes. For this reason, hash indexing is more efficient than the tree-based indexing in terms of equality or point query [29]. Simply, search on shorter hashed keys can be faster than search on unpredictable length key (found in tree-based indexes). Though, hashing technique works fine with limited data size, it tends to exhibit indexing computational overhead as data size increases or with a very large data size [49].

2.5.2.3 *Custom Indexing Strategy*

Custom indexing supports multiple field indexing based on arbitrary or user defined indices [28]. They are usually based on indexing strategies such as B-tree, R-tree, inverted index, and hash indexing strategy. Two types of custom indexing strategies are Generalized Search Tree (GiST) [60] and Generalized Inverted Index (GIN) [61].

1. GiST Indexing Strategy: The Generalized Search Tree or GiST indexing strategy, is an indexing strategy based on the B-tree or the R-tree [60]. It allows for the creation of custom or arbitrary fields as indexes. The GiST has same implementation (for indexing and retrieval) as the R-tree for those based on the R-tree, and as the B-tree for those based on the B-tree. Hence, they support indexing and query on one-dimensional data, as well as multi-dimensional or spatial data. Just like every other tree-based indexing strategy, GiST has root nodes, leaf nodes, pointers, and other characteristics of a balanced tree structure. Despite these similarities between the GiST and the tree-based strategies, the former has an advantage of supporting ad-hoc queries over the latter. Taking GiST based on R-tree for example, each node contains a key-pointer pair, where the key is the searched key, and the pointer points or refers to the corresponding

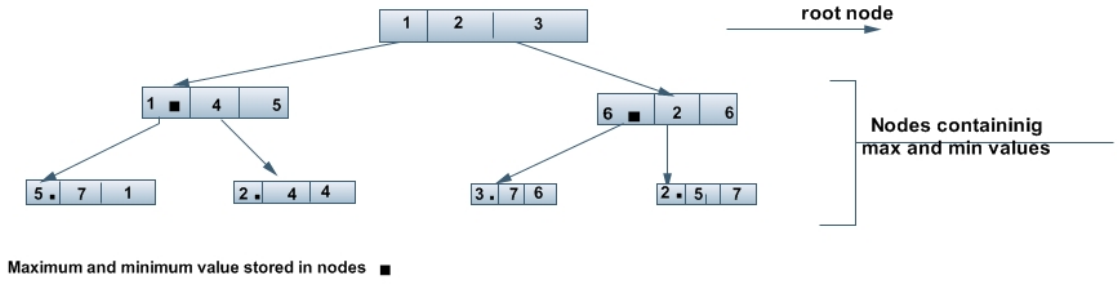

Figure 2.10: Nodes in a Generalized Search Tree

node (or data item, in the case of a leaf node) as illustrated in Figure 2.10. Also, each node contains minimum and maximum values, with the exception of the root node. The GiST performs well in terms of query search, but is considered generally slower than the GIN [60].

2. GIN: Just as in the GiST, the Generalized Inverted Index or GIN indexing strategy (or access method) uses custom or arbitrary fields as indexes [61]. It is designed for specific user requirements. Though the GIN is implemented like the B-tree and has properties of the inverted index, GIN differs from B-tree which have comparison-based operations that are predefined. GIN is made up of a B-tree index which comprises of Entries Tree or list (ET) and Posting Tree (PT) or Posting List (PL) [61]. In the ET, each entry represents an element of the searched key or indexed value, for example, arrays. The PL is a pointer to a list of items, or a pointer to a B-tree (for leaf nodes), in which case it is called a PT, as illustrated in Figure 2.11. While the B-tree is good for single-match indexes or range queries, the GIN works best for indexes having many duplicates. This is because the GIN queries data only by point or equality matching. This is often viewed as a limitation.

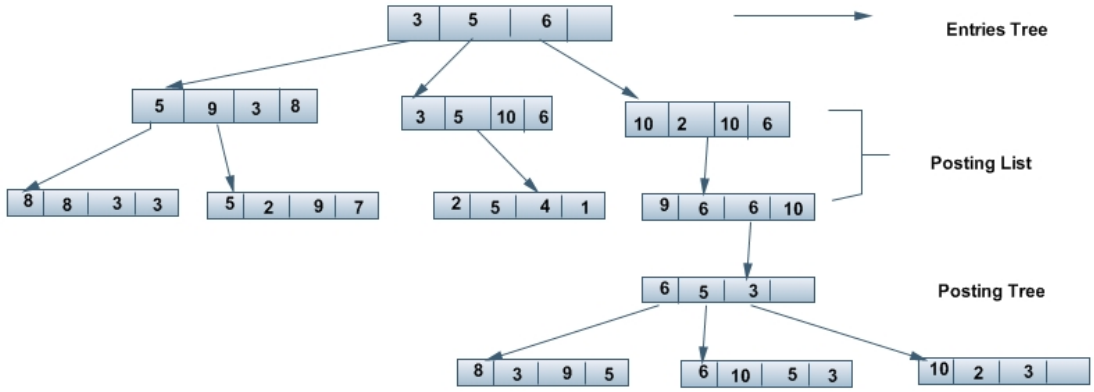

Figure 2.11: Generalized Inverted Index with Posting Tree

2.5.2.4 *Inverted Indexing Strategy*

Inverted indexing strategy allows for the design of inverted indices which are used for full-text search like what is obtainable in Google and other search engines [67, 30, 25]. It supports keyword queries. An inverted index is made up of a list of all unique words which appear in documents, and a list of documents in which each word appears. With an inverted index, multiple documents can have the same key as index. Also, multiple keys can be used in indexing a document. For example, a blog post can have multiple tags (as key), and each tag can refer to more than one blog posts. Though some inverted indexing strategies use B-tree or can be populated with rows, it is worth noting that not all inverted indexes are based on B-trees. The difference between an inverted index and a B-tree is that B-trees use row-structured data, unlike inverted indexes. Inverted indexing is implemented by storing or indexing a set of keypost list pairs, where key is the searched index, and post list is a collection of documents where the key occurs. The problem with inverted indexes is that, two or more words (keys) might be separate terms, but will appear to the user as the same term. Also, synonyms (of the keys) might not be recognized or retrieved during query search [67].

In general, most of the existing indexing strategies work fine with data sets of small (or fairly large) amounts, but incure computational overhead when implemented on very

Table 2.1: Indexing Strategies and Query-Types

Indexing Strategies	Data-type	Query-type
B-tree	Log data, multimedia data	Range queries, similarity queries (NNS): 1 dimension
R-tree	Spatial data e.g Geographical coordinates, multimedia data	Spatial or range query: 2-3 dimensions
X-tree	Spatial data	Spatial or range query: Multiple dimensions
Hash	Log data, multimedia data	Point query (Equality search)
GiST	Spatial data, log data	Range query, ad-hoc query
GIN	Log data, spatial data	Similarity query, ad-hoc query
Inverted	Multimedia data, documents	Keyword queries
LSI	Multimedia data, spatial data (textual data)	Keyword queries
HMM	Multimedia data, unstable signals etc.	Ad-hoc query

large amounts of data [49]. Researches have proposed solutions using the MapReduce framework, to solve the problem. Although, most of the solutions proposed concentrated more on the mapper and reducer phase.

2.5.3 Comparison of Indexing Strategies

Table 2.1 summarizes the various types of indexing strategies, along with the possible type of data and queries they support. Table 2.2 outlines the main characteristics of each indexing strategy. The key features and challenges faced in the use of each of the indexing strategies, are also described on Table 2.2.

The taxonomy of the popular indexing strategies used in Big Data is as illustrated in Figure 2.12. The two main categories are AI and NAI. AI utilizes the contextual meaning of data to establish relationships (between the data items) which serves as the bases to index creation. The most popular AI indexing approaches are LSI and

Table 2.2: Characteristics of Indexing Strategies

Indexing Strategies	**Properties**	**Challenges**
B-tree	- One dimensional access method -Tree structure with nodes and pointers - Scales linearly	- Waste storage space - Not suitable for multidimensional access -Consumes huge computing resources
R-tree	- More scalable than the B-tree - 2 to 3 dimensional access method	- Index consumes more memory space
X-tree	- Multidimensional access method	- Consumes memory space
Hash	- Presents the exact answer (uses '=' operator) - Quick information retrieval	- Computational overhead
GiST	- Arbitrary indexes - Based on the tree structures	- Slower query response
GIN	- Arbitrary indexes - Based on the tree structures	- Longer processing time
Inverted	- Index consumes less space - Full text search (keyword search)	- Longer data processing time - Limits the search space, not necessarily producing the exact answer - Can present wrong answers due to synonyms and polysemy
LSI	- Uses data and meaning of data for indexing - Presents accurate query results (since it uses more information)	- Demands high computational performance - Consumes more memory space
HMM	- Based on the Markov model - Recognizes relationships between data	- Demands high computational performance

HMM. NAI on the other hand, does not detect the unknown behavior of data [49]. The most popular NAI techniques are B-tree, R-tree (and their variants), Hash, Inverted, and custom indexing.

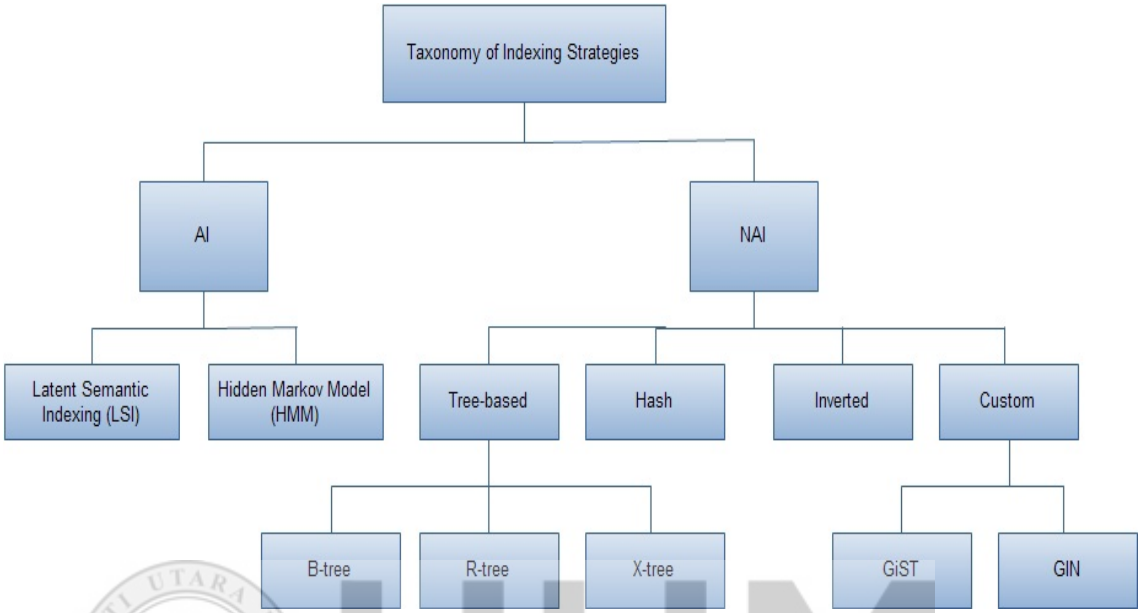

Figure 2.12: Taxonomy of Indexing Strategies

2.6 IAN FOSTER'S TASK-SCHEDULING CONCEPT

MapReduce operates on distributed computers and in a parallel fashion. Some researchers have developed techniques, tools, and algorithms for high performance distributed and parallel computing. Ian Foster [44] is one of such researchers and is well known for his development of techniques, tools, and algorithms for high-performance distributed and parallel computing. This renders his concepts suitable for application in cases of distributed and parallel computing such as MapReduce. One of such known concepts is Ian Foster's Task-Scheduling concept [44] which explains that in parallel processing, processes who have completed task execution should return to the master for further allocation of task without waiting for other processes to complete task execution. This could be applied to processing of Big Data using the MapReduce framework.

As mentioned earlier, Big Data can be processed in numerous ways, depending on the type of queries to be performed on it. Based on the type of data sets (log data) used in this study, keyword queries will be performed on the data sets. From Table 2.1, it is evident that the Inverted Indexing technique supports keyword queries. Hence, the Inverted Indexing technique was chosen as the most suitable technique to be deployed in the study. In the Inverted Indexing technique, the whole MapReduce process is divided into:

- The input phase - where the data sets are divided into fragments and records are assigned to mappers for processing
- The mapper phase - where the keys are sorted out
- The reducer phase - where the keys and values are combined

Most of the related works concentrate more on the mapper and reducer phase. With this, less attention is paid to the input phase, which consist of the InputFormat and the RecordReader. The current InputFomat's RecordReader as used in the study by Gollub *et al* [30], divides the data sets into splits and assigns ones record to each individual mapper as task. The mapper sorts the most frequent library queries as keys, and the set of library documents as values. The RecordReader assigns to task to the mappers simultaneously, after the previous task has been completed. This is done until the data sets have all been allocated to the mappers for processing. The reducer finally collects these keyvalue pairs from all the mappers and combines them as output. They successfully indexed the data sets, resulting in easy information retrieval. The drawback with this approach is that it incures computational overhead especially with larger data sets, since it concentrates on the MapReduce phase only. Also, the mappers process a single record at a time and are assigned more task simultaneously. This slows down the processing time, and consequently slows down the time it takes to retrieve

information. Ian Foster's Task-Scheduling concept could be applied to the way the records are assigned to mappers, hence, reducing processing or task execution time, and consequently reducing the time it takes to retrieve information from the data sets.

2.7 BIG DATA STORAGE AND INFORMATION RETRIEVAL

2.7.1 Big Data Storage

Inforamtion retrieval is being challenged by the huge amounts of heterogeneous data, related to Big Data [48, 26, 27, 28, 25]. The storage and retrieval of information in an efficient way, is a rising problem in Big Data. This is because data of very large quantity such as video, audio, logs, and images, are collected across different domains such as social networks, defense systems, health care, security, marketing, agriculture, entertainment, and lots more. Organizations are encountering challenges with Big Data storage and retrieval [23]. The solution is in using tools and designing strategies that responds to the need of easy retrieval for future analysis. Organizations need an efficient indexing strategy to enable an organized manner of data storage for quick information retrieval when needed for analysis and decision making [8]. HDFS, HBase, and other similar tools have been designed to support the processing and storage of Big Data efficiently.

2.7.2 Information Retrieval

Big Data is studied and analyzed for the useful information that can be retrieved from it, in other word, the value in it. This necessitates seeking a proper means of organizing datasets in order to facilitate information retrieval. Identifying the key to be indexed is about understanding and having knowledge of the type of queries that will be performed on a particular datasets to retrieve a data item (s) or information. Information retrieval from a collection of complex datasets becomes easy, if processing of the datasets is efficiently performed. According to Fasolin et al (2013) in [27], infor-

mation retrieval takes lesser time with an indexed database. Data access, search, or retrieval of information from an indexed dataset takes three steps [27] :

1. Finding fragment(s) (index or indexes) with data that satisfies some query predicate(s)
2. Filtering the data in the selected fragment(s) based on the predicate(s) connected to the fragment or index.
3. Producing the results generated from each fragment

An example of information retrieval problem is retrieving Shakespeare's plays containing the word Brutus AND the word Ceasar but NOT the word Calpurnia, from the large collections of Shakespear's works [68]. One way of achieving this, is by starting at the beginning of every play and reading through all the text, as well as noting or being mindful of whether a play (for each play) contains Brutus and Ceaser, and not considering plays that contain Calpurnia - even though they might also contain the words Brutus and Ceasar. Indexing strategy is an easy means of facilitating information retrieval that suits this form of queries.

2.8 IEPM PINGER

This study will be conducted using PingER log data. This section elaborates on PingER, and describes log data as an example of Big Data.

2.8.1 ICMP PING

Large amounts of log data acquired from pings, is an example of Big Data. Ping (Packet Internet Groper) is an Internet program or a command that enables a user to verify the existence of an host, and to check if it can accept requests [69]). A

user uses the ping command to get the Round Trip Time (RTT) or to know how long it takes to receive a reply after pinging a host. Ping is also used to know if an IP address is working. Ping operates by sending a data packet – the Internet Control Message Protocol (ICMP) echo request packets – to a host or a particular IP address, and waiting for a reply (echo-reply) or time-out. ICMP is used to report error messages if a requested host or IP address is not available or reachable. Ping is generally used to ensure the availability and reachability of the host that is being reached, and to observe packet loss, RTT, Jitter, duplicate packets and throughput performance. The data from the observations are stored, and can be used for future analysis.

The benefits obtained from the proper storage of ping results have been overemphasized. Areni et al. in [70], used the accumulation of ICMP packets generated by pings to maximize the throughput performance of a channel by detecting signal interference with radio services due to radiation. Also, Quang et al. [71], used ping to test the availability and reachability of several applications in a smart grid communication system. The results gathered from the test were used to evaluate the performance of the applications or technology. Furthermore, Yang et al. [72], proposed an approach for testing the Quality of Service (QoS) of networks (to improve on it, thereby, maximizing customer satisfaction), using ICMP ping command or function.

2.8.2 PINGER / SLAC

PingER is an acronym for Ping End-to-end Reporting or Packet Internet Grover End-to-end Reporting. It is an Internet End-to-end Performance Measurement (IEPM) project that does end-to-end performance monitoring of Internet links [73]).

PingER is led by the Stanford Linear Accelerator Center (SLAC), presently known as SLAC National Accelerator Laboratory. From 1962 – 1995, SLAC was known

for their researches on particle physics [74, 73]. However, it focuses more on Internet performance of late [73]. PingER's development includes National University of Sciences and Technology / School of Electrical Engineering and Computer Sciences (NUST/SEECS) – formerly NUST Institute of Information Technology (NIIT) Pakistan, Fermi National Accelerator Laboratory (FNAL), International Centre for Theoretical Physics/Trieste (ICTP/Trieste), Universiti Malaya (UM), Universiti Malaysia Sarawak (UNIMAS), and University Teknologi Malaysia (UTM) [73]. Recently, Universiti Utara Malaysia (UUM) joined PingER in 2014.

PingER measurements now cover over 700 sites found in over 160 countries, and are actively searching for new sites to monitor, monitoring sites, and people interested in their data [73].

The first Malaysian PingER MA installation was conducted in UNIMAS, by the Faculty of Computer Science and Information Technology (FCSIT), UNIMAS, in December, 2012. Subsequently, UTM and UM also carried out the installation of the Reverse Traceroute Server and PingER MA in January, 2013.

The results (logs) gathered and stored from the PingER MAs, are used to make case studies related to Internet performance. Currently, PingER stores the log data in flat files. This makes retrieval of the data or information, a cumbersome exercise.

2.9 FLAT FILE

A flat file is a table which contains one record per line. Data in flat files are stored in plain text format. Also, the data are not associated and do not have folders. In a flat file, the first row holds the field names which are distinct for easy identification of data contained in each field. Fields in flat files are demarcated using delimiters such as

tab or commas, and are restricted to certain data types. Flat files hold data which are in their original state, until they are transmitted to RDBMS, HDFS, or other storage systems where they will be processed and stored.

Even though the data stored in flat files are unstructured, many programs still prefer flat files [18]. This is due to the ease of transmission, which is related to flat files [75]. Despite this, flat files still need to be converted to a unified form[75] to allow for easy information retrieval. Retrieval of data and analysis tend to be difficult when using data in their original form. Currently, PingER stores large amounts of historical log data in flat files. This data will be transmitted to Hadoop, where it will be processed and stored for future analysis and decision making.

2.10 RELATED WORK

Several studies have been carried out, and various strategies have been proposed on Big Data processing. Mo et al. in [37] used Asynchronous Index Strategy (AIS) and partial replication as a solution to Big Data information retrieval. In the study, mongoDB was used to store stream data. The result was a high performance solution with improved execution performance, improve retrieval strategy, and data replication that assures consistency without failure. However, an increase in computational overhead can result from this strategy due to series of MapReduce activities.

Giangreco et al in 2014 [29] tried to solve the problem of information retrieval from multimedia data. They proposed the design of a multimedia retrieval system based on an indexing strategy that allows similarity search and boolean retrieval. Their approach uses hash indexing strategy based on the MapReduce framework. The draw back of the approach is its limitation to equality search and computational overhead due to numerous MapReduce activities.

Another study on Big Data indexing is by Gollub et al in 2014 [30]. Their study designed an indexing strategy based on library taxonomy system for information retrieval. The approach used the inverted indexing technique, taking queries as keys against sets of library documents. Although the method was successful, it still creates computational overhead as a result of multiple MapReduce phases.

Grunzke et al in 2014 [25] proposed a design of a metadata management strategy to facilitate data search across communities. Their approach used the inverted indexing technique based on the MapReduce framework to retrieve information from data on organisms, molecules, and physical events (biological data). Although the processing time was minimized, it still incurred computation overhead due to series of MapReduce activities.

Zhang et al in 2014 [5], Irudeen et al in 2013 [28], Zhou et al in 2013 [9], Chung et al in 2013 [26], Fasolin et al in 2013 [27], and several other studies were conducted on the processing of Big Data with each differing in terms of the type of data used, contribution, and based on varying indexing strategies and MapReduce activities, as illustrated on Table 5.15.

2.11 SUMMARY

This chapter presents Big Data by classifying it into structured and unstructured data. Structured data are those that conforms to the rules governing RDBMS, while unstructured data such as email, images, audio, video, and log data do not fit into RDBMS. The chapter also explains Big Data integration which is sub-divided into Big Data extraction, Big Data transformation, and Big Data loading. Furthermore, it explains Big Data processing which is based on the MapReduce framework and uses HDFS as a storage unit. Finally, it describes Big Data indexing as related in previous studies. The

indexing technique to be used in processing a particular data type depends on the type of query to be performed on it. In this case, the Inverted Indexing technique is the most suitable since keyword queries would be performed. Most of the related works concentrated more on the MapReduce phase than the InputFormat, hence incuring computational overhead due to prolonged processing time. Figure 2.13 is a literature map for the study. The following chapter (Chapter Three) explains the methodology applied in the study.

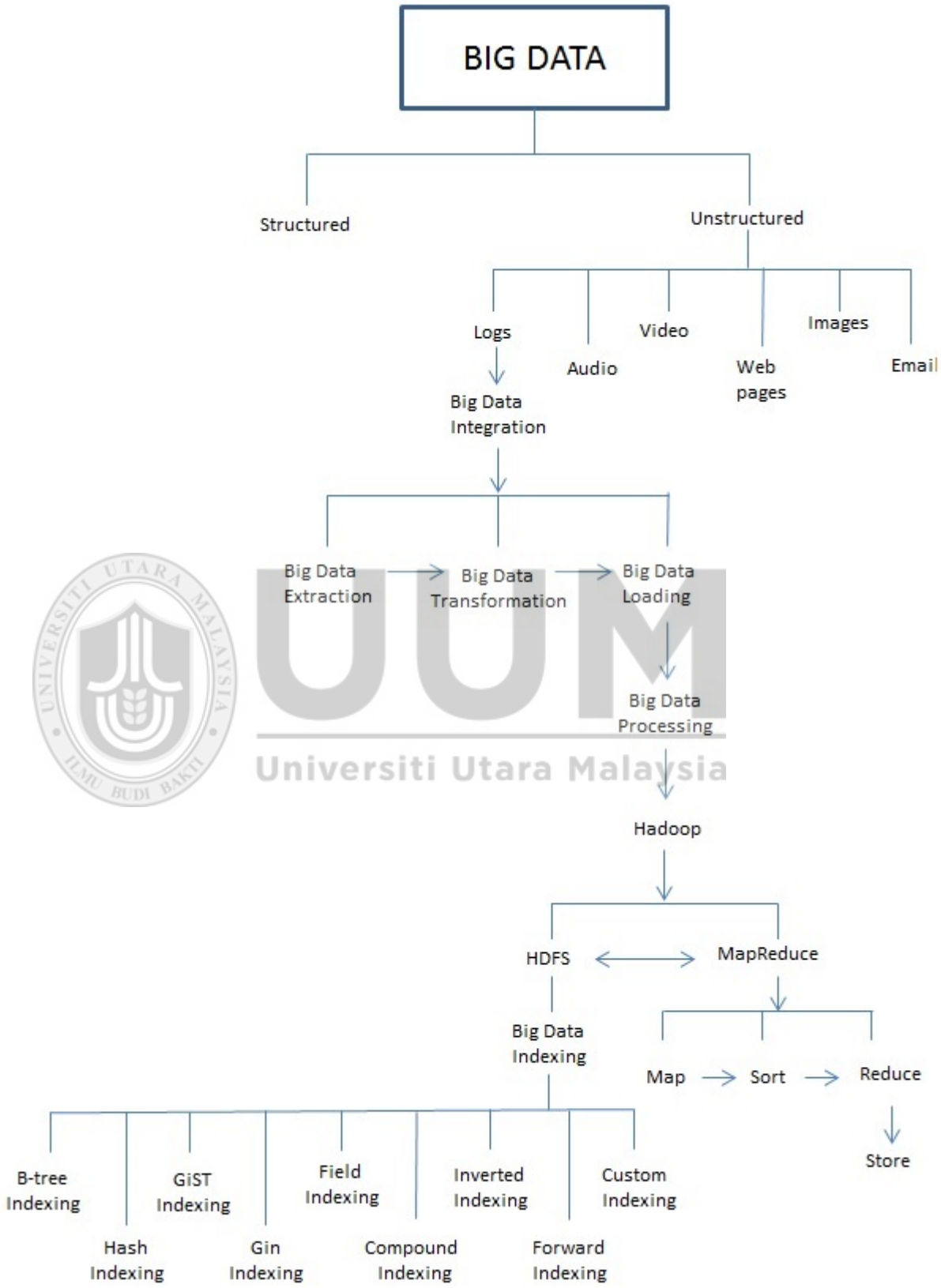

Figure 2.13: Literature Map

CHAPTER THREE
RESEARCH METHODOLOGY

This chapter explains the phases in the study and the design strategy that was adopted in the study. It includes the procedures and approach that was used in the study. The chapter also elaborates on the steps that lead to achieving the research objectives. The study proposes the design of a Big Data indexing strategy called *BIND (Big Data IN-Dexing*) Strategy, that facilitates the processing and storage of large amounts of data to ease information retrieval. The activities that lead to the achievement of the objectives have been grouped into phases as described on Figure 3.1. Phase one is the study review. In this phase, the literature review was conducted in order to understand the concept, related works, and previous strategies used. Phase two is data transmission, where transmission of Big Data samples and analyzing of data sets is done. The third phase is data conversion or preprocessing of data. Phase four is the design scheme. It is the most important phase consisting of three steps. Step one is designing the proposed strategy, step two is loading Big Data samples, and step three is verification of code. Phase five is data control, where the remaining datasets will be processed (batch processing). The code was also validated in this phase. Phase six is where result analysis and conclusions were made.

3.1 PHASE ONE: STUDY REVIEW

This study started with reviewing of past literatures, to acquire precise understanding of the study, as well as a research plan that was implemented as phases in the study. This was achieved by iteratively following the steps below:

1. Identifying topics or areas of interest
2. Choosing interested area and topic of interest

3. Comprehending area of study
4. Determining the area of focus
5. Determining questions and objectives
6. Determining relevance, contributions, and expected outcomes
7. Producing a research plan

The review of previous studies was used to gain a deeper understanding of the study. This involves analyzing the past and current situation in the area of study, repeatedly.

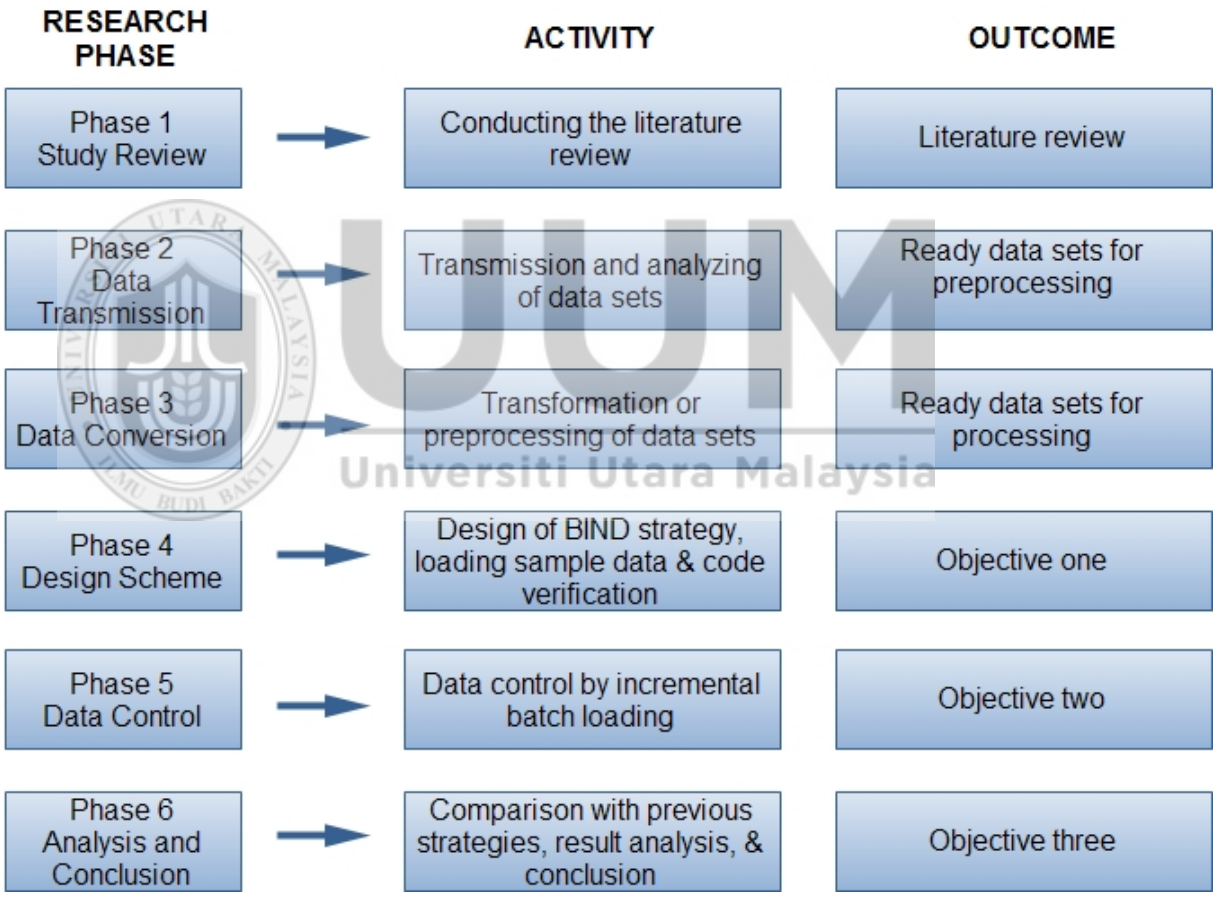

Figure 3.1: Research Framework

3.2 PHASE TWO: BIG DATA TRANSMISSION

Phase two is the beginning of the design stage. For this study, live data acquired from PingER, was used. Results from Internet monitoring, in the form of log data,

was acquired from PingER in their original form, for the purpose of this study. The data sets were extracted from PingER archives to Local File Sytem (LFS), where data analysis was performed. The analysis is important to determine the type of query that can be supported by the data, hence, choosing a suitable indexing strategy. The data sets are then loaded into HDFS as illustrated in Figure 3.2. Phase 2 is a very important phase because it gets the whole data into Hadoop, ready for conversion, processing, and storage.

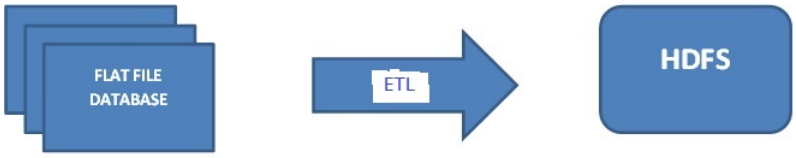

Figure 3.2: Extract, Transform, and Loading of Data

3.3 PHASE THREE: BIG DATA CONVERSION

This phase explains how Big Data will be preprocessed or transformed before being loaded in Hadoop. Firstly, data will be extracted from PingER archives to LFS, where preprocessing will take place. This is performed to enable the data to be supported by Hadoop. After preprocessing, the transformed data will be loaded into HDFS. Conversion of the data sets will only take place after a complete transfer of the data into LFS. Text file will be taken as input; that is, TextInputFormat will be used to feed one record per line from HDFS for processing. Each line will have an identification number (id) to avoid duplicates. This phase gets the data ready for processing and storage.

3.4 PHASE FOUR: DESIGN SCHEME (PARAMETER SETTING / CONDITIONS)

This phase consists of three steps as shown in Figure 3.3. First, the design of the proposed strategy. Next, loading of Big Data samples, and finally, the verification of

the code.

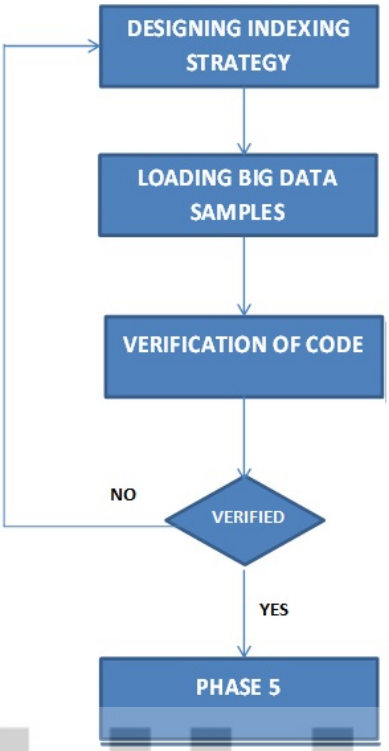

Figure 3.3: Design of the Proposed Strategy

3.4.1 Step One: Examining The Proposed Indexing Strategy

The existing indexing strategies were examined to enhance the design of the proposed strategy. In general, most of the existing strategies work fine with data sets of small (or fairly large) amounts, but incure computational overhead when implemented on very large amounts of data [49]. This arises from series of MapReduce activities since most of the studies concentrate more on MapReduce phase. Researches [29, 30, 25, 28, 9, 5, 26] have proposed solutions to solve the problem. The current indexing strategy or approach based on the study by Gollub *et al* [30] presented a solution by indexing library documents to mitigate the problem. Their strategy also concentrated on the mapper and reducer phase only as explained in Section 2.6. To reduce the number of MapReduce activities, Ian Foster's Task-Scheduling concept will be applied to the chosen Indexing technique. A suitable Indexing technique was chosen based on the type of query to be performed on the data sets. In this study, keyword queries will be

considered, hence, the Inverted indexing technique will be deployed. The following were examined:

- The current Strategy on the MapReduce framework and
- How the Proposed Strategy Works

3.4.1.1 *Examining the Current Strategy on the MapReduce Framework*

Referring to Figure 3.4, the current design of the Inverted Indexing Strategy on the MapReduce framework, is implemented by taking a given data set and passing it through the following phases:

1. Inputting files: Files are submitted to HDFS, which serves as a mechanism for receiving inputs for MapReduce applications.
2. Inputformat: Hadoop MapReduce supports numerous file formats; SequenceFileInputFormat, NLineInputFormat,FixedLengthInputFormat, TextInputFormat, etc. The InputFormat is used to define how data will be read into the mapper instances. A user can define an InputFormat implementation to format the data or input to be programmed based on personal or organizational requirements.In Inverted Indexing, lines of text files are read by the default TextInputFormat. The byte offset per line read, is the key produced for each record. The content of the line terminated by '\n' character, is the value. The InputFormat also divides the input file or data into fragments (called splits) that will be fed into individual mappers as tasks. The InputSplit is performed before the start of the MapReduce job. Hence in a nutshell, InputFormat performs two tasks:

    a) diving the data sets into fragments/blocks/splits and allocating to mappers as jobs.

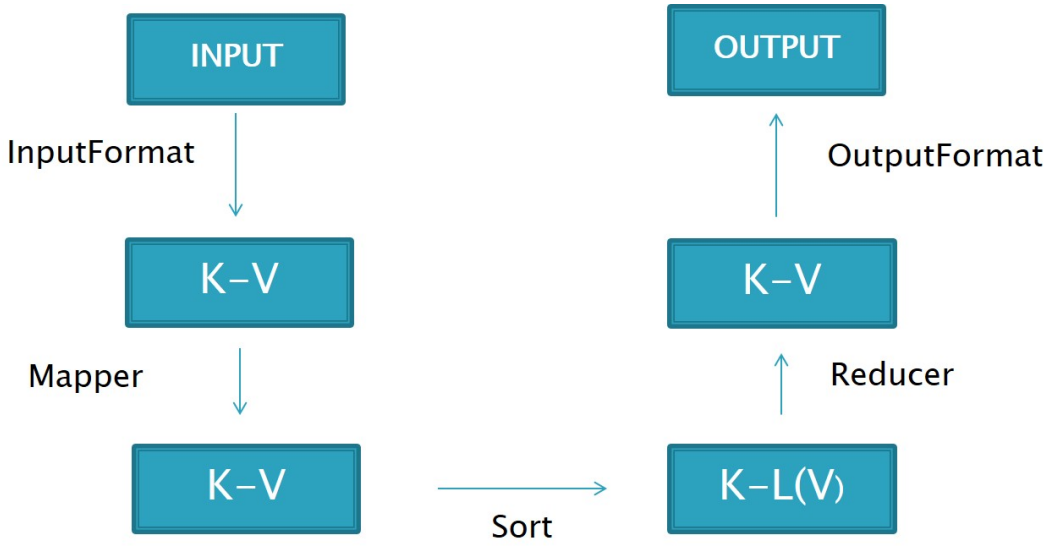

Figure 3.4: MapReduce Phases

b) sub-dividing the splits (jobs) into records, which are fed one after the other to the mapper as tasks. This is achieved through the RecordReader class, as illustrated in Figure 3.5.

3. Mapper: Mapper takes one line at a time, and produce key-value pairs by splitting the lines and sorting the texts. In Inverted Indexing Strategy, the key is a unique representation of each text that occurs in the data sets, and the value is the files in which the keys occur.
4. Reducer: Reducer acquire the keys and list of values from each mapper, combining the keys that conform to each other and their corresponding values. The list of values are converted into comma delimited strings, ad emitted as output.

3.4.1.2 *How the Proposed Strategy Works*

The proposed design is based on Ian Foster's task-scheduling concept [44]. Ian Foster's task-scheduling concept explains that in parallel processing, rather than wait for the whole processes to finish executing task before returning to the master, processes who have completed task will return to the master for further allocation of task. Consequently, based on Ian Foster's task-scheduling concept, the research employed the

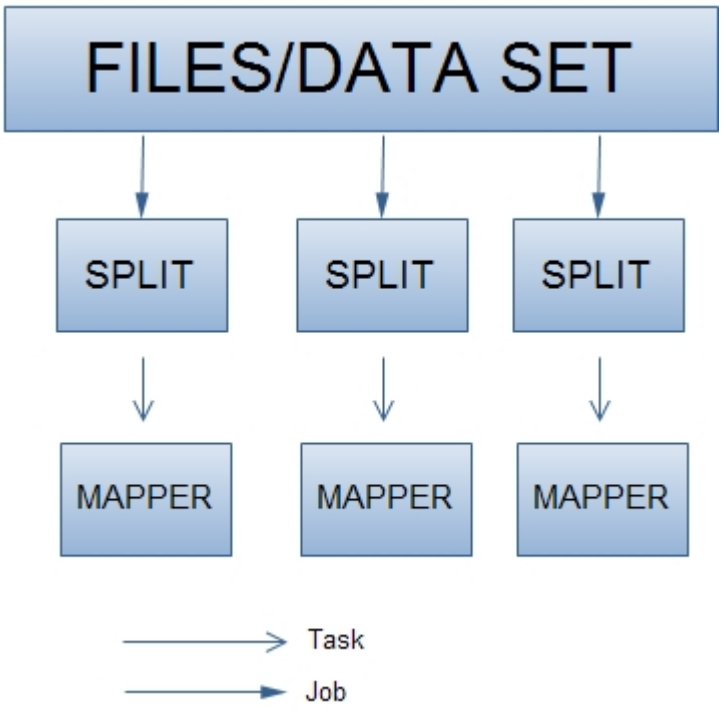

Figure 3.5: Task Execution in RecordReader

traditional way of feeding data into the mapper (InputFormat and RecordReader) to implement the proposed Indexing Strategy. The study concentrated on customizing the RecordReader, thereby allowing the proposed Indexing Strategy to be implemented on records that have been fed into the mapper as tasks. The mapper receives three lines or records from the RecordReader as task, instead of one line at a time as accustomed to the traditional or default TextInputFormat. For each mapper with a completed task, is assigned another task (sequentially) without waiting for the other mappers as illustrated in Figure 3.6. The TaskTrackers (a component of the mapper), are responsible for conveying of tasks between the RecordReader and the mapper. It is expected of the proposed design to improve processing of Big Data by reducing the processing time of data sets, thereby mitigating computational overhead.

The operational description of the entire indexing strategy is presented as a flow chart in Figure 3.7, where x = number of assigned task. Logical and mathematical comparison are tested against the data as they arrive. When the condition is satisfied, data

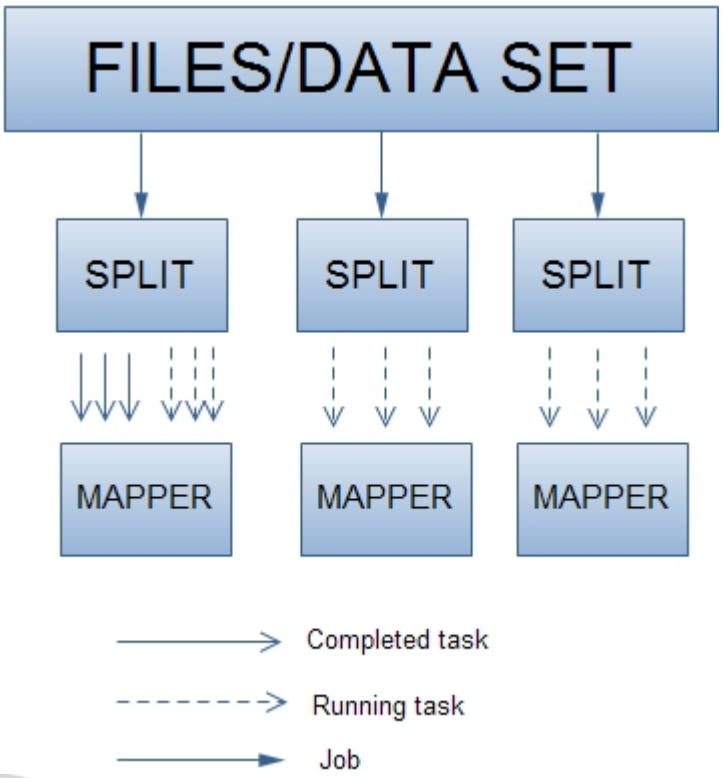

Figure 3.6: RecordReader for the Proposed Strategy

are then indexed and stored. Also, the algorithm for the proposed indexing strategy is presented in Algorithm 3.1. The algorithm will be implemented in the InputFormat, RecordReader, mapper, and reducer classes. The number of input splits is dependent on the amount of resources used (CPU cores and RAM). Efficient management of splits before the get to the mapper phase, could mitigate computational overhead unlike what is obtainable in the current indexing strategy.

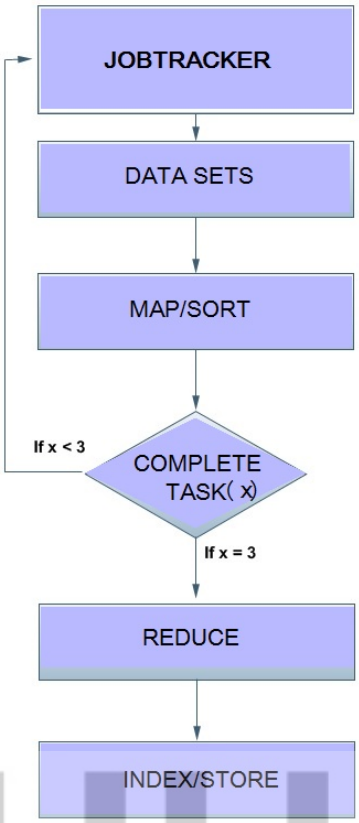

Figure 3.7: Task Execution with the proposed Strategy

Algorithm 3.1 Algorithm for the Proposed Strategy

if {
  $x \neq 0$
  split x ;
}
where x = InputFiles
let
  y = split
  m = mapper
  n = lines to process
  t = task
  t = 0,1,2,_ _ _,n
  y = 0,1,2,_ _ _,n
where $\forall m \exists y$
if {
  $y \neq 0$
  assign t;
}
$\forall m$
if {
  t = 0
  t < n
  t ++ ;
}
let
  k = key or item to be indexed
  v= value associated with key
  l(v) = list of values
  i = 1,2,3,_ _ _,n
  j = 1,2,3,_ _ _,n
  where j > i
do {
  map (k_i, v_j)
  sort $(k_i, l(v_j))$
  combine $(k_i, l(v_j))$
  reduce (k_i, v_j)
  index k_i
  store;
}

3.4.2 Step Two: Loading Big Data Samples

The sample data set, which is a small amount of the data, will be used to test for configuration and program errors. The data sets will be assigned by the JobTracker

(ResourceManager) for processing. Both the current and the proposed design will be implemented on the data sets. The JobTracker gets the data ready for processing by splitting the data (as assignment) to various TT for MapReduce, and also by assigning key-value to be used in index extraction.

3.4.3 Step Three: Code Verification

The sample data set will serve as the test batch. Verification of codes will be performed using this data set. If the process return error(s), then step one will be performed again. Otherwise, we move on to phase five (refer to Figure 3.3).

3.5 PHASE FIVE: DATA CONTROL (BATCH PROCESSING)

After the code has been tested and verified using the sample data set, the strategies (both the current and the proposed) will be implemented on the remaining data sets. The data sets will be grouped into four sets with varying sizes namely D1, D2, D3, and D4, and will be feed into HDFS one batch at a time.

3.6 PHASE SIX: EVALUATION AND CONCLUSION

This phase marks the end of the design stage. Generally, the performance of indexing strategies used in processing Big data are evaluated with the processing or execution time and the query time [24, 54, 76, 10, 27, 26, 25, 5]. The study shall consider these metrics to be used in evaluating the proposed strategy. The metrics are explained as follows:

• Processing time: This is the interval between assigning a dataset to a node for MapReduce and indexing, and the time when the data is finally stored. This means the time taken to create an index. The processing time could otherwise be known as the execution time or the CPU time. The proposed strategy is expected to emerge with

a minimal processing time after evaluation.

• Query time: This is the period or interval between querying the datasets and information retrieval. A decrease in query time or access time is expected from the proposed design.

The performance of the proposed strategy will be evaluated using:

- the processing time for varying sizes of data sets, to note if the performance improves or declines as the size of the data increases
- the processing time for varying number of computational nodes, to note the performance when the number of computation resources are increased.
- and the query time for varying size of data sets, to check if it facilitates information retrieval and if it maintains a consistent performance despite the growth of data

The Hadoop process logs explains MapReduce-specific information which includes the job execution time, processing time, or CPU time. This can be viewed in the ResourceManager web User Interface (UI), which will be available in the 50070 and 8088 port of JobTracker. Also, MapReduce job activities are displayed on the terminal at the end of every job execution. Microsoft excel software will be used to compute, view, and evaluate the metrics.

3.7 SUMMARY

This chapter explains the methodology used in achieving the objectives of the study. It starts by introducing the proposed Big Data indexing strategy called BIND strategy

which facilitates processing of large amounts of data to ease storage and information retrieval. It then explains the research phases which includes study review, data transmission, data conversion, data scheme, data control, analysis, and conclusion. The chapter also describes the algorithm that will be used in the implementation of the proposed strategy, as well as the metrics that will be used in evaluating the proposed BIND strategy. The next chapter (Chapter Four) explains the implementation of BIND strategy and the tools utilized in the study.

CHAPTER FOUR
THE IMPLEMENTATION OF BIG DATA INDEXING STRATEGY (BIND)

This chapter explains the implementation of the proposed indexing strategy. Specifically, Section 4.1 outlines the tools used in carrying out the study. It also lists the prerequisites and dependencies of Hadoop installation. Section 4.2 describes the step by step approach used in achieving the objectives of the study, and Section 4.3 outlines the metrics for the evaluation of the strategies. It also explains how the performance of the proposed strategy will be evaluated.

4.1 EXPERIMENTAL SET UP

This Section outlines the tools used in carrying out the study. It also lists the prerequisites and dependencies of Hadoop installation. The experiment was carried out on Ubuntu 14.04 operating system. The tools used include:

1. Hadoop: Hadoop is a software framework for processing and storing of Big Data on clusters of hardware. It is an open-source software that utilizes the MapReduce framework for processing, and Hadoop Distributed File System (HDFS) for storage. Hadoop version 2.6.0 was deployed in the study for processing and storage of data. Its installation includes downloading and extraction of the tar file, downloading and extraction of the dependecies, installation of SSH (Secure Shell) protocol, configuration of Hadoop files (by editing and customizing the files to suit user requirements), and formatting of Hadoop namenodes.The implementations of Hadoop are confined to some dependencies listed as follows:

    a) Maven: Apache Maven manages a pojects build and documentation. Apache Maven 3.3.3 was used in the study.

b) Cmake: Cmake manages applications and directory hierarchies which is dependent on multiple libraries. The study used Cmake version 2.8.12.2

c) Java: Java softwares are used for running applications written in the Java programming language. Java version 1.7.0_79 was used.

d) Protobuf: Protocol Buffer (Protobuf) is a basis for Remote Procedure Call (RPC) system used for inter-machine communication. Hadoop depends on this, and specifically for this study, Libprotoc 2.5.0 was used.

e) Snappy: Snappy is used in Hadoop to test decompression, to detect error in the compressed stream. Snappy version 1.1.2 was utilized.

Hadoop can be used in one of three forms as follows:

i. Single cluster/standalone/local mode: Everything runs on a single machine. This is suitable for running MapReduce test programs during development, because of the ease to test and debug errors.

ii. Pseudo-distributed mode: The daemons run on the local machine. This is used to simulate a cluster running on a small scale.

iii. Fully distributed mode: This depicts a large scale environment consisting of clusters of machines.

In this case, a pseudo-distributed multi-node Hadoop cluster was built consisting of three computers or machines. Hadoop was installed, configured, and tested on each of the machines - forming three single node clusters. The three single node clusters were merged into one multi-node cluster. One machine served as the master as well as a slave, and the other two served as slaves, as illustrated in Figure 4.1. The master node was in charge of the JobTracker (ResourceManager), the NameNode, and the Secondary-NameNode. The slaves were in charge of the DataNode and the NodeManager or TaskTracker as presented on Figure 4.3. The master in this case consists of all daemons because it is a combination of a master and a slave,

as illustrated in Figure 4.2. This was done to maximize resources. The master daemons were responsible for management and coordination of all other daemons, while the slave daemons operate on data processing and storage. Figure 4.4 displays the DataNode information before the execution of jobs. As illustrated in Figure 4.4, all three nodes were configured using Hadoop version 2.6.0. Each of the machines were able to reach and access the others when connected to the network. This was possible as a result of SSH which was installed prior to Hadoop installation. Machine 1 or the master's public SSH key was added to the list of keys authorized by the slaves, to grant the master access and control of all nodes.

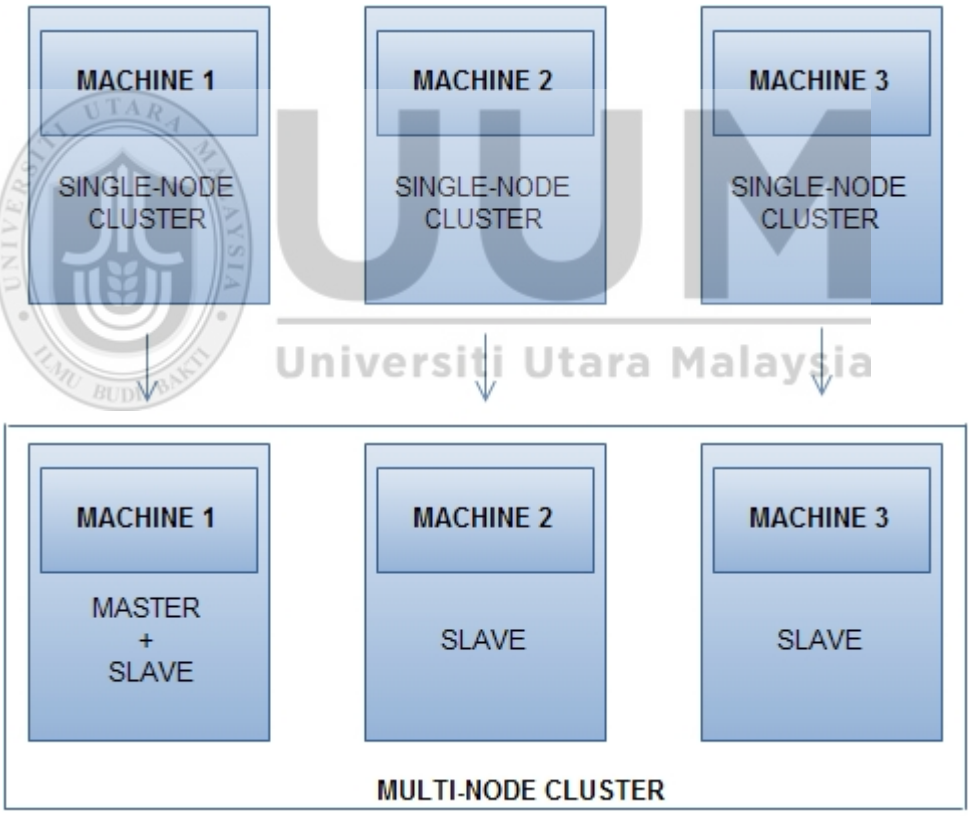

Figure 4.1: Configuring Hadoop Cluster

2. Eclipse IDE: As an Integrated Development Environment (IDE), Eclipse contains a workspace for writing computer program codes. In this study, Eclipse Kepler IDE was used in writing the codes for the processing of Big Data. It was

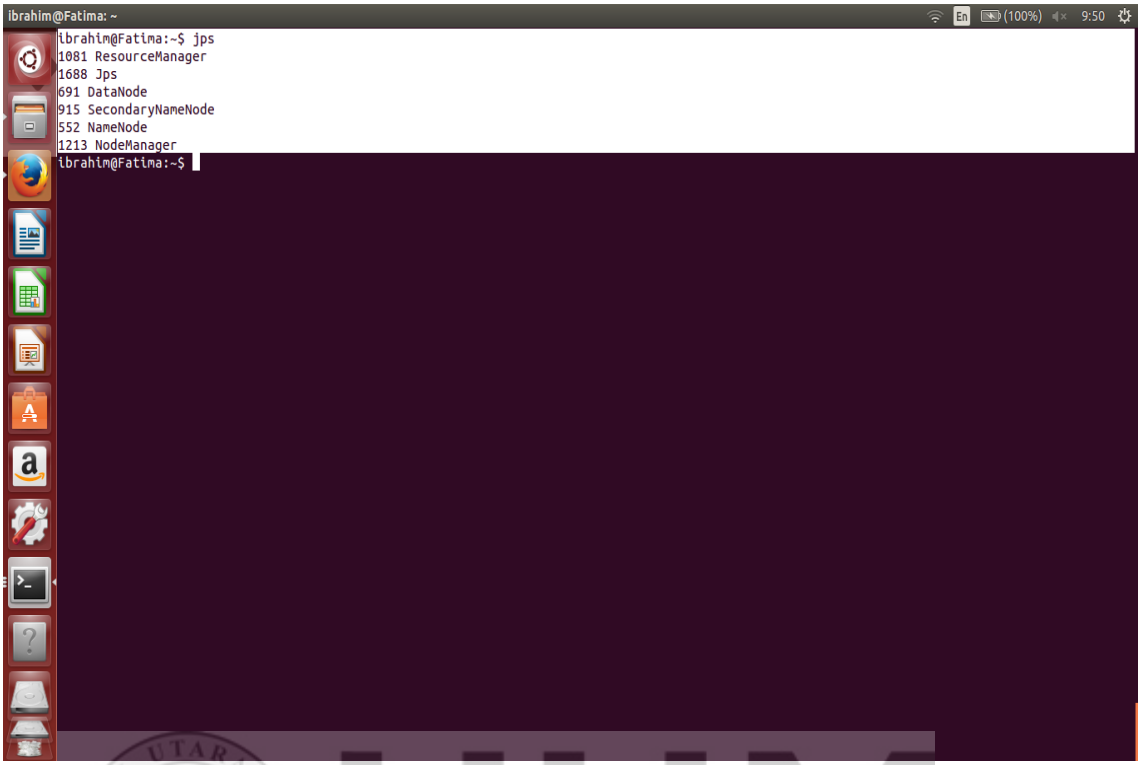

Figure 4.2: The Master Node Deamons

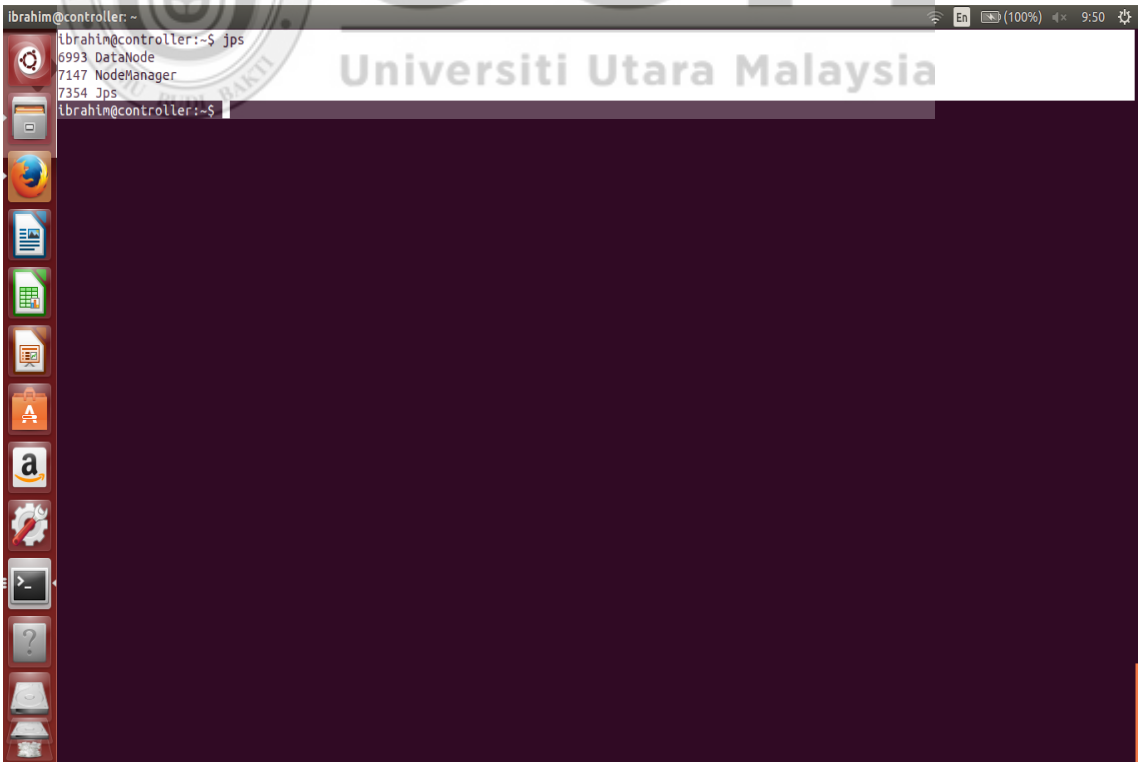

Figure 4.3: The Slave Node Deamons

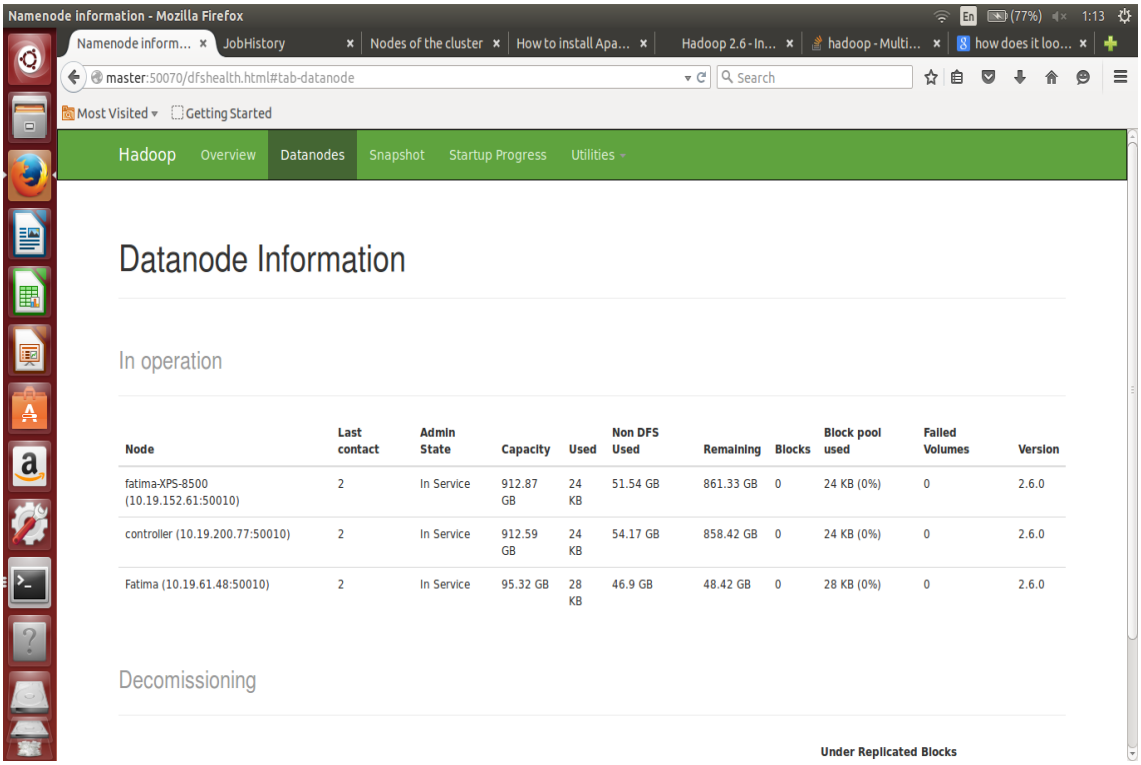

Figure 4.4: DataNode Information

also used in exporting the JAR (Java Archive) executable files used by Hadoop to run the codes.

3. Hive: Apache Hive as a data warehouse infrastructure, is used for data summarization and query. It is built or installed on top of Apache Hadoop. It uses an SQL-like language called Hive-QL (or HQL for short), for data query. Hive 1.2.0 was utilized for data query in the study.

4.2 EXPERIMENTAL PROCEDURE

This Section describes the step by step approach used in achieving the objectives of the study. Sub-Section 4.2.1 explains objective one and how it was achieved, while sub-Section 4.2.2 describes how the second objective was achieved in the study.

4.2.1 Design and Coding of BIND Strategy

The study presents the design of an Indexing Strategy (IS) that processes Big Data using the MapReduce framework, in order to facilitate storage and Information Retrieval (IR) as the first objective. The concept behind the design is as explained in Section 3.4.1. Section 3.4.1 explains the theory behind the current inverted indexing strategy, and how Ian Foster's concept fits into that theory. This Section (4.2.1) explains the design of:

- the current Inverted Indexing Strategy on the MapReduce framework
- the proposed Inverted Indexing Strategy on the MapReduce framework

4.2.1.1 *Inverted Index and MapReduce Framework*

Big Data can be processed in numerous ways, depending on the type of queries to be performed on it. Examining the PingER historical log data (refer to Figure 4.5), the fields are delimited by spaces with the first two fields representing the source address and the destination address respectively. The values (or dots in some cases) represents the Time-To-Leave (TTL), and the last two fields also represents the source and destination address.

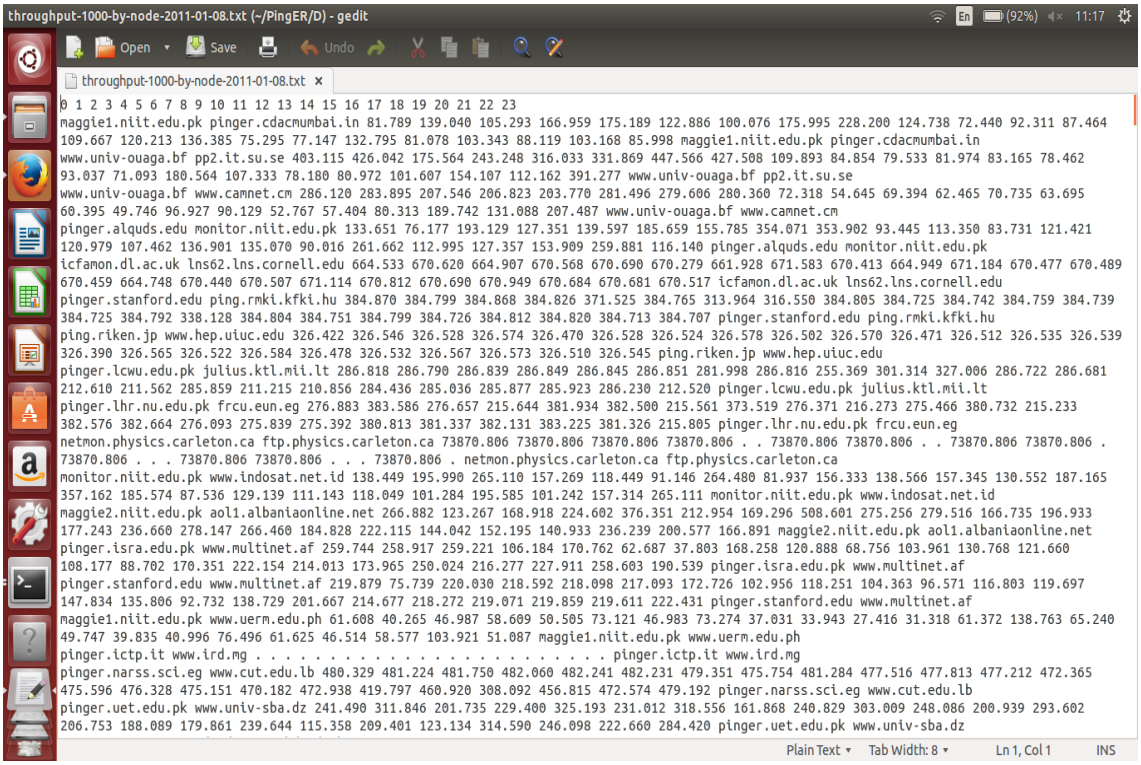

Figure 4.5: PingER Historical Log Data

The data sets were extracted from PingER archives to the local file system where data analysis was performed. The analysis was done by examining the properties of the data to determine the type of query that can be supported by it, hence, choosing a suitable indexing strategy. In this research, a presumption was taken that the information to be retrieved from the data sets, is the number of times a particular address was pinged in a given period of time. This could be interpreted as a keyword query. Therefore, the Inverted Indexing technique was deployed (since it supports keyword queries). In the design of the current inverted indexing strategy, three (3) Java classes were created in eclipse kepler IDE: the mapper class, the reducer class, and the main or driver class. Each class does the following:

1. The driver class: The main or driver class drives the implementation of the mapper and the reducer class as illustrated in Figure 4.6. Also, the default libraries of

the InputFormat (and its RecordReader) class were imported by adding a couple of code lines into the Hadoop import libraries of the driver class, as in Figure 4.7. These classes were implemented as used in MapReduce applications.The default implementation of the InputFormat (which is the TextInputFormat) in the current inverted indexing strategy, is by dividing the files (specifically, text files) into splits and feeding each split to mappers. The InputFormat's RecordReader further divides each split into records which was fed one record at a time to the mappers as tasks.

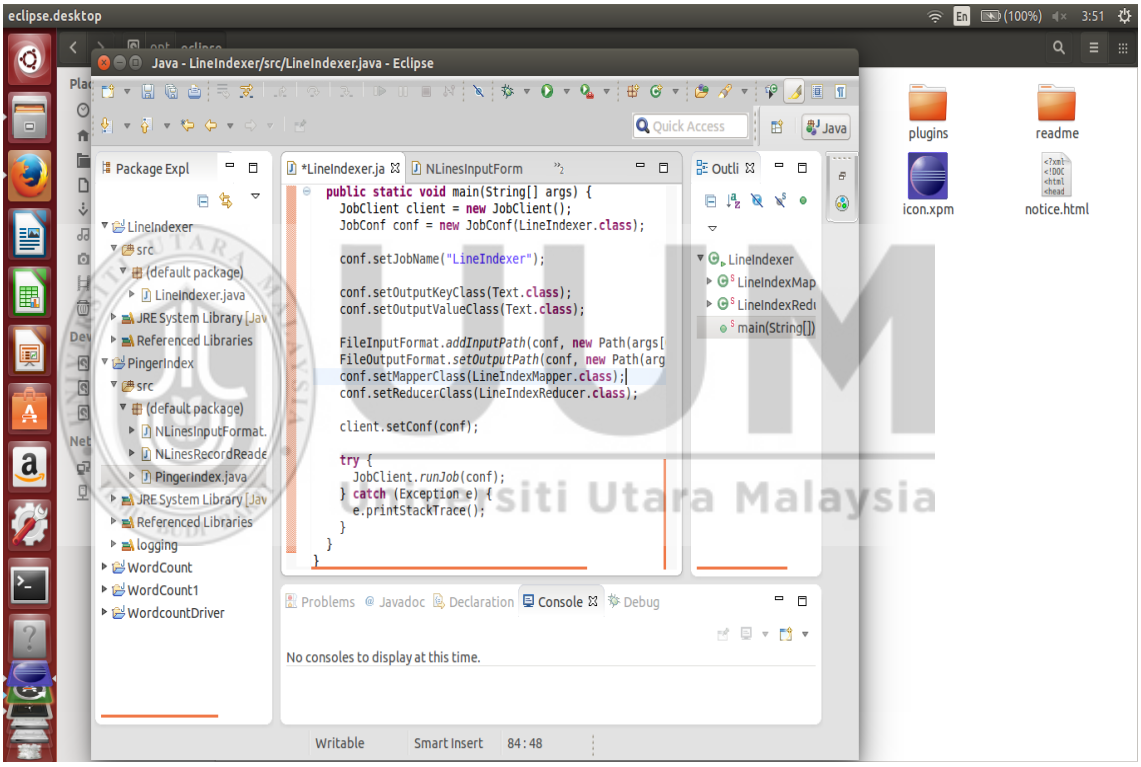

Figure 4.6: The Driver Class for the Current Strategy

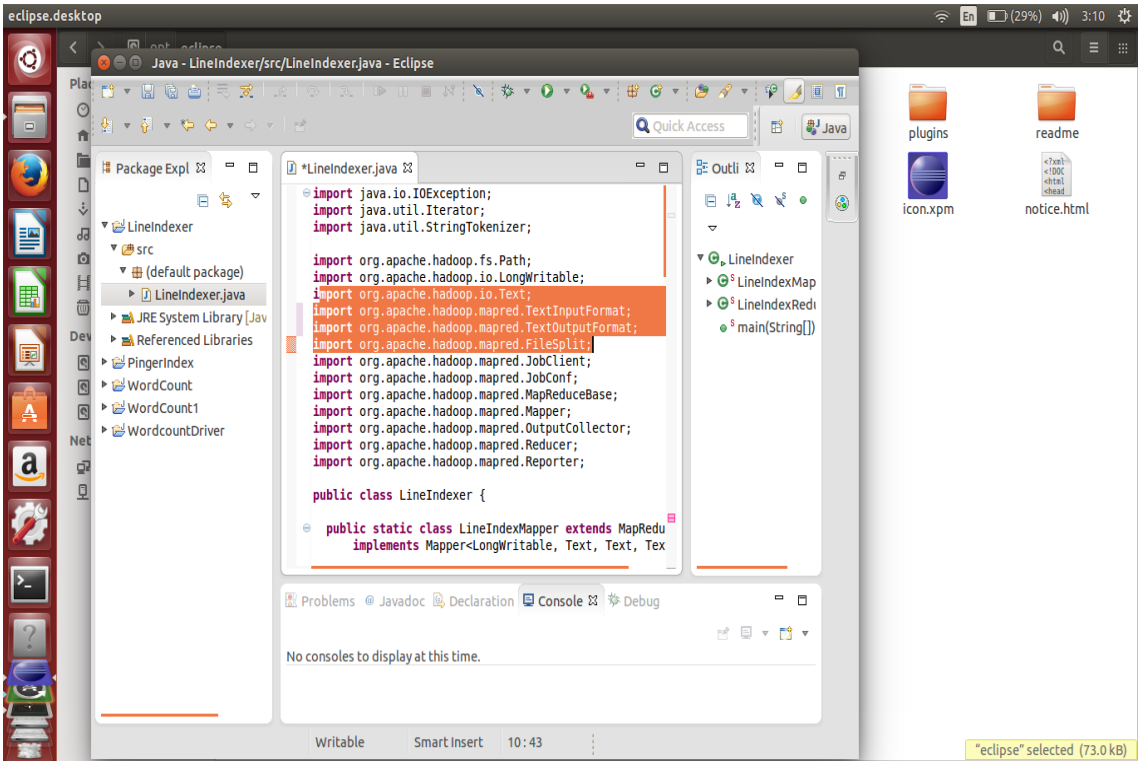

Figure 4.7: Default InputFormat Library

2. The mapper class: Mapper classes are written to sort keyvalue pairs according to the information (indexes) the user desires to acquire from the data sets. Take for instance, the regular search engines. In search engines, keyvalue pairs are sorted with each occuring word in documents as keys, and the documents in which such words occur, as values. Another example is as seen in the study by Gollub *et al* [30], in which a set of user queries is taken as key, against a set of library documents. In this light, the mapper class for the current inverted indexing strategy sorts the records to provide keyvalue pairs, where the keys are the unique occurence of each word, and the values are the list of locations (files/documents) in which the words occur as illustrated in Figure 4.8.

   In the study, the mapper class for the current strategy was written to sort each word in the files as key, and return the number of times such word occurs in the files as value (refer to Figure 4.15). The mapper class for the current strategy performs the same task as that of the proposed strategy.

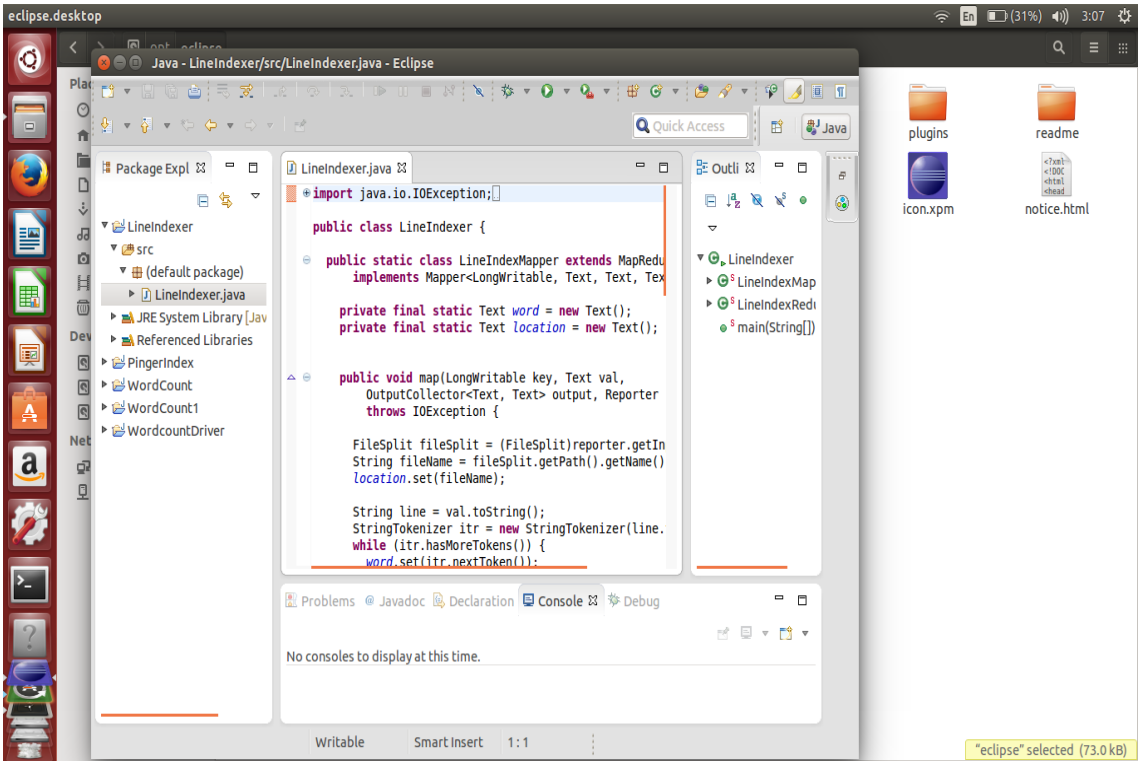

Figure 4.8: Mapper Class for Inverted Indexing Strategy

3. The reducer class: A reducer class was written to sum up the keys and list of values from the mappers, and then output them as keyvalue pairs (refer to Figure 4.16). The reducer class for the current strategy performs the same task as that of the proposed strategy.

4.2.1.2 *The Proposed Strategy*

In the design of the proposed inverted indexing strategy, five (5) Java classes were created: the mapper class, the reducer class, the main/driver class, the InputFormat class, and the RecordReader class. The design concentrated more on the InputFormat's RecordReader, as this is where the algorithm was implemented. Each class does the following:

1. The InputFormat class: The InpuFormat class was created by extending the default TextInputFormat class (refer to Figure 4.9) using the code line:

public class NLinesInputFormat extends TextInputFormat{ _ _ _ }

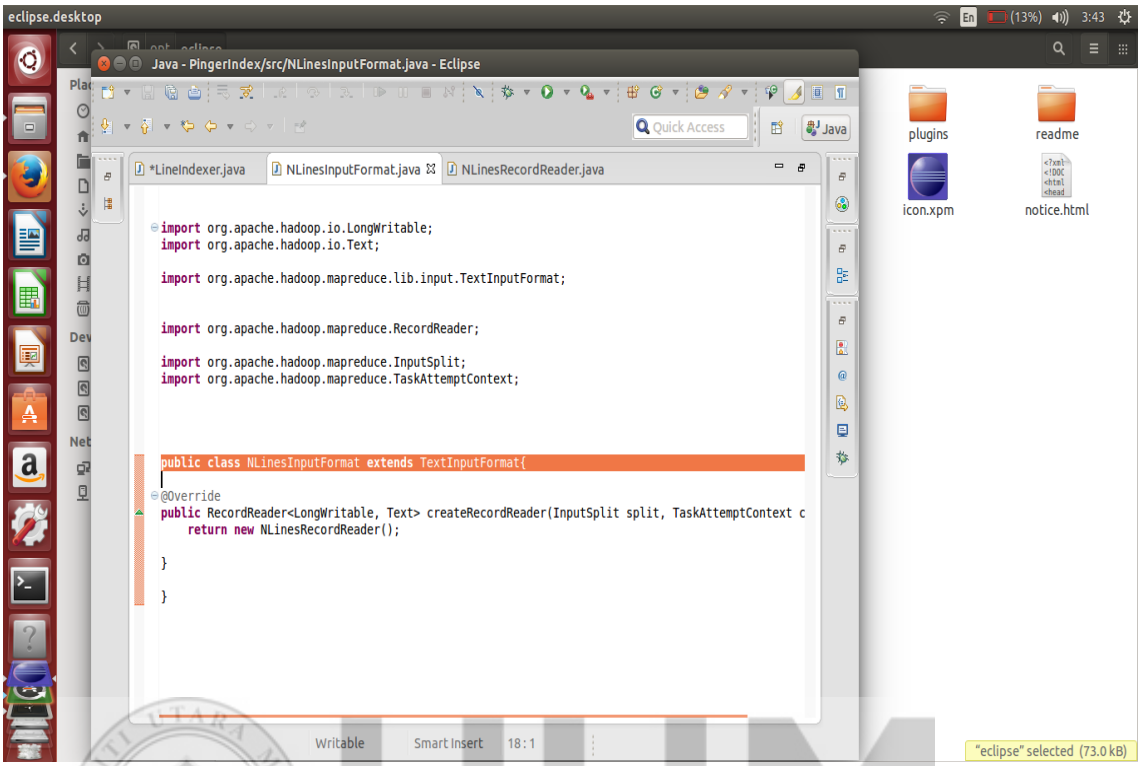

Figure 4.9: NLinesInputFormat Class for the Proposed Strategy

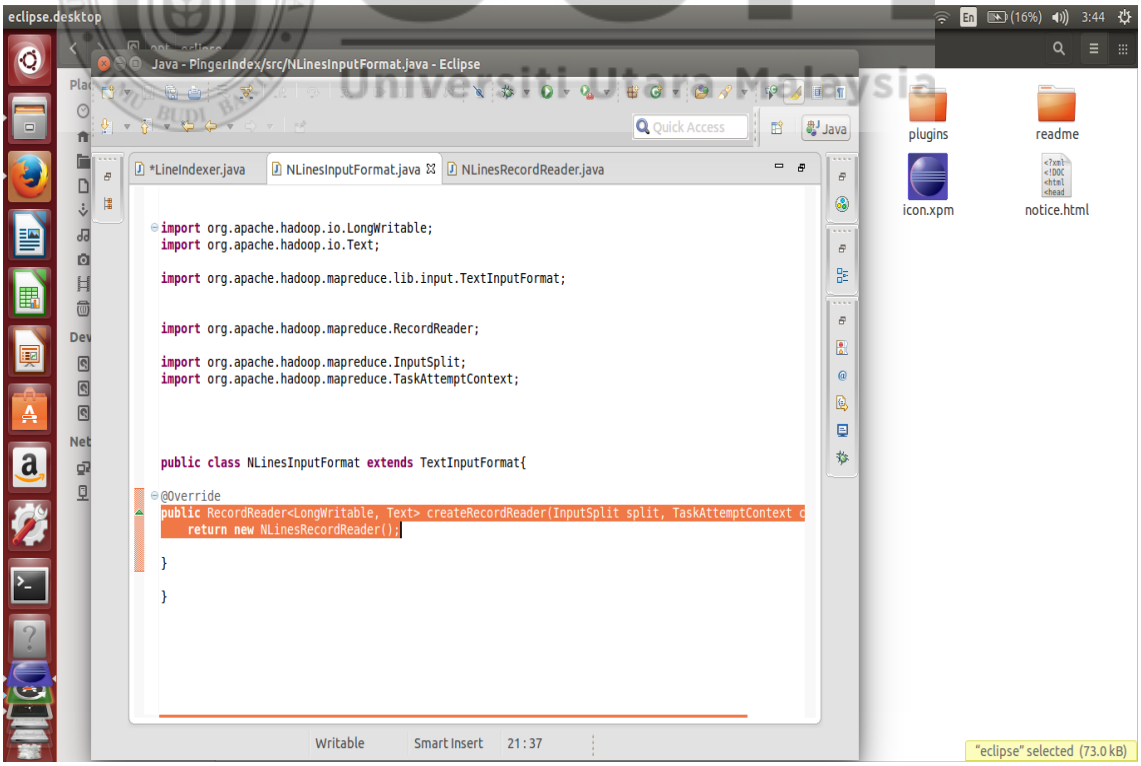

Figure 4.10: Accept n-Number of Lines in RecordReader

The NLinesInputFormat allow a given number of lines to be fed into the mapper by the RecordReader, unlike the default TextInputFormat used in the current strategy which treats offset as keys, and contents of lines as value thereby permitting one (1) line to be read at a time. From the InputFormat class, permission was given to the RecordReader to assign n number of tasks to each individual mapper (where n is the number of task defined by the user), using the constructor (refer to Figure 4.10):

public RecordReader _ _ _{

return new NLinesRecordReader ( ) }

The constructor takes InputSplit as parameter, to enable the action to be taken on the splits. It also overrides the default RecordReader class to use the proposed RecordReader class.

1. The RecordReader class: The default RecordReader class was extended to implement the proposed strategy. For the implementation of the proposed strategy, the variable:
   private final int NLINESTOPROCESS = 3;
   was assigned to allow the RecordReader fed three (3) records to individual mappers as task instead of the default 1 (refer to Figure 4.11).

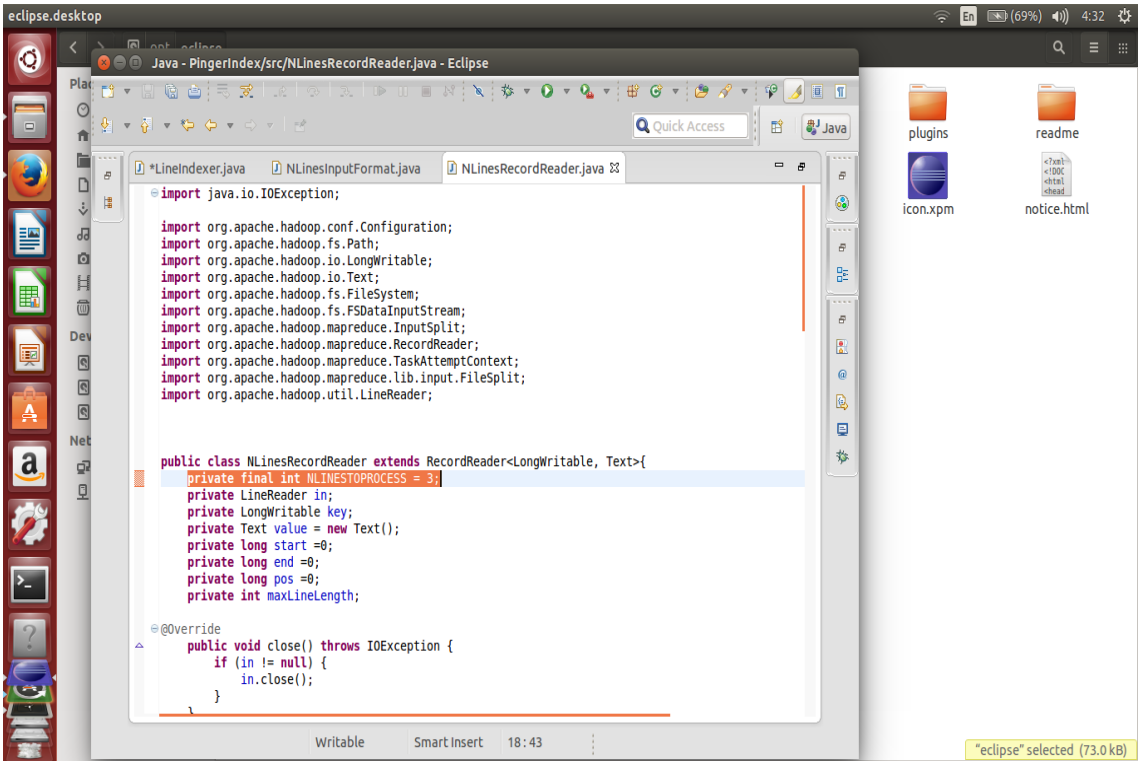

Figure 4.11: NLinesToProcess in RecordReader

Six (6) methods were inherited and implemented from the default RecordReader class. They are:

- initialize ( )
- getCurrentKey ( )
- getCurrentValue ( )
- nextKeyValue ( )
- getProgress ( )
- close ( )

The methods initialize ( ) and nextKeyValue ( ) were overriden to apply the proposed algorithm. The method initialize ( ) was customized to get the splits, and assign them to each mapper, allowing mappers that have 'ended' or completed task, to be assigned new task (or to 'start' new task) as illustrated in Figure 4.12.

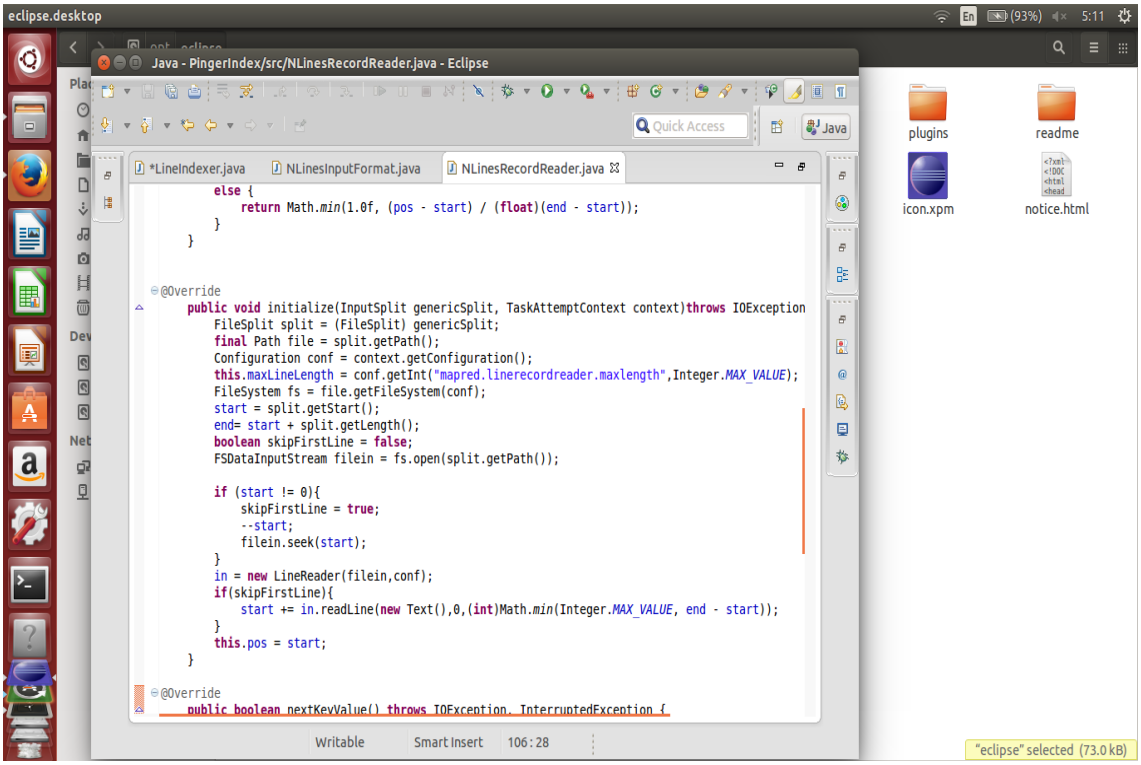

Figure 4.12: The Method Initialize ( )

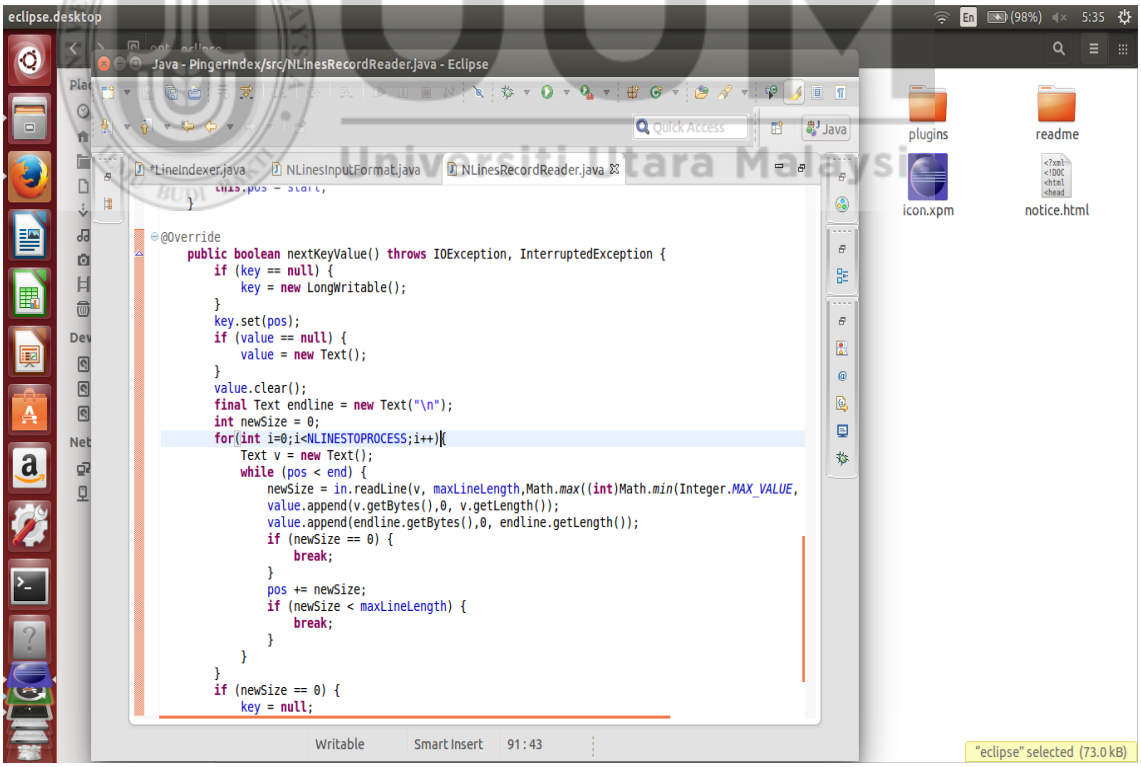

Figure 4.13: The Method nextKeyValue ( )

The method nextKeyValue ( ), uses sub-algorithm 4.1 to assign line per task to individual mappers. The algorithm was interpreted into codes and implemented as illustrated in Figure 4.13.

Algorithm 4.1 nextKeyValue

for {
  i = 0
  i < n
  i ++;
}
where i = number of task being processed by the mapper
  n = number of task assigned to each mapper to be processed

1. The main (driver) class: The main class was written with the InputFormat type set to the InputFormat class for the proposed strategy as illustrated in Figure 4.14.

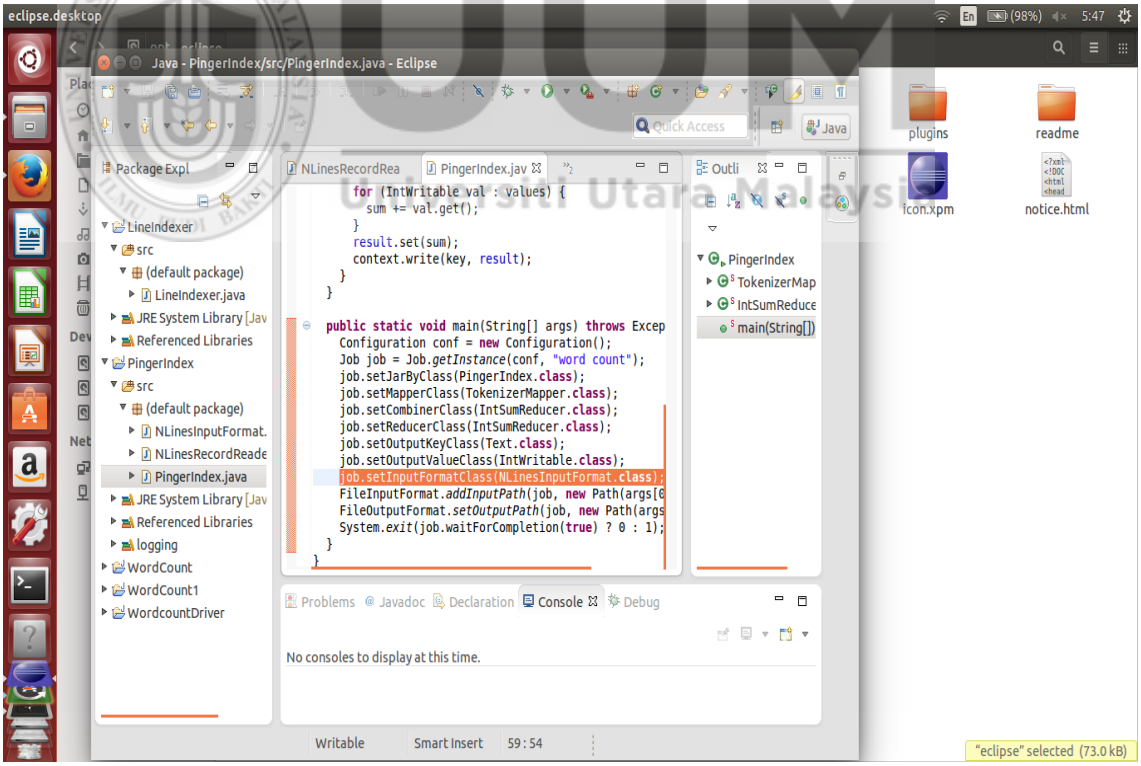

Figure 4.14: The Main Class for the Proposed Strategy

2. The mapper class: The mapper class for the proposed design was written to set each word in the files as key, and return the number of times such key occurs in

the files as value, illustrated in Figure 4.15. The mapper class for the proposed strategy performs the same task as that of the current strategy.

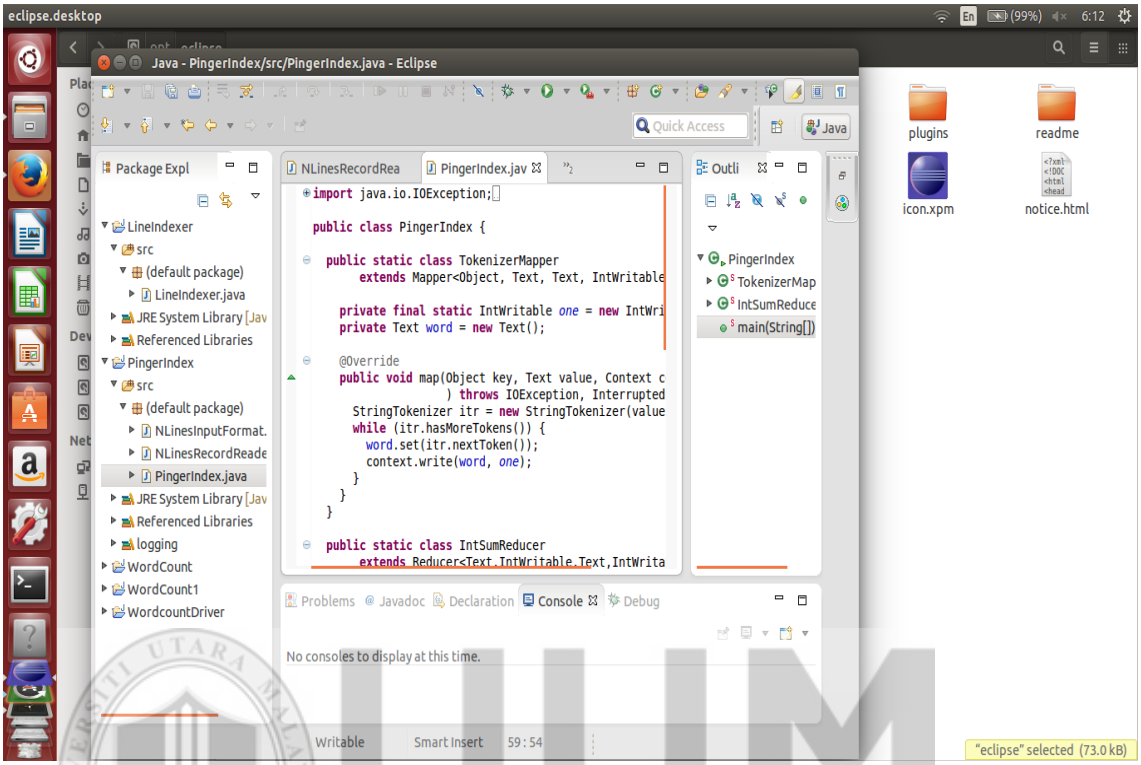

Figure 4.15: The Mapper Class

3. The reducer class: The reducer class was written to sum up or combine the keys and list of values from the mappers, and then output them as keyvalue pairs, illustrated in Figure 4.16. The reducer class for the proposed strategy performs the same task as that of the current strategy.

4.2.2 Implementation of the Proposed Strategy

The output of the second objective is the implementation of the proposed strategy using PingER log data. This was achieved through the following steps:

4.2.2.1 Step One: Extraction of Unstructured Big Data

This is the process of gathering Big Data from its original source, in its complex form. This study is supported by PingER historical log data, which is gathered from PingER

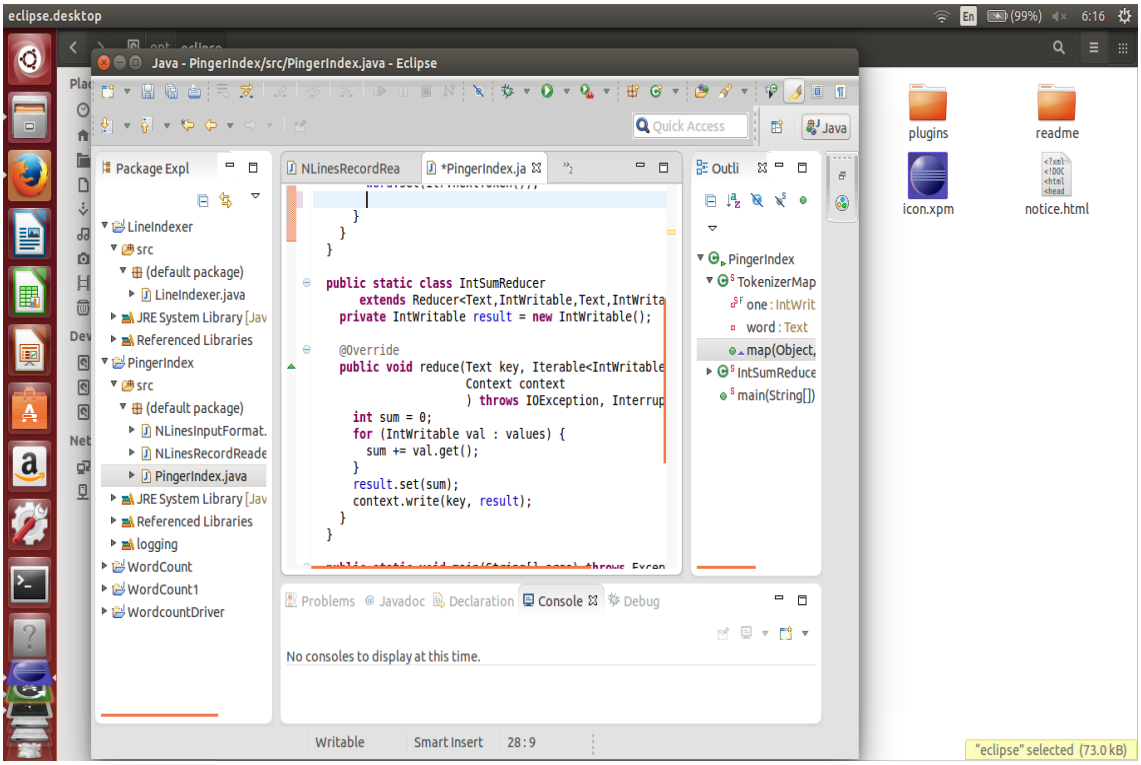

Figure 4.16: The Reducer Class

MAs (or archives) for making case studies of Internet performance within Malaysia, S. E. Asia, and the rest of the world. Figure 4.5 depicts PingER data in its raw or unprocessed state.

4.2.2.2 Step Two: Transformation or Preprocessing of Big Data

The initial state of the data used in the study was in a compressed form stored in flat file formats. Also the size of the files were in bytes and kilobytes (too small to be supported by Hadoop). Hence, needed to be uncompressed and merged (preprocessed) in order to be supported by Hadoop. The preprocessing of the files (data set) was performed using a shell script. The data sets were grouped into four sets with varying sizes namely D1, D2, D3, D4, and populated with data sizes of 50.1MB, 100.3MB, 150.1MB, and 200.4MB respectively. Another data set D, was populated with 20MB and was used as the sample data to test for configuration and program errors. Though,

50MB - 200MB of data is far from what the size of Big Data is considered to be, it is large enough for potential improvements to be noticed using indexing. Also, the amount of computational resources used in the study is limited, and thus, does not permit for experimenting on very large data sets.

4.2.2.3 Step Three: Verification of Codes

A sample data set, D, with size 20MB was used in verifying the codes for the detection of configuration and program errors. The current strategy as well as the proposed strategy, were implemented on it. All configuration issues were dealt with and program errors debbuged before the actual experiment was performed. This is to ensure that the tools have been configured correctly, and the codes are working fine. Figure 4.20 illustrates a MapReduce job in progress, with the proposed strategy being tested on the data set D to ensure the codes are free of error.

4.2.2.4 Step Four: Loading of Big Data into Hadoop Distributed File System (HDFS)

An important step prior to processing of the data sets is loading it into HDFS, which is the primary storage used by Hadoop. Also, data is passed on to MapReduce for processing from HDFS. A couple of commands were written to move the data sets from local file system to HDFS. Data in HDFS and all other Hadoop processes can be viewed through Hadoop web User Interface (port 50070 and port 8088), as illustrated in Figure 4.17. Figure 4.17 shows the data sets D, D1, D2, D3, and D4 in HDFS. It also displays the properties of each data set such as permission type, user name, user group, file size, replication number, and block size. The properties - file size, replication number, and block size, reads zero. Each data set or file will have to be opened to view the actual values of these properties (size, replication, and block size) as illustrated in Figure 4.18 which presents the properties of the sample data set D

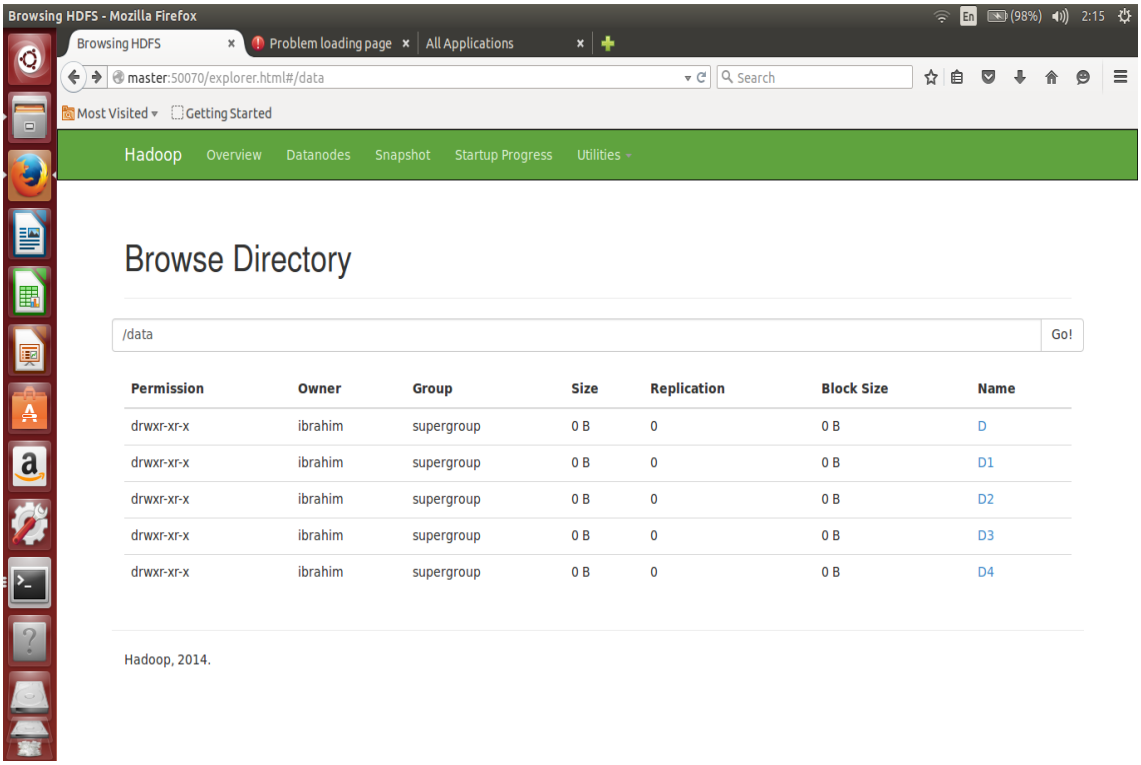

Figure 4.17: Unprocessed Data Sets D - D4

(with size 20MB), and Figure 4.19 which presents the properties of data set D1 (with size 100.3MB). The same applies to the rest of the data sets.

4.2.2.5 Step Five: Processing of Big Data

The concept behind processing of the data sets is as explained in Section 4.2.1. Eclipse kepler IDE was used in the study as a workspace for writing the Java codes used for processing of the data sets. It was also used in exporting the JAR executable files used by Hadoop to run the codes. The current Inverted Indexing technique, as well as the proposed strategy were implemented on the data sets. The steps involved includes:

1. Creation of new projects in Eclipse.
2. Creation of Java classes in Eclipse that will process the data sets. Three Java classes were created for the current Indexing Strategy; the driver or main class, the mapper class, and the reducer class. The proposed Indexing Strategy has

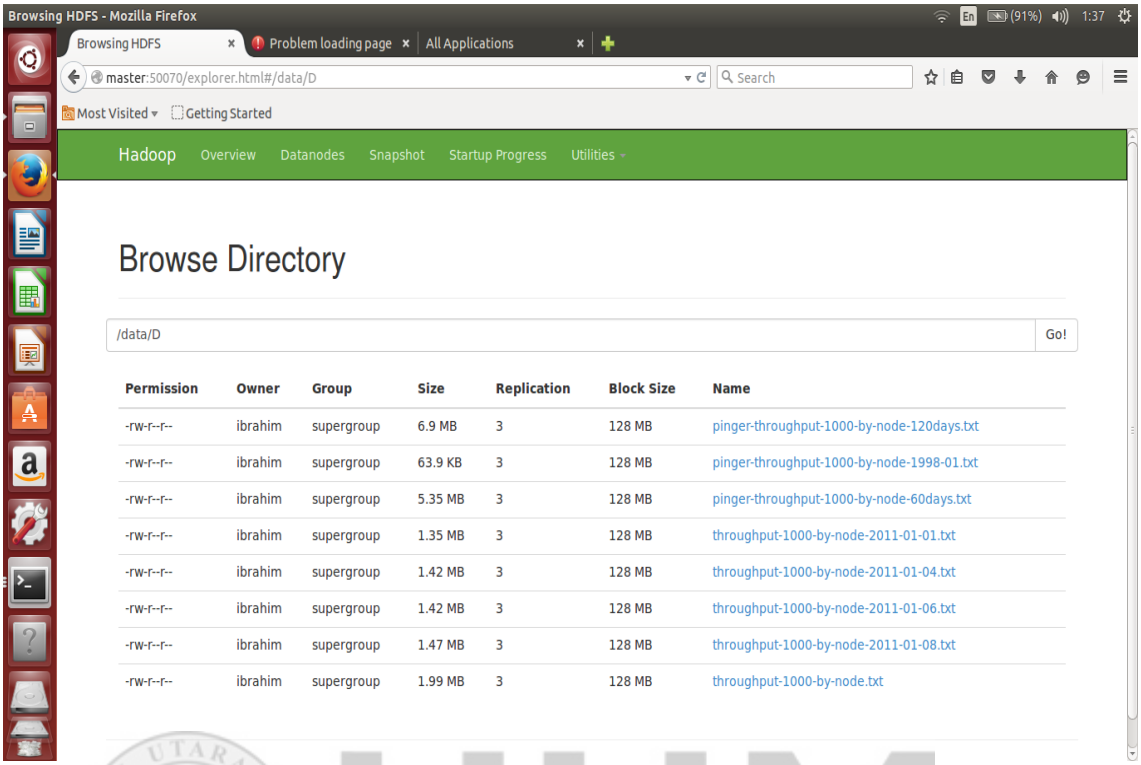

Figure 4.18: Unprocessed Data Set D

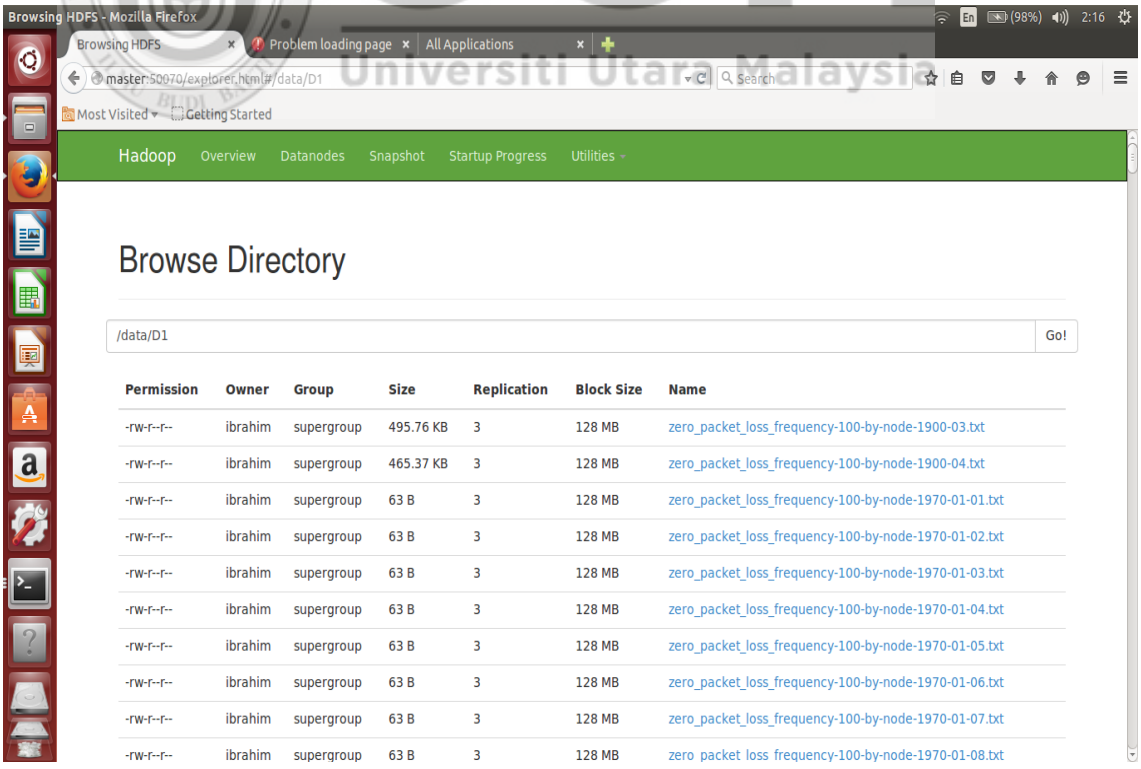

Figure 4.19: Unprocessed Data Set D1

similar number of classes, with the addition of custom InputFormat and RecordReader class.

3. Compilation of JAR files from Hadoop configuration files.
4. Exporting of the compiled JAR files and other resources to be used by Hadoop.
5. Writing specific Hadoop commands for processing of each data set.

The MapReduce process is initiated by writing a linux command that calls the JAR executable file and the main Java class to run on the specified data sets. The command includes the specification of the path where the data was stored in HDFS, and also the definition of a path where the data will be stored after the program has been executed on it. The folder for storing the processed data was created after the command has been activated. An experiment was first conducted on data set D (the sampe data set), to ensure the tools were properly configured and the codes were working efficiently. As stated, all configuration issues were dealt with and errors debbuged before the actual experiment was performed. The main experiments were conducted seperately on data sets D1, D2, D3, and D4, and for both the current Inverted Indexing technique and the proposed Indexing Strategy. The results obtained from each of the experiments conducted, was recorded for examination and evaluation of performance. The number of splits on each data set, were recorded along with the execution time (processing or CPU time). The number of splits is normally displayed on the terminal at the beginning of every job execution as illustrated in Figures 4.20, 4.21, 4.22, and 4.23. Figure 4.20 illustratres the proposed strategy being implemented on the data set D, with eight (8) splits. Also, Figure 4.21 is a MapReduce application or job in progress with the current inverted indexing technique implemented on the data set D1. Figure 4.21 shows that the file was divided into 543 splits. When the proposed strategy was implemented on data set D1 (referring to Figure 4.23), the same number of splits (543) as was assigned for the current strategy (Figure 4.21), were assigned to the mappers.

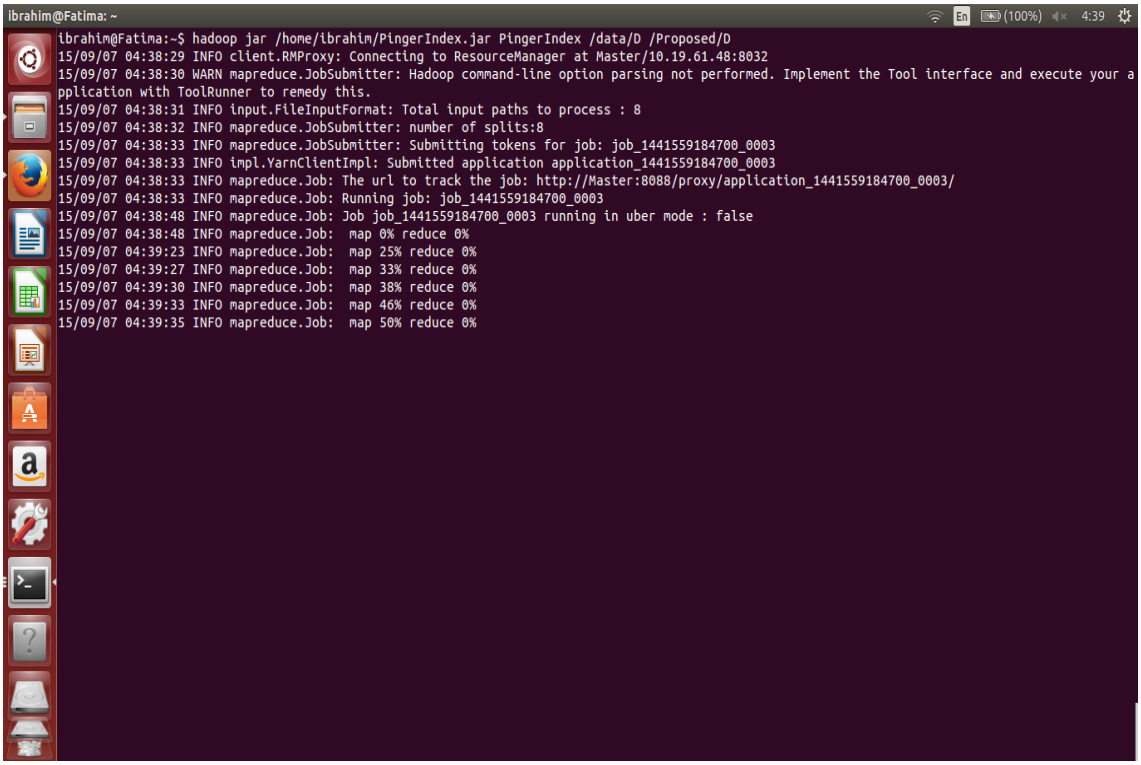

Figure 4.20: Processing of Data Set D

This proves that the same number of mappers were used in the current strategy, as well as the proposed. The same approach was applied to the rest of the data sets. From Figure 4.22, it is obvious that the number of splits increased as the data sets. The data set D4 (with size 200.4MB) was divided into 1181 splits for processing. Figure 4.24 shows a MapReduce application in progress, and Figure 4.25 presents a completed MapReduce job. The CPU time as well as other MapReduce specific informations are displayed on the terminal at the end of each job execution.

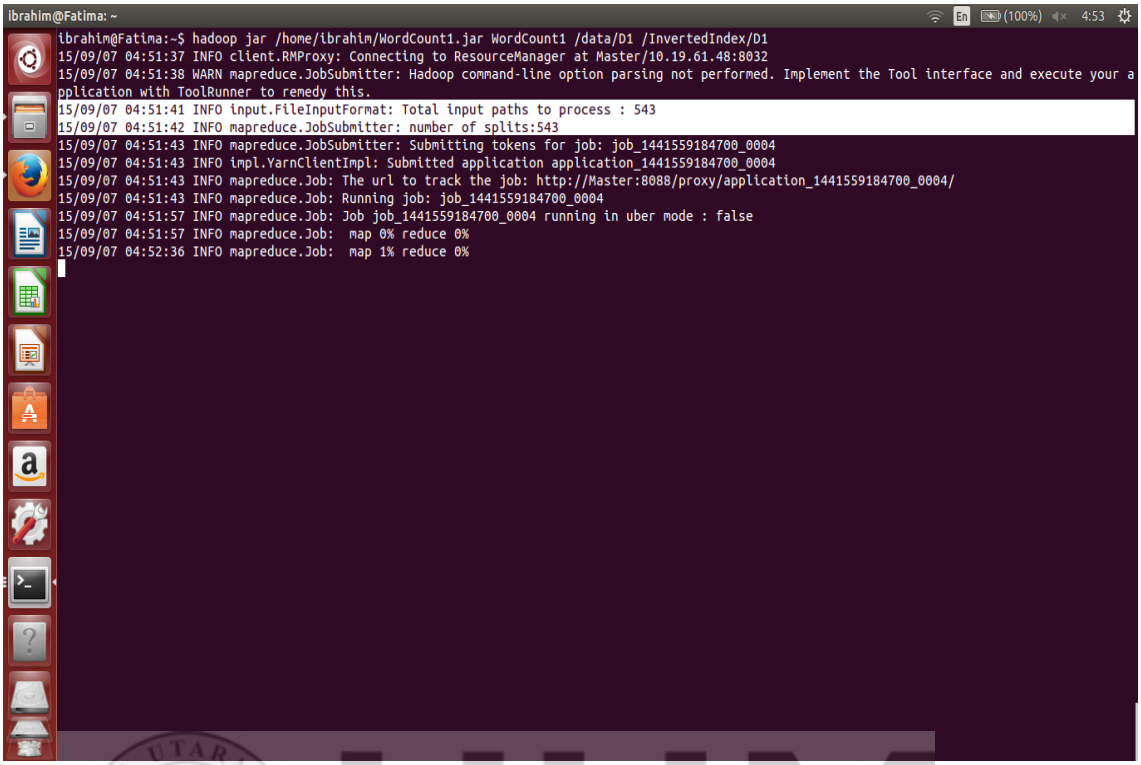

Figure 4.21: Processing of Data Set D1 with Current Strategy

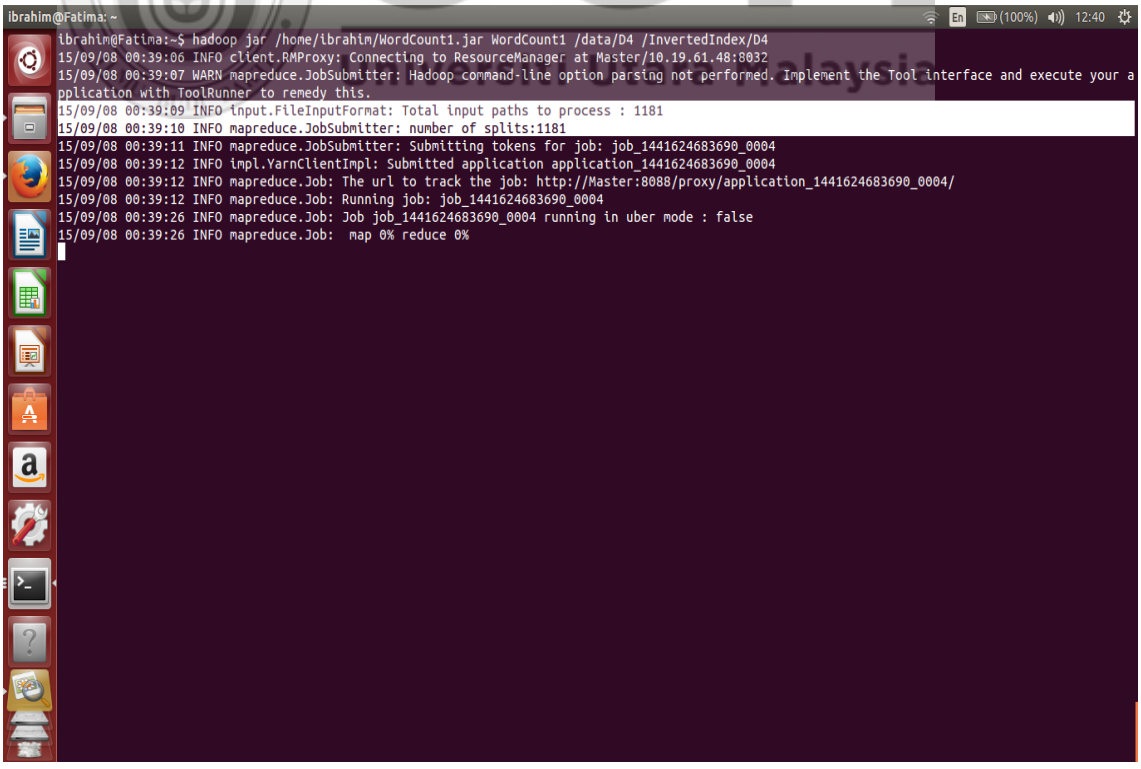

Figure 4.22: Processing of Data set D4 with Current Strategy

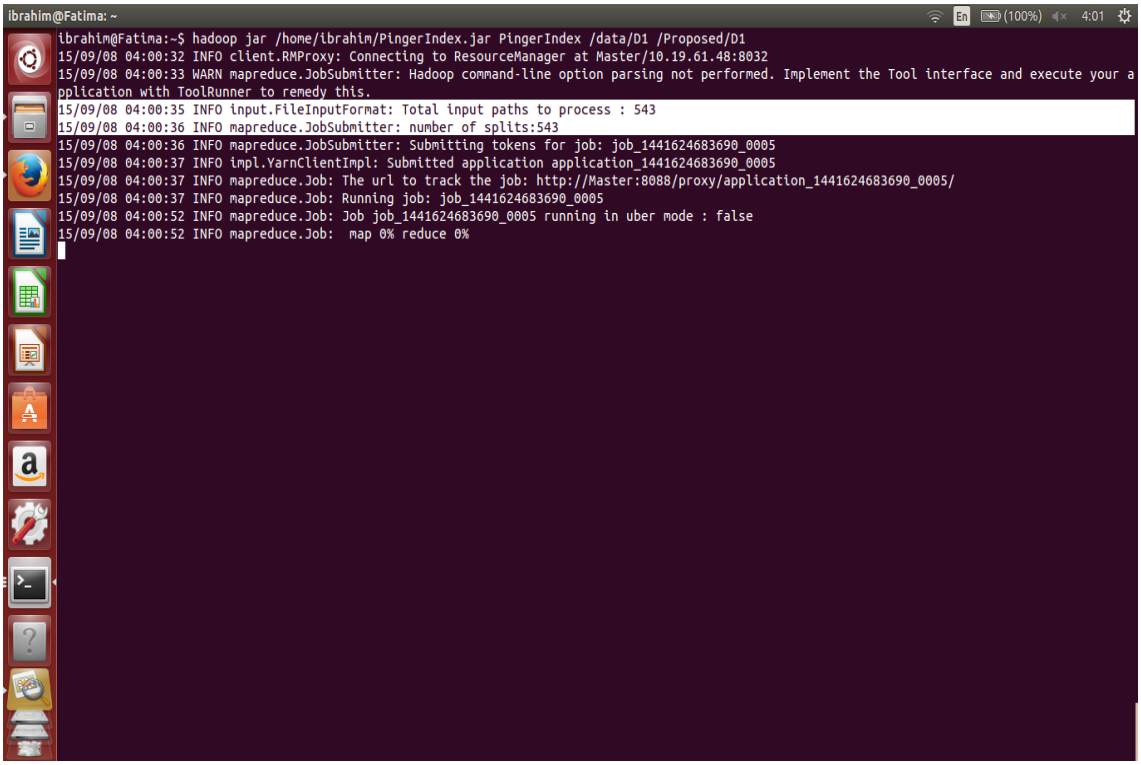

Figure 4.23: Processing of Data Set D1 with Proposed Strategy

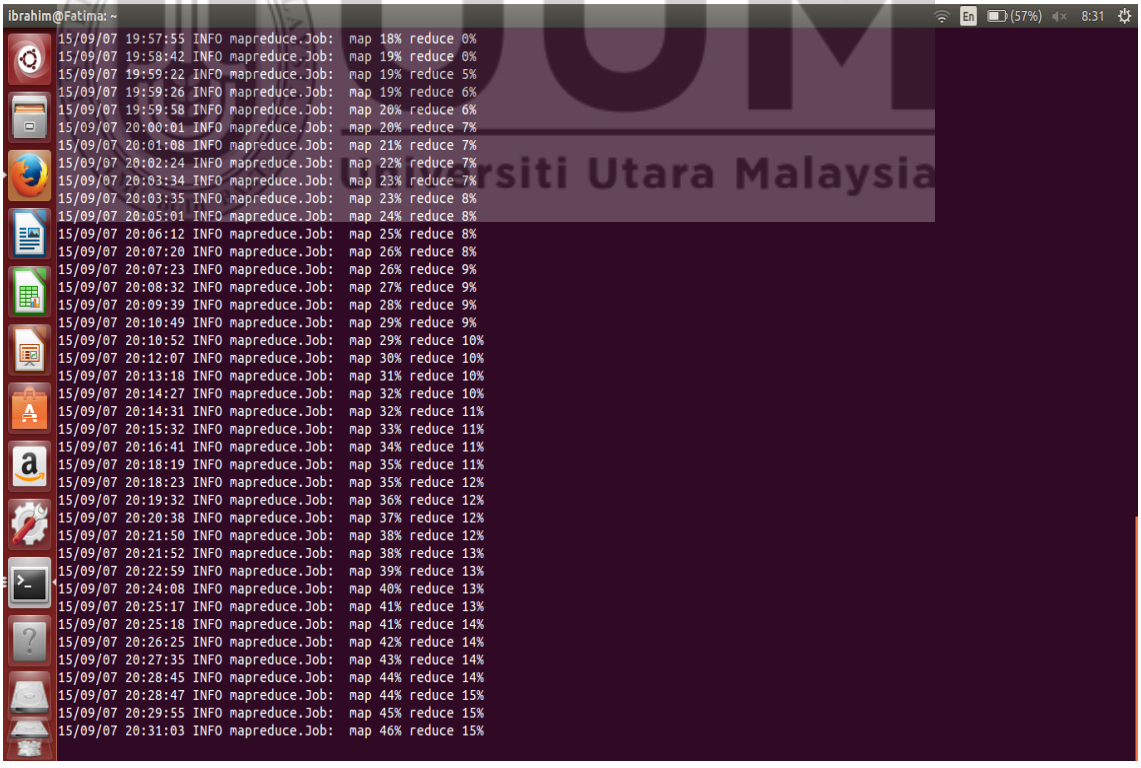

Figure 4.24: MapReduce Application in Progress

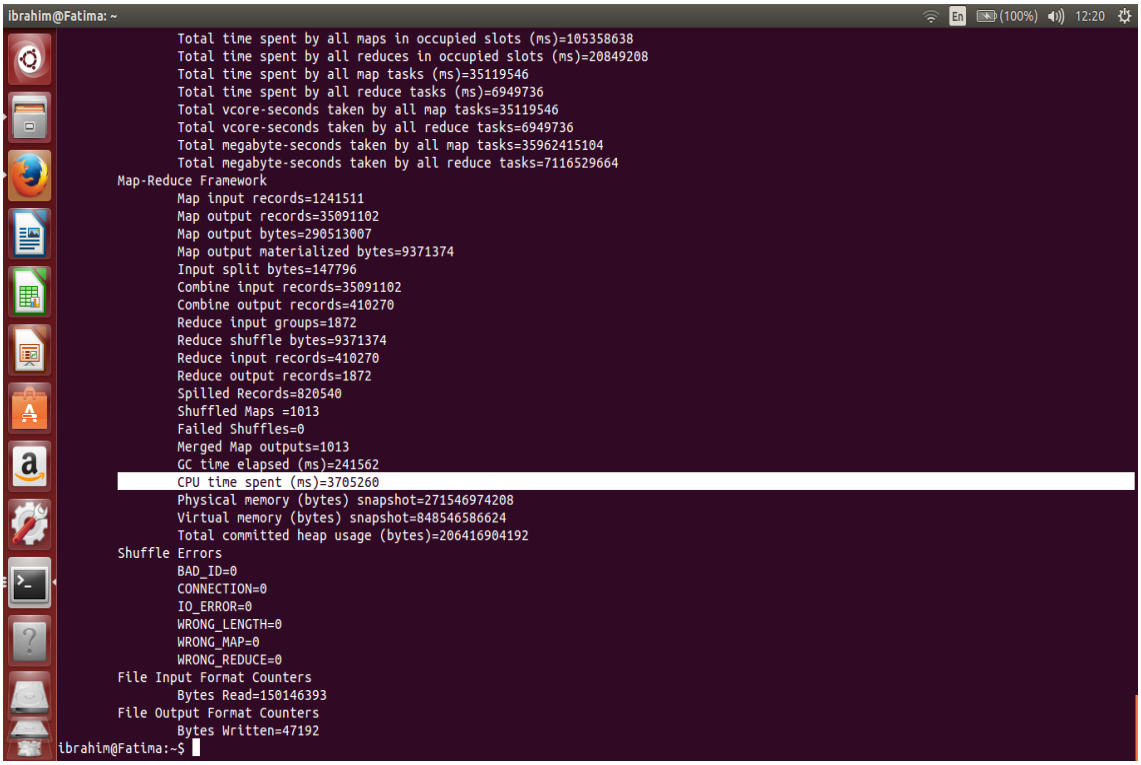

Figure 4.25: Completed MapReduce Application

4.2.2.6 Step Six: Storage of Big Data

HDFS serves as the primary storage for Hadoop. The data sets or files are loaded in it to be passed to the Mapper for processing, and are loaded back in it after processing (as results/outputs) according to the path specified for storage. The output stored in HDFS can be further processed or retrieved for analysis. In the study, the processed data were stored back in HDFS as in Figures 4.26 and 4.28 for the current strategy and the proposed strategy respectively. Similar to the loading of the data sets (Step four), the properties - file size, replication number, and block size, displays zero until each data set is opened to view its actual properties. Figures 4.27 and 4.29 displays the actual properties of the data set D (current strategy) and the data set D1 (proposed strategy) after processing. By default, HDFS names the location of successfully processed data set as _SUCCESS/part-r-00000, as presented in Figures 4.27 and 4.29.

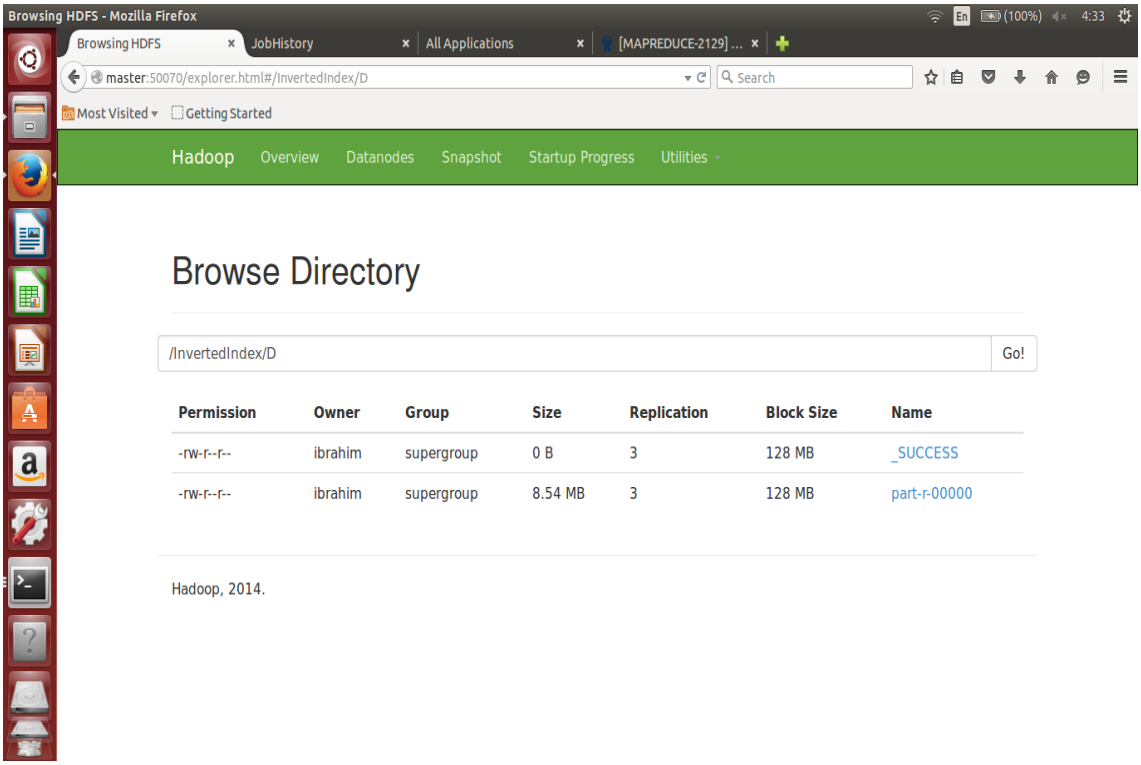

Figure 4.27: Processed Data Set D

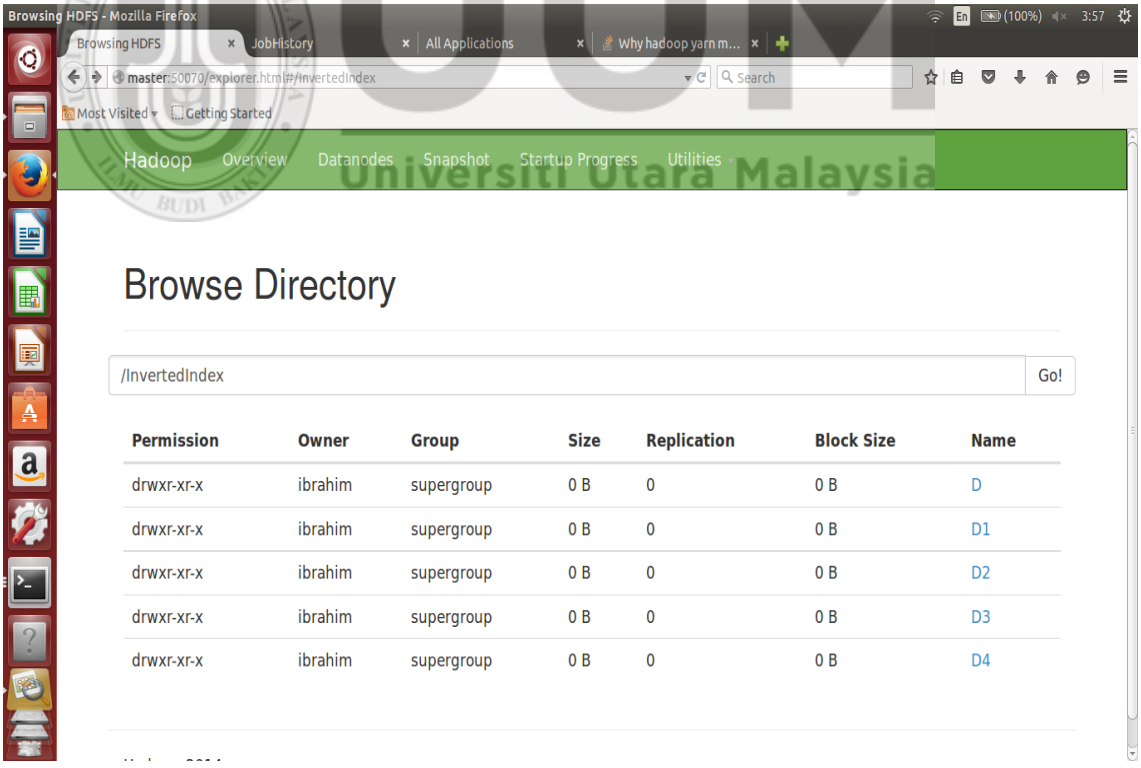

Figure 4.26: Processed Data Sets with Current Strategy

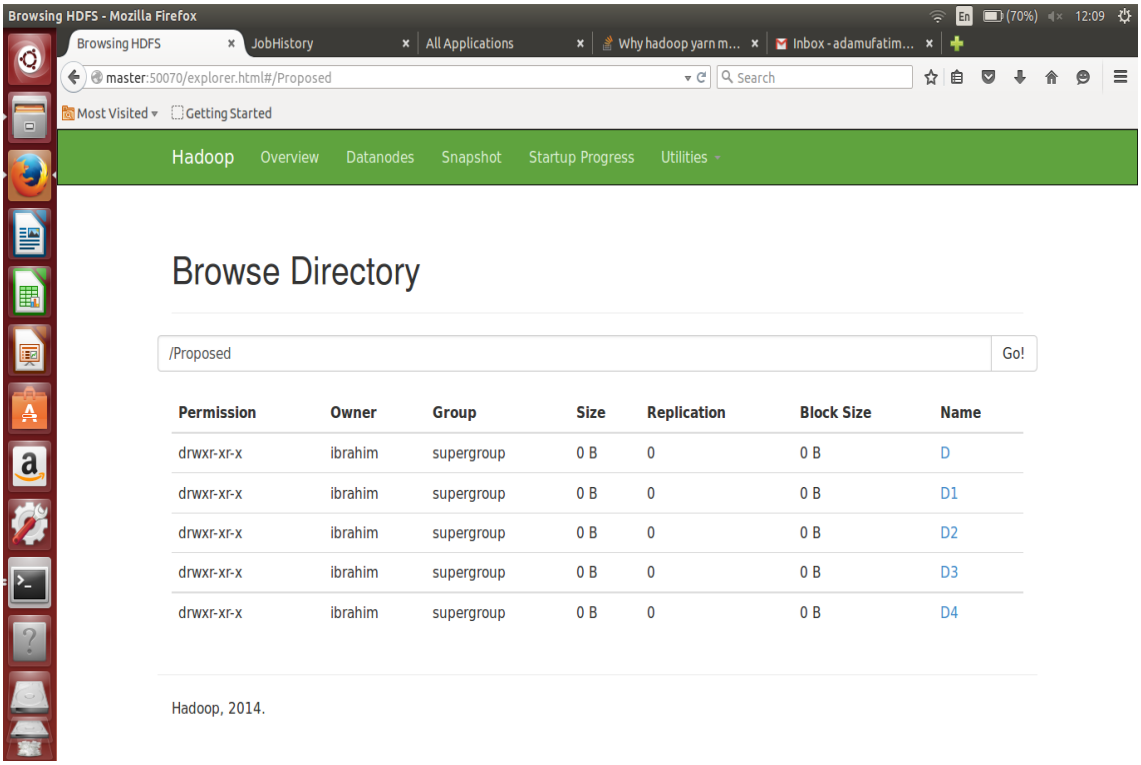

Figure 4.28: Processed Data Sets with Proposed Strategy

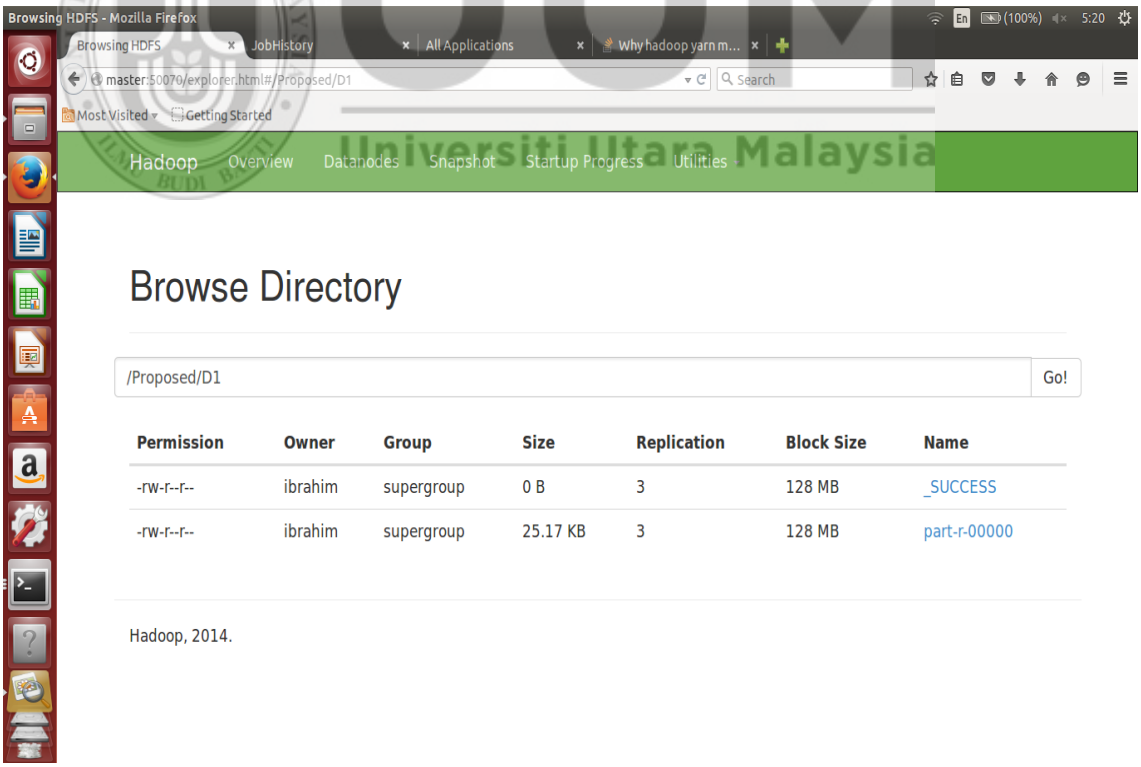

Figure 4.29: Processed Data Set D1

4.2.2.7 Step Seven: Retrieval of information or query processing

Information Retrieval from a collection of complex data sets becomes easier if processing is efficiently performed. Hive was used to test for information retrieval or query time in the study. It uses an SQL-like language known as HQL. In the study, the statement ' select * from <data set location>; ' was used in retrieving the data sets and obtaining the query time. As illustrated in Figure 4.30, the statement ' select * from d; ' was used in retrieving all the web addresses and the number of pings from the data set D. The result obtained is as illustrated in Figure 4.31, with the time taken to retrieve the information (query time) as well as the number of rows fetched, displayed under the result (on the terminal). The same approach was applied to the other data sets.

Figure 4.30: Querying the Data Sets

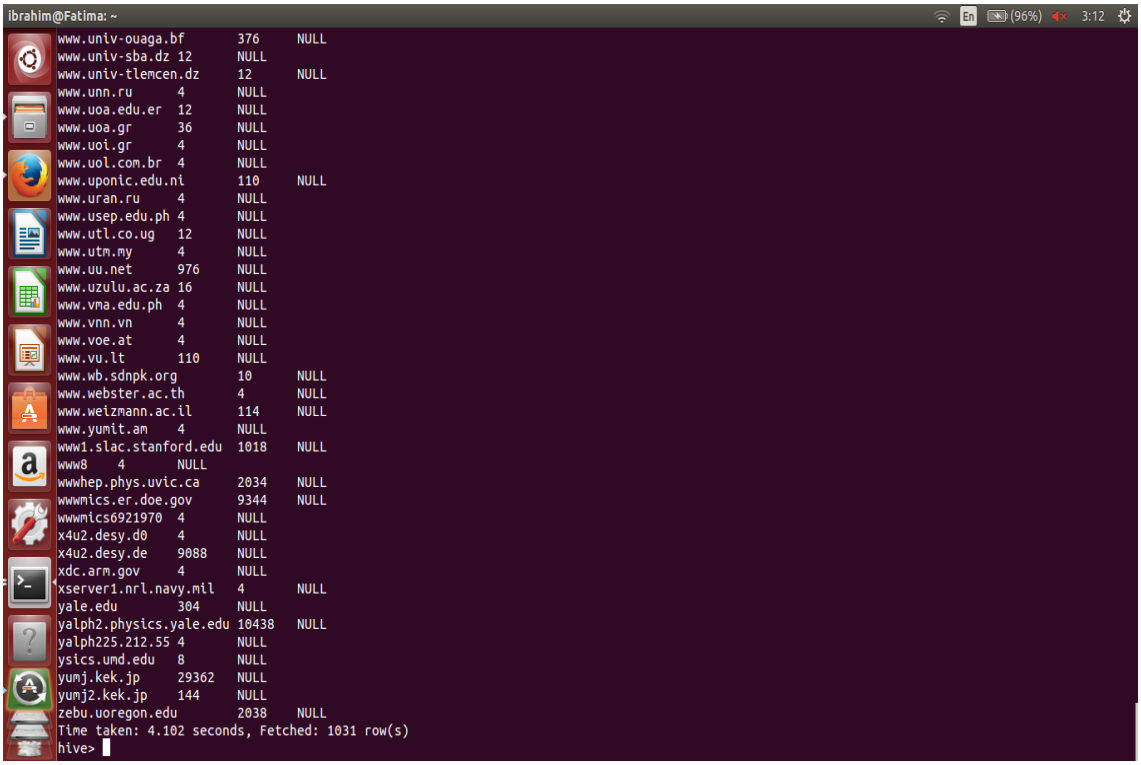

Figure 4.31: Query Result

4.3 RECORDING OF RESULTS

The metrics for evaluating the performance of the strategies are processing time and query time. The performance was evaluated with:

- time taken to process the data sets with varying number of nodes (Current Inverted Indexing strategy Versus proposed Inverted Indexing strategy)
- time taken to process varying sizes of data sets (Current Inverted Indexing strategy Versus proposed Inverted Indexing strategy)
- time taken to query varying size of data sets (Unindexed data sets Versus Indexed data sets)

The results were recorded for each case, with each case consisting of four separate experiments. The number of splits, processing or execution time, as well as query

time, were recorded against each data set for every case.

4.4 SUMMARY

This chapter presents the implementation of the proposed strategy. It starts by explaining the tools utilized in the study. It then describes the implementation of the current inverted indexing strategy using the MapReduce framework. Finally, it explains how BIND strategy was implemented, which is based on how Ian Foster's concept fits into the MapReduce framework. The following chapter (Chapter Five) presents the performance evaluation of BIND strategy and compares it with the current inverted indexing strategy.

CHAPTER FIVE
PERFORMANCE EVALUATION

This chapter discusses the results obtained from the study. It also includes limitations of the study and recommends some areas for future work. Finally, it concludes the study. Specifically, Section 5.1 is on validation, Section 5.2 presents the results obtained from the study, Section 5.3 evaluates the results (discussion), Section 5.4 discusses the limitations, Section 5.5 suggests some future works, and Section 5.6 concludes the study.

5.1 VALIDATION OF BIND STRATEGY

The validation of the study follows some analytical assumptions where the splits, task, and mappers are changing variables. A change in the number of splits, causes a change in the number of mappers. Also, the number of pending task (which is not equal to the number of splits or mappers), keeps increasing until all the jobs are completed. The validation of the study is as follows:

where:

S = splits

T = task

M = mapper

A split S (or job) has a number of task and mapper represented as :

$S = T.M$

Let:

$S = \delta \frac{T}{M} = \delta T^{-M}$

$S = S_j = \delta \frac{T_i}{M_j}$

where:

$i = 0, 1, 2, \_ \_ \_ n$

$j = 1, 2, \_ \_ \_ m$

In general;

$S = \delta \frac{T_i}{M_j}, \delta \frac{T_k}{M_l}, \delta \frac{T_p}{M_q}, —, \delta \frac{T_y}{M_z}$

For the proposed strategy:

$S^1 = \delta \frac{T_i}{M_j} + \theta$

where:

θ= *pending task*

$\theta \neq 0$

$\forall$ *M that has completed task is assigned a new task asymetrically:*

$$S^1 = \delta \frac{T_i}{M_j} + \theta_u, \delta \frac{T_k}{M_l} + \theta_v, \delta \frac{T_p}{M_q} + \theta_w \text{ , —, } \delta \frac{T_y}{M_z} + \theta_x$$

where $x \neq y \neq z$

$\delta \frac{T_y}{M_z} + \theta_x$ signifies the completion of a job in a non-ordered fashion.

5.2 PERFORMANCE ANALYSIS

Several experiments were conducted by running sets of test cases using hadoop. Each test case consist of four test with varying number of data sets. The experiments were performed on a 2-node cluster, and also on a 3-node hadoop cluster. Each node was operating on Ubuntu 14.04 O.S with four processor cores and 4GB of RAM. A number of reruns were necessary on all experiments due to network signal interference, node proximity, and hadoop natural overhead which is caused by inter-process communication. The test cases are explained as follows:

5.2.1 Case One: Execution time using the current strategy (2-node cluster)

A hadoop cluster consisting of 2-node was used to test for the execution time (processing time) using the current strategy. The results obtained from Case 1, is as illustrated on Table 5.1. Table 5.1 shows that 543 - 1181 number of splits were used for varying sizes of data sets, with an increase in split as the data sizes increases. This represents the number of fragments the data sets were divided into, and also translates into the number of mappers that carried out the jobs. Table 5.1 also shows the execution time obtained after the experiments were conducted. The time taken to process data set D1 (with size 50.1MB) is 2520900 milliseconds (as reported by hadoop), which gives 2520.9 seconds. 3922.92 seconds is obtained from data set D2, 4481.46 seconds from

data set D3, and 5862.66 seconds from data set D4. From Table 5.1, it is observed that the execution time in this case ranges from approximately 2500 seconds to 5900 seconds.

Table 5.1: Case One: The Execution Time of the Current Strategy (2-node Cluster)

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	2520900	2520.9
D2	835	3922920	3922.92
D3	1013	4481460	4481.46
D4	1181	5862660	5862.66

5.2.2 Case Two: A rerun of case one

A number of tests were conducted on the same set-up parameter as in Case 1. That is, the current strategy was implemented on the data sets several times using a 2-node cluster. Reruns were necessary to re-confirm the result obtained in Case 1. The results obtained from Case 2 (a rerun of Case 1) is as presented on Table 5.2. Table 5.2 shows that the same data sets, number of splits as well as number of nodes were used as in Case 1 (showing that the experiment was run on the same set-up as in Case 1). A slight variance was noticed in the values obtained as the time taken to execute data set D1 is observed to be 2512.32 seconds (refer to Table 5.2), with similar variance obtained as execution time for the other data sets. Although the values obtained slightly differs from that of Case 1, the execution time obtained still falls in the range of 2500 seconds to 5900 seconds for all rerun tests.

Table 5.2: Case Two: The Execution Time of the Current Strategy (2-node Cluster) - Rerun

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	2512320	2512.32
D2	835	3913440	3913.44
D3	1013	4543260	4543.26
D4	1181	5857980	5857.98

5.2.3 Case Three: Execution time using the proposed strategy (2-node cluster)

A hadoop cluster consisting of 2-node was used to test for the execution time using the proposed strategy. The results obtained from Case 3, is as presented on Table 5.3. Table 5.3 shows that the data set D1 was processed within 2388.3 seconds. The execution time 3651.66 seconds, was obtained from processing data set D2, 4413.12 seconds from dat set D3, and 5585.82 seconds from data set D4. The result shows that the execution time obtained ranges from approximately 2300 seconds to 5600 seconds. This explains that although the same set-up parameters were used as in Case 1 and Case 2, lower values were obtained (as compared to Case 1 and Case 2) indicating a performance gain.

sugge

Table 5.3: Case Three: The Execution Time of the Proposed Strategy (2-node Cluster)

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	2388300	2388.3
D2	835	3651660	3651.66
D3	1013	4413120	4413.12
D4	1181	5585820	5585.82

5.2.4 Case Four: A rerun of case three

The same set-up as in Case 3 was used to run several tests to re-confirm the results obtained from Case 3. The results obtained from Case 4 (a rerun of Case 3) is as shown on Table 5.4. Conducting the experiments on the same data sets as in Case 3, the execution time obtained ranges from approximately 2300 seconds to 5700 seconds for all rerun cases. The results obtained from running the experiment several times produces similar values falling within this range.

Table 5.4: Case Four: The Execution Time of the Proposed Strategy (2-node Cluster) - Rerun

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	2370840	2370.84
D2	835	3796620	3796.62
D3	1013	4212540	4212.54
D4	1181	5706780	5706.78

5.2.5 Case Five: Execution time using the current strategy (3-node cluster)

A hadoop cluster consisting of 3-node was used to test for the execution time using the current strategy. The results obtained from Case 5 is as presented on Table 5.5. Table 5.5 shows that the time taken to process the data set D1 using the current srategy on a 3-node cluster is 1822.15 seconds. It aslo shows that D2 was processed within 3007.98 seconds, D3 within 3705.26, and D4 was processed within 4352.58 seconds. The execution time obtained ranges from 1800 seconds to 4400 seconds, which is lower than the values obtained as in Case 1 to Case 4. This explains that on a 3-node cluster or with an increase in the number of nodes, lower values were obtained indicating a better performance. This is as a result of an increase in the computational resources utilized during the experiment.

Table 5.5: Case Five: The Execution Time of the Current Strategy (3-node Cluster)

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	1822150	1822.15
D2	835	3007980	3007.98
D3	1013	3705260	3705.26
D4	1181	4352580	4352.58

5.2.6 Case Six: A rerun of case five

The same set-up as in Case 5 was used in running several experiments. The results obtained from Case 6 (a rerun of Case 5) is as shown on Table 5.6. As observed on Table 5.6, the execution time obtained ranges from 1800 seconds to 4400 seconds,

which is the same as in Case 5. The results obtained from running the experiment several times produces similar values falling within this range. This confirms that lower execution times are obtained as the number of nodes increases suggesting a better performance.

Table 5.6: Case Six: The Execution Time of the Current Strategy (3-node Cluster) - Rerun

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	1882260	1882.26
D2	835	3248940	3248.94
D3	1013	3702660	3702.66
D4	1181	4336260	4336.26

5.2.7 Case Seven: Execution time using the proposed strategy (3-node cluster)

A hadoop cluster consisting of 3-node was used to test for the execution time using the proposed strategy. In this case, the data set D1 was processed within 1783.26 seconds. The time taken to process the data set D2 was 2904.96 seconds, D3 took 3500.37 seconds, and D4 took 4142.26 seconds. The results obtained from Case 7 is as presented on Table 5.7, with a range of approximately 1700 seconds to 4200 seconds representing the values obtained as the execution time. It is observed that lower execution times were obtained as compared to Case 1 to 6 indicating an increase in performance. This explains that implementing the proposed strategy on a data set, a performance gain is observed with an increase in the number of nodes.

Table 5.7: Case Seven: The Execution Time of the Proposed Strategy (3-node Cluster)

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	1783260	1783.26
D2	835	2904960	2904.96
D3	1013	3500370	3500.37
D4	1181	4142260	4142.26

5.2.8 Case Eight: A rerun of case seven

Series of tests were conducted using the same parameters as in case 7 to re-confirm the results obtained. The result from Case 8 (a rerun of Case 7) is as presented on Table 5.8. On Table 5.8, the data set D1 was processed in 1811.94 seconds, D2 in 2905.38 seconds, D3 in 3566.82 seconds, and D4 in 4158.48 seconds. The execution time obtained from the rerun (Case 8) ranges between 1700 seconds to 4200 seconds, which is similar to the results obtained from Case 7 and from other reruns.

Table 5.8: Case Eight: The Execution Time of the Proposed Strategy (3-node Cluster) - Rerun

Data Set	Number of Splits	Execution Time (ms)	Execution Time (s)
D1	543	1811940	1811.94
D2	835	2905380	2905.38
D3	1013	3566820	3566.82
D4	1181	4158480	4158.48

5.2.9 Case Nine: Retrieval time (query time) on unindexed data sets

Table 5.9 presents the results obtained from querying the unindexed data sets. As presented on Table 5.9, it took 33.924 seconds to retrieve information from data set D1, 52.346 seconds from data set D2, 85.811 seconds from data set D3, and 127.051 seconds from data set D4. The query time obtained ranges from approximately 33 seconds to 128 seconds on data sets D1, D2, D3, and D4.

Table 5.9: Case Nine: Query Time with Unindexed Data Sets

Data Set	Query Time (s)
D1	33.924
D2	52.346
D3	85.811
D4	127.051

5.2.10 Case Ten: A rerun of case nine

Case 9 was rerun several times to re-confirm the values obtained. A rerun of case 9 with the results obtained is as presented on Table 5.10. From Table 5.10, it took 35.023 seconds to retrieve information from data set D1, 59.104 seconds from data set D2, 90.122 seconds from data set D3, and 121.542 seconds from data set D4. The query time obtained ranges from approximately 35 seconds to 122 seconds, which is similar to the values obtained from all reruns, and which is not far from the values obtained in Case 9.

Table 5.10: Case Ten: Query Time with Unindexed Data Sets - Rerun

Data Set	Query Time (s)
D1	35.023
D2	59.104
D3	90.122
D4	121.542

5.2.11 Case Eleven: Query time on the indexed data sets

Case 11 is a test to obtain the information retrieval time (query time) on the indexed data sets. As presented on Table 5.11, it took 11.101 seconds to retrieve information from data set D1, 18.339 seconds from data set D2, 27.893 seconds from data set D3, and 35.095 seconds from data set D4. The result from Case 11 is presented on Table 5.11. The query time obtained ranges from approximately 11 seconds to 36 seconds. It is observed that a significant decrease in the query time is obtained as compared to Case 9 and Case 10. This proves that the proposed strategy facilitates information retrieval on data sets.

Table 5.11: Case Eleven: Query Time with Indexed Data Sets

Data Set	Query Time (s)
D1	11.101
D2	18.339
D3	27.893
D4	35.095

5.2.12 Case Twelve: A rerun of case eleven

Several reruns of Case 11 were conducted to re-confirm the result obtained. The results from a rerun case is presented on Table 5.12. From Table 5.12 (a rerun of Case 11), it took 10.342 seconds to retrieve information from data set D1, 18.411 seconds from data set D2, 28.021 seconds from data set D3, and 35.155 seconds from data set D4. The query time obtained ranges from approximately 10 seconds to 36 seconds, similar to the values from Case 11 and other reruns. A significant decrease in the query time indicates that the proposed strategy facilitates information retrieval on data sets.

Table 5.12: Case Twelve: Query Time with Indexed Data Sets - Rerun

Data Set	Query Time (s)
D1	10.342
D2	18.411
D3	28.021
D4	35.155

A summary of the results for all cases is as presented on Table 5.13 and Table 5.14 for 2-node cluster and 3-node cluster respectively. Tables 5.13 and 5.14 presents the average values computed from running the experiments several times. The results were obtained using the same number of splits, implying that the same number of mappers carried out the jobs for both the current and the proposed strategy. From Table 5.13, the current strategy returns an average execution time of 2516.6 seconds for data set D1, which is greater than the execution time obtained from implementing the proposed strategy on the same data set D1 (which is 2379.52). This is observed for the other data sets also, where an execution time of 3918.2 seconds for the current strategy is obtained against an execution time of 3724.14 seconds for the proposed strategy (on the data set D2). A decrease in the average execution time obtained from the proposed strategy as compared to the current strategy implies that the proposed strategy performs better. Also on Table 5.14, the execution time obtained from implementing

the current strategy on the data set D1 using a 3-node cluster is 1852.2 seconds. On the same condition, the proposed strategy returns an average execution time of 1797.6 seconds on the data set D1. For the data set D2, the current strategy returns an average execution time of 3128.5 seconds which is less than 2905.2 seconds, obtained from the proposed strategy. The decrease in execution time obtained from the proposed strategy implies that the proposed strategy performances better than the current strategy with an increase in the number of nodes. On Table 5.13, the average time it took to retrieve information from the unindexed data set D1 is 34.474 seconds and 10.722 seconds from the indexed data set D1(using the proposed strategy). For the data set D2, it took 55.724 seconds to retrieve information from the unindexed data set, and 18.374 seconds from the indexed data set. A significant decrease is observed from the values obtained when the indexed data sets are queried. This indicates that the proposed strategy facilitates information retrieval when implemented on data sets.

Table 5.13: Results obtained using 2-node cluster

Data Set	Number of Splits	Execution Time (s)		Query Time (s)	
		Current Strategy	Proposed Strategy	Unindexed Data Set	Indexed Data Set
D1	543	2516.6	2379.52	34.474	10.722
D2	835	3918.2	3724.14	55.724	18.374
D3	1013	4512.4	4312.8	87.967	27.961
D4	1181	5860.32	5646.3	124.296	35.125

Table 5.14: Results obtained using 3-node cluster

Data Set	Number of Splits	Execution Time (s)		Query Time (s)	
		Current Strategy	Proposed Strategy	Unindexed Data Set	Indexed Data Set
D1	543	1852.2	1797.6	34.474	10.722
D2	835	3128.5	2905.2	55.724	18.374
D3	1013	3703.96	3533.6	87.967	27.961
D4	1181	4344.4	4150.37	124.296	35.125

5.3 EVALUATION OF RESULTS (DISCUSSION)

Figure 5.1 illustrates that using a 2-node cluster (small scale), the time taken to process the data sets using the proposed strategy is less than that of the the current strategy.

This proves that an increase in performance is obtained from the proposed strategy as compared to the current strategy.

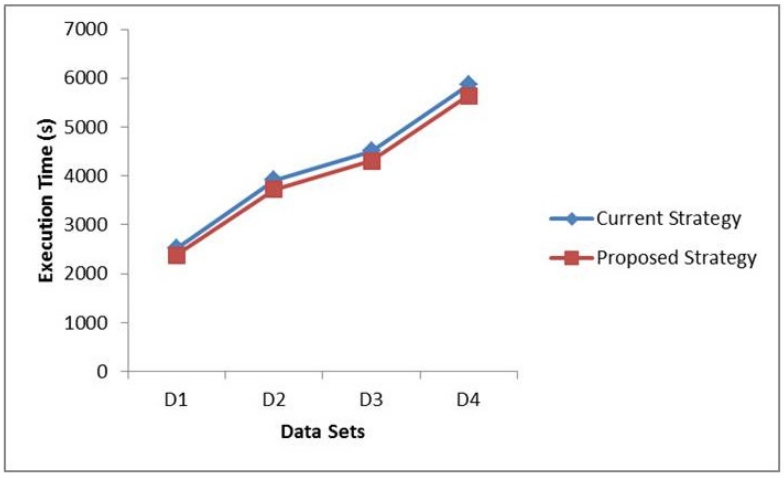

Figure 5.1: Execution time using 2-node cluster

Figure 5.2 shows a comparison of the execution time for the current strategy and the proposed strategy using 3-node. It is observed that the proposed strategy produces lesser values as compared to the current strategy. This explains that the performance of the proposed strategy does not decline with increase in nodes. It also shows that the proposed strategy still performes better than the current strategy.

Figure 5.3 shows the difference in information retrieval time between the indexed data sets and the unindexed data sets. A decrease in the query time obtained, is observed

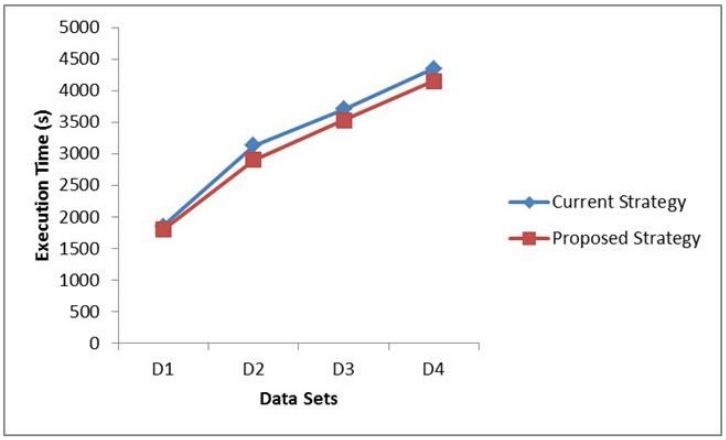

Figure 5.2: Execution time using 3-node cluster

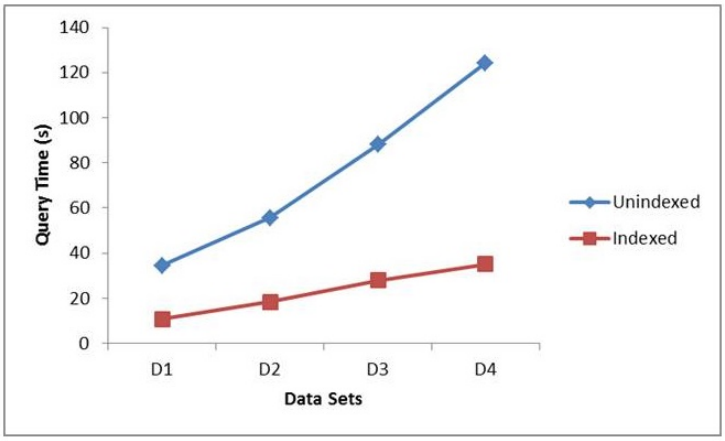

Figure 5.3: Query Time

on the indexed data sets. This proves that the proposed strategy facilitates query on data sets.

Figure 5.4 is a summary of the performance of the proposed strategy and the current strategy. In general, it is observed that although the value of the execution time increases with an increase in the size of data sets, the proposed strategy still performs better than the current strategy in all cases. Figure 5.4 illustrates that there is a performance gain when the proposed strategy is implemented on a 2-node cluster as compared to when the current strategy is applied on a 2-node cluster using the same sizes of data sets. Also, the same performance is obtained when implemented on a 3-node cluster. In addition, a significant decrease in the execution time is observed when a 3-node cluster was used as compared to the execution time obtained with a 2-node cluster. This is due to the additional computational resources (processing and memory/storage resources) used. The proposed strategy still produces lesser values as compared to the current strategy when the number of nodes is increased. This proves that the performance of the proposed strategy does not decline with increase in size of the data set, as well as nodes (as noted in all cases). Hence, the performance of the proposed strategy improves with increase in nodes, and therefore, could be implemented (and would perform better) on thousands of clusters as could be found in large organizations. In a

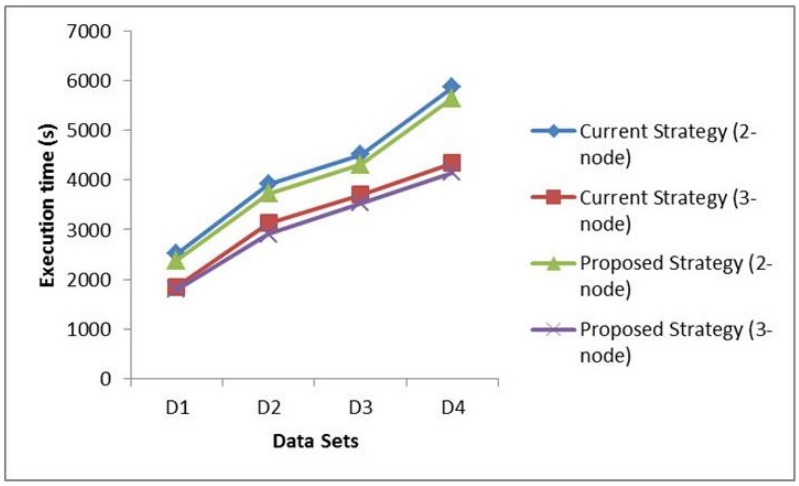

Figure 5.4: Overall Execution Time

nutshell, the proposed strategy excels the currents strategy in all cases tested.

5.4 LIMITATIONS

The study was faced with some challenges such as limitations in computational resources, network issues, and time. The amount of computational resources used in the study as well as the study time frame, were limited. Also, frequent weak network signals could affect the results obtained from the experiments, and also prolong the time taken to complete the experiments. Due to these limitations, the study could not be conducted on a very large data set.

5.5 FUTURE WORK

Future works related to this study proposes an implementation of this approach using other types of data sets, and based on other indexing strategies. The expectation is a prove that the proposed strategy is applicable to other data types, as well as other strategies. More promising is the extension of this approach for the purpose of creating semantic indexes, as well as implementing it in real time scenarios.

5.6 CONCLUSION

The objective of the study is to propose the design of a Big Data indexing strategy called BIND strategy, for processing of Big Data using the MapReduce framework. The proposed strategy facilitates processing of Big data, as well as storage and information retrieval. The strategy was designed and implemented using Java programming language. The program was executed a number of times on varying data sets (PingER historical log data), as well as varying amount of nodes to compare and analyze the results.

The overall results obtained, indicates that the proposed strategy produces lower values than the current strategy. It is noted that the proposed strategy still performs well with increase in nodes, and query is facilitated when the proposed strategy is implemented on data sets. The study has answered the research questions listed in chapter one. The time taken to process a given amount of data using the proposed strategy has been minimized. This is supported by the results obtained from the study. Users and organizations that manage Big Data will find the proposed strategy helpful in the processing of Big data.

REFERENCES

[1] D. Bouquin. (2015, May) R and data mining. [Online]. Available: http://med.cornell.libguides.com/HINF5008

[2] A. Marco. (2012,, October.) Driving big data. driving big data.

[3] Vishnur. (2014,, March) Hadoop. [Online]. Available: https://hadoop.apache.org/

[4] a. M. R. Sivaraman, E., "High performance and fault tolerant distributed file system for big data storage and processing using hadoop." in *Intelligent Computing Applications (ICICA), 2014 International Conference on, (pp. 32-36).*, 2014, March.

[5] R. Zhang, D. Hildebrand, and R. Tewari, "In unity there is strength: Showcasing a unified big data platform with mapreduce over both object and file storage," in *Big Data (Big Data), 2014 IEEE International Conference on*, Oct 2014, pp. 960–966.

[6] J. Li, Z. Xu, Y. Jiang, and R. Zhang, "(2014, aug). the overview of big data storage and management. cognitive informatics cognitive computing (icci*cc),," in *IEEE 13th International Conference on, (pp. 510-513*, 2014.

[7] C. Liu, J. Y. Chen, L. Z. Ranjan, X. R., Zhang, C. Yang, D. Georgakopoulos, and J. Chen, "Public auditing for big data storage in cloud computing -," in *A Survey. Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on, (pp. 1128-1135).*, 2013, Dec.

[8] H. Tan, W. Luo, and L. M. Ni, "Clost: A hadoop-based storage system for big spatio-temporal data analytics." in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (pp. 2139-2143). New York, NY, USA: ACM.*, 2012.

[9] W. Zhou, C. Yuan, R. Gu, and Y. Huang, "Large scale nearest neighbors search based on neighborhood graph," in *Advanced Cloud and Big Data (CBD), 2013 International Conference on*, Dec 2013, pp. 181–186.

[10] M. Cheminod, L. Durante, L. Seno, and A. Valenzano, "On the description of access control policies in networked industrial systems," in *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*, May 2014, pp. 1–10.

[11] A. Desai, P. Garg, and P. Madhusudan, "Natural proofs for asynchronous programs using almost-synchronous reductions," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '14. New York, NY, USA: ACM, 2014, pp. 709–725. [Online]. Available: http://doi.acm.org/10.1145/2660193.2660211

[12] E. Bertino and B. Samanthula, "Security with privacy - a research agenda," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, Oct 2014, pp. 144–153.

[13] M. Herland, T. Khoshgoftaar, and R. Wald, "Survey of clinical data mining applications on big data in health informatics," in *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, vol. 2, Dec 2013, pp. 465–472.

[14] J. Patel and P. Sharma, "Big data for better health planning." in *Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on, (pp. 1-5).*, 2014, Aug.

[15] M. Ali-ud-din Khan, M. Uddin, and N. Gupta, "Seven v's of big data understanding big data to extract value," in *American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1 Conference of the*, April 2014, pp. 1–5.

[16] W. Li, W. Changyao, H. Pengyu, and a. A. A. Kaifen, S., "Cotton area estimation using muti-sensor rs data and big plot survey in xinjiang. (pp. 1-5). aug," in *Agro-geoinformatics (Agro-geoinformatics 2014), Third International Conference on,*, 2014,.

[17] H. Nakada, H. Ogawa, and T. Kudoh, "Stream processing with bigdata: Sss-mapreduce," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, Dec 2012, pp. 618–621.

[18] A. Datta and F. Oggier, "Storage codes: Managing big data with small overheads," in *Network Coding (NetCod), 2013 International Symposium on*, June 2013, pp. 1–6.

[19] A. Desai, P. Garg, and P. Madhusudan, "Natural proofs for asynchronous programs using almost-synchronous reductions," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '14. New York, NY, USA: ACM, 2014, pp. 709–725. [Online]. Available: http://doi.acm.org/10.1145/2660193.2660211

[20] R. Zhang and R. Hildebrand, D.and Tewari, "In unity there is strength: Showcasing a unified big data platform with mapreduce over both object and file storage." in *Big Data (Big Data), 2014 IEEE International Conference on, (pp. 960-966).*, (2014, Oct).

[21] N. Hu, B.and Carvalho, L. Laera, and T. Matsutsuka, "Towards big linked data: A large-scale, distributed semantic data storage." in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, (2012).

[22] T. Chardonnens, "Big data analytics on high velocity streams," Master's thesis, University of Fribourg (Switzerland), June 2013. [Online]. Available: http://exascale.info/students_projects/Chardonnens.pdf

[23] C. Liu, J. Chen, L. Yang, X. Zhang, C. Yang, and R. Ranjan, "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates." *Parallel and Distributed Systems, IEEE Transactions on, 25(9), 2234-2244.*, 2014, Sept.

[24] A. Alnafoosi and T. Steinbach, "An integrated framework for evaluating big-data storage solutions - ida case study," in *Science and Information Conference (SAI), 2013*, Oct 2013, pp. 947–956.

[25] R. Grunzke, R. Muller-Pfefferkorn, R. Jakel, J. Hesser, N. Kepper, M. Hausmann, J. Starek, S. Gesing, M. Hardt, V. Hartmann, J. Potthoff, and S. Kindermann, "Device-driven metadata management solutions for scientific big data use cases," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, Feb 2014, pp. 317–321.

[26] C. Wei-Chun, C. Yu-Jung, C. Chien-Chih, L. Der-Tsai, and H. Jan-Ming, "Optimizing a mapreduce module of preprocessing high-throughput dna sequencing data," in *Big Data, 2013 IEEE International Conference on*, Oct 2013, pp. 1–6.

[27] K. Fasolin, R. Fileto, M. Krugery, D. Kaster, M. Ferreira, R. Cordeiro, A. Traina, and C. Traina, "Efficient execution of conjunctive complex queries on big multimedia databases," in *Multimedia (ISM), 2013 IEEE International Symposium on*, Dec 2013, pp. 536–543.

[28] R. Irudeen and S. Samaraweera, "Big data solution for sri lankan development: A case study from travel and tourism," in *Advances in ICT for Emerging Regions (ICTer), 2013 International Conference on*, Dec 2013, pp. 207–216.

[29] I. Giangreco, I. Al Kabary, and H. Schuldt, "Adam - a database and information retrieval system for big multimedia collections," in *Big Data (BigData Congress), 2014 IEEE International Congress on*, June 2014, pp. 406–413.

[30] T. Gollub, M. Volske, M. Hagen, and B. Stein, "Dynamic taxonomy composition via keyqueries," in *Digital Libraries (JCDL), 2014 IEEE/ACM Joint Conference on*, Sept 2014, pp. 39–48.

[31] Y. Hu, H.and Wen, T. Chua, and X. S. Li, "Toward scalable systems for big data analytics: A technology," *Tutorial. Access, IEEE, 2, 652-687.*, (2014).

[32] G. Press. (2013,, September) A very short history of big data. a very short history of big data.

[33] D. Garlasu, V. Sandulescu, I. Halcu, G. Neculoiu, O. Grigoriu, M. Marinescu, and V. Marinescu, "A big data implementation based on grid computing," in *Roedunet International Conference (RoEduNet), 2013 11th*, Jan 2013, pp. 1–4.

[34] J. Yu, F. Jiang, and T. Zhu, "Rtic-c: A big data system for massive traffic information mining." in *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on, (pp. 395-402).*, (2013, Dec).

[35] S. Bansal, "Towards a semantic extract-transform-load (etl) framework for big data integration," in *Big Data (BigData Congress), 2014 IEEE International Congress on*, June 2014, pp. 522–529.

[36] ——, "Towards a semantic extract-transform-load (etl) framework for big data integration," in *Big Data (BigData Congress), 2014 IEEE International Congress on*, June 2014, pp. 522–529.

[37] X. Mo and H. Wang, "Asynchronous index strategy for high performance real-time big data stream storage." in *Network Infrastructure and Digital Content (IC-NIDC), 2012 3rd IEEE International Conference on, (pp. 232-236).*, 2012, Sept.

[38] S. Mariyah, "Identification of big data opportunities and challenges in statistics indonesia." in *ICT For Smart Society (ICISS), 2014 International Conference on, (pp. 32-36).*, 2014, Sept.

[39] X. Tao, F. Ge, T. Huaiyuan, Z. Hong, and L. Xinran, "Thump storage: A management and analysis system for structured big data. mechatronic sciences,," in *Electric Engineering and Computer (MEC), Proceedings 2013 International Conference on, (pp. 2424-2427).*, (2013, Dec).

[40] K. Grolinger, M. Hayes, W. Higashino, A. L'Heureux, D. Allison, and M. Capretz, "Challenges for mapreduce in big data," in *Services (SERVICES), 2014 IEEE World Congress on*, June 2014, pp. 182–189.

[41] A. Patel, M. Birla, and U. Nair, "Addressing big data problem using hadoop and map reduce," in *Engineering (NUiCONE), 2012 Nirma University International Conference on*, Dec 2012, pp. 1–5.

[42] R. Mao, H. Xu, W. Wu, J. Li, Y. Li, and M. Lu, "Overcoming the challenge of variety: big data abstraction, the next evolution of data management for aal communication systems." *Communications Magazine, IEEE, 53(1), 42-47.*, 2015, January.

[43] A. Patel and . N. U. Birla, M., "Addressing big data problem using hadoop and map reduce." in *Engineering (NUiCONE), 2012 Nirma University International Conference on, (pp. 1-5).*, 2012, Dec.

[44] I. Foster. (1995) Designing parallel algorithm. [Online]. Available: http://www.mcs.anl.gov/~itf/dbpp/text/node19.html#SECTION02350000000000000000

[45] J. Yu, F. Jiang, and T. Zhu, "Rtic-c: A big data system for massive traffic information mining," in *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, Dec 2013, pp. 395–402.

[46] M. K. William. (2015, August) Research methods knowledge base: Types of data. [Online]. Available: http://www.socialresearchmethods.net/kb/datatype.php

[47] R. Hu, X. Zhang, and P. Wang, "Classification and evaluation of online indexing strategies," in *Technologies and Applications of Artificial Intelligence (TAAI), 2011 International Conference on*, Nov 2011, pp. 233–238.

[48] A. Chacon, S. Marco-Sola, A. Espinosa, P. Ribeca, and J. Moure, "Boosting the fm-index on the gpu: effective techniques to mitigate random memory access," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[49] A. Gani, A. Siddiqa, S. Shamshirband, and F. Hanum, "A survey on indexing techniques for big data: taxonomy and performance evaluation," *Knowledge and Information Systems*, pp. 1–44, 2015.

[50] Y. Tang and L. Liu, "Multi-keyword privacy-preserving search in personal server networks," *Knowledge and Data Engineering, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[51] F. Amato, A. De Santo, F. Gargiulo, V. Moscato, F. Persia, A. Picariello, and S. Poccia, "Semtree: An index for supporting semantic retrieval of documents," in *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, April 2015, pp. 62–67.

[52] A. Matsui, S. Nishimura, and S. Katsura, "A classification method of motion database using hidden markov model," in *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, June 2014, pp. 2232–2237.

[53] Widodo and W. Wibowo, "Improving classification performance by extending documents terms," in *Data and Software Engineering (ICODSE), 2014 International Conference on*, Nov 2014, pp. 1–5.

[54] V. Alvarez, S. Richter, X. Chen, and J. Dittrich, "A comparison of adaptive radix trees and hash tables," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, April 2015, pp. 1227–1238.

[55] I. Jaluta, "Transaction management in b-tree-indexed database systems," in *Information Science, Electronics and Electrical Engineering (ISEEE), 2014 International Conference on*, vol. 3, April 2014, pp. 1968–1975.

[56] Y. Yu, Y. Zhu, W. Ng, and J. Samsudin, "An efficient multidimension metadata index and search system for cloud data," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, Dec 2014, pp. 499–504.

[57] S. Puri and S. K. Prasad, "A parallel algorithm for clipping polygons with improved bounds and a distributed overlay processing system using mpi," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, May 2015, pp. 576–585.

[58] A. Eldawy and M. Mokbel, "Spatialhadoop: A mapreduce framework for spatial data," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, April 2015, pp. 1352–1363.

[59] H. Xu, N. Yao, W. Hu, H. Pan, and X. Gao, "The design and implementation of image information retrieval," in *Computer Science Service System (CSSS), 2012 International Conference on*, Aug 2012, pp. 1547–1550.

[60] GiST. (2015, March) Introduction to gist. [Online]. Available: http://www.sai.msu.su/~megera/postgres/gist/doc/intro.shtml

[61] PostgreSQL. (2015, March) Gin indexes. [Online]. Available: http://www.postgresql.org/docs/9.4/static/gin-intro.html

[62] M. Rouse. (March, 2015) B-tree definition. [Online]. Available: http://searchsqlserver.techtarget.com/definition/B-tree

[63] N. Du, J. Zhan, M. Zhao, D. Xiao, and Y. Xie, "Spatio-temporal data index model of moving objects on fixed networks using hbase," in *Computational Intelligence Communication Technology (CICT), 2015 IEEE International Conference on*, Feb 2015, pp. 247–251.

[64] SQlite. (2015, March) The sqlite r*tree module. [Online]. Available: https://www.sqlite.org/rtree.html

[65] G. Bui Cong and A. Duong-Tuan, "Improving sort-tile-recusive algorithm for r-tree packing in indexing time series," in *Computing Communication Technologies - Research, Innovation, and Vision for the Future (RIVF), 2015 IEEE RIVF International Conference on*, Jan 2015, pp. 117–122.

[66] Teradata. (2013, March) Hash indexing. [Online]. Available: http://www.info.teradata.com/HTMLPubs/DB_TTU_14_00/index.html#page/Database_Management/B035_1093_111A/ch02.021.26.html

[67] Elastic. (2015, March) Inverted index. [Online]. Available: http://www.elastic.co/guide/en/elasticsearch/guide/master/inverted-index.html

[68] C. U. Press. (2015, March) Indexes. [Online]. Available: http://nlp.stanford.edu/IR-book/html/htmledition/an-example-information-retrieval-problem-1.html

[69] A. Kucharik. (2015,, March) What is ping? what is ping?

[70] I. Areni, S. Tsuzuki, and Y. Yamada, "Packet size optimization of pps based radiation detection for aee-plc," in *Power Line Communications and Its Applications (ISPLC), 2012 16th IEEE International Symposium on*, March 2012, pp. 47–51.

[71] D. N. Quang, O. H. See, D. V. Nga, L. L. Chee, C. Y. Xuen, and e. a. Karuppiah, S., "Customized ping tool for smart grid communication network testing." in *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on, (pp. 223-227).*, 2012, Nov.

[72] W.-C. Yang, J.-D. Jhan, D.-Y. Chen, K.-H. Lai, and R.-R. Lee, "Quality of service test mechanism and management of broadband access network." in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific, (pp. 1-4).*, 2014, Sept.

[73] P. Team. (2015,, March) Pinger ping end-to-end reporting. pinger ping end-to-end reporting.

[74] S. Team. (2015, March) Slac history. slac history.

[75] W. Jamal, A.and Pradani, N. Hasanati, A. Supriyanto, and R. Pujianto, "Scalability of dna sequence database on low-end cluster using hadoop. (pp. 50-55)." in *Information Technology Systems and Innovation (ICITSI), 2014 International Conference on,*, (2014, Nov).

[76] A. Babenko and V. Lempitsky, "The inverted multi-index," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 6, pp. 1247–1260, June 2015.