

**DEVELOPMENT OF AN ENHANCED DATA ANNOTATION AND
FILTERING TECHNIQUE FOR IOT SENSORS**

BY

EMMANUEL JERRY DAUDA

P16PSCS8070

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES

AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF

MASTER OF SCIENCE (M.Sc.) DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE,

AHMADU BELLO UNIVERSITY,

ZARIA

DECEMBER, 2019

DECLARATION

I declare that this dissertation titled “DEVELOPMENT OF AN ENHANCED DATA ANNOTATION AND FILTERING TECHNIQUE FOR IOT SENSORS” has been carried out by me under the supervision of Prof. A. A. Obinyi and Dr. A. F. D. Kana. This dissertation has not been presented for another degree at any university. The information contained in the literature has been duly acknowledged in the text and a list of references provided.

.....

.....

.....

Emmanuel Jerry Dauda

Signature

Date

DEDICATION

This dissertation is dedicated to the Almighty God for His divine direction, guidance and provision throughout the period of this dissertation.

ACKNOWLEDGEMENT

First of all, I would like to acknowledge the uncommon grace of God upon this work. At first, it was as though all hope was lost. But God intervened and everything was back on track again.

Also, I would like to acknowledge the immeasurable effort of my supervisors, Prof. A. A. Obiniyi and Dr. A. F. D. Kana. Their tireless contributions, support, advice and encouragement made this research a success.

My sincere appreciation also goes to my friends D. M. Chai, Musa Yusuf, O. M. Ayo and all the Department of Computer Science Postgraduate Students for their assistance, support and contribution towards the completion of this work.

Furthermore, my profound gratitude goes to The Head of Department (HOD), Department of Computer Science, Prof S. B. Junaidu for his relentless effort towards ensuring I and other students don't relent in pursuit of success. Similarly, the entire staff and management of ICICT ABU Zaria are highly appreciated for giving me a solid foundation in the world of computing.

Finally, my utmost gratitude to my entire family, immediate and extended. Starting with my ever-supportive wife, Mrs Opeyemi Emmanuel, my Dad, Pa. Kolawole Emmanuel (JP), my Mum, Mrs Juliana Emmanuel (JP), siblings and family friends for parental guidance, prayers, provisions and all-round support.

God bless you all.

ABSTRACT

Internet of Things (IoT) applications depend on data that are meaningful to the machine to efficiently function. Amongst other sources, data are generated by different types of sensors such as proximity sensor, pressure sensor, temperature sensor and ultrasonic sensor. The diversity of these sensors reflects on the data they generate. As a result, IoT applications encounter challenges understanding and processing these data. The most recent solution to this problem is data filtering and annotation on gateways. This has also resulted into a bottleneck processing thereby causing delay and inconsistencies in processed data. Consequently, an enhanced data preparation and annotation technique is proposed. This approach uses a distributed programming model for sensory data processing. The proposed approach seeks to develop a Hadoop MapReduce algorithm which efficiently filters and annotates sensory data in a distributed manner. To evaluate the feasibility of the proposed approach, data generated by sensors are stored on Hadoop Distributed File System (HDFS) and are processed by a MapReduce job. Semantic Web technologies such as Extensible Markup Language (XML) and Resource Description Framework (RDF) were used for the data annotation. Two categories of experiments were conducted and comparison between the proposed system and the existing system were done based on data size and processing time. This dissertation concludes that the proposed system has 61.65% processing time and 13.41% data size enhancement respectively over the existing system

TABLE OF CONTENTS

DECLARATION.....	ii
CERTIFICATION.....	iii
DEDICATION.....	iv
ACKNOWLEDGEMENT.....	v
ABSTRACT.....	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES	x
CHAPTER ONE	1
INTRODUCTION.....	1
1.1 Background to the Study.....	1
1.2 Motivation.....	3
1.3 Problem Statement.....	3
1.4 Aim and Objectives.....	5
1.5 Research Methodology	5
1.6 Organization of Dissertation	7
CHAPTER TWO	8
LITERATURE REVIEW	8
2.1 Introduction.....	8
2.2 Internet of Things (IoT).....	8
2.2.1 Characteristics of Internet of Things	9
2.2.2 Internet of Things Architecture	10
2.3 Semantic Web.....	14
2.3.1 Semantic Web Technologies	14
2.3.2 Role of XML and RDF	16
2.3.3 Domain Ontology	16
2.4 Apache Hadoop	17
2.4.1 MapReduce.....	18
2.4.2 Hadoop Distributed File System (HDFS)	20
2.5 Related Works.....	20
2.6 Literature Gap and Contribution of this Dissertation.....	27

2.7 Data Source.....	27
CHAPTER THREE.....	30
DESIGN OF AN ENHANCED DATA FILTERING AND ANNOTATION TECHNIQUE FOR IOT SENSORS	30
3.1Introduction.....	30
3.2 Data Preparation and Preprocessing Method	31
3.2.1 Data Extraction	32
3.2.2 Data Cleaning	33
3.2.3 Data Transformation	33
3.3 Distributed Data Storage and Parallel Data Processing.....	34
3.3.1 Distributed Data Storage on HDFS	34
3.3.2 Parallel Data Processing using MapReduce Model	37
3.4 System Architecture.....	40
CHAPTER FOUR.....	41
IMPLEMENTATION, RESULT AND ANALYSIS.....	41
4.1 Introduction.....	41
4.2 Block Information.....	42
4.3 Experimental Results for Data Processing on Existing and Proposed System.....	46
4.4 Comparative Analysis of Existing System and Proposed System.....	52
CHAPTER FIVE	56
SUMMARY, CONCLUSION AND FUTURE WORK	56
5.1 Summary.....	56
5.2 Conclusion	56
5.3 Future Work.....	57
REFERENCES.....	58
APPENDIX A	61
APPENDIX B	66
APPENDIX C.....	69

LIST OF FIGURES

Figure 2.1: IoT Architecture (Keyur and Sunil, 2016)	11
Figure 2.2: IoT Gateway Architecture (Al-Osta <i>et al.</i> , 2017).....	24
Figure 2.3: Data Preparation Module (Al-Osta <i>et al.</i> , 2017)	25
Figure 2.4: Data Annotation Module (Al-Osta <i>et al.</i> , 2017).....	26
Figure 2.5: Smart Green Infrastructure Monitoring Sensors sample data (Data.gov, 2018).....	29
Figure 2.6: Smart Green Infrastructure Monitoring Sensors sample data - CSV (Data.gov, 2018)	30
Figure 3.1: Data Preparation and Preprocessing Tasks	32
Figure 3.2: Distributed Data Storage Architecture	36
Figure 3.3: Distributed Storage of 1.42GB GSIMS Data	37
Figure 3.4: Parallel Data Processing Architecture.....	38
Figure 3.5: System Architecture	40
Figure 4.1: 1.42GBBlocks Information	41
Figure 4.2: Time taken for processing datasets on existing System.....	47
Figure 4.3: Processed file size on existing System.....	48
Figure 4.4: Time taken for processing datasets on Proposed System.....	49
Figure 4.5: File size generated by Proposed System	50
Figure 4.6: Processed and Invalid document size.....	51
Figure 4.7: Processing Time on Existing and Proposed System	53
Figure 4.8: File size generated on Existing and Proposed System	54

LIST OF TABLES

Table 3.1: Attributes Description	32
Table 3.2: Cleaned Attributes.....	33
Table 4.1: Block information for 1.42GB dataset storage	42
Table 4.2: Block information for 2.14GB dataset storage	43
Table 4.3: Block information for 3.02GB dataset storage	44
Table 4.4: Block information for 5.07GB dataset storage	45
Table 4.5: Time taken for processing datasets on existing System	47
Table 4.6: Processed file size on existing System	48
Table 4.7: Time taken for processing datasets on Proposed System	49
Table 4.8: Processed File Size on the Proposed System.....	50
Table 4.9: Invalid sensor data size	51
Table 4.10: Time taken for processing datasets on Existing and Proposed System	52
Table 4.11: Processed File Size on Existing and Proposed System.....	54

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

Internet of Things (IoT) has been visualized to redefine Information and Communication Technology (ICT). This restructuring is popularized by the never before seen exponential increase of Internet connected devices in our modern world. Recent analysis by Cisco shows that by 2030, 500 billion devices are expected to be connected to the Internet (Cisco, 2016). These devices have sensors that retrieve data, interact with the environment, and send information over a network. Connection of all these devices is referred to as Internet of Things (IoT). Data generated by these intelligent devices are used to aggregate, analyze and deliver insight which helps drive more informed decisions and actions.

The data generated are often sent to the cloud for storage and processing (Borgia, 2014). The processed data can be used for predictive analysis to predict possible occurrences in the real world and could also be beneficial to applications.

As revealed by Google, five (5) exabytes of data were generated from the dawn of civilization till 2013. Now the world generates five (5) exabytes of data every two days (Data-Flair, 2018). These huge amounts of data generated are transferred through a medium to the Internet cloud platforms. These platforms have the processing abilities to further process and analyze the collected data to ensure their usefulness for IoT applications that can potentially enhance several areas of human daily life such as weather forecasting, traffic monitoring, and health care (Khan *et al.*, 2015).

Researchers have proposed and built several systems to eliminate the transfer of redundant data and to also filter data for storage in the cloud. Recently, IoT Gateways were introduced to minimize the huge amount of unwanted data sent to the cloud. The IoT Gateways serve as a point where data generated by IoT devices and sensors are collected and gathered. Also, data filtering and annotation occur using the semantic web-based approach before being sent to the cloud for further usage by end-user applications. A recent approach of employing IoT gateway devices as a hub for data processing has been widely adopted (Al-Osta *et al.*, 2017). This is to reduce the task of transferring data to the cloud through IoT networks, and to minimize processing cost at the cloud level.

Al-Osta *et al.* (2017) described the gateway as composed of three main modules. These modules are: The Data Preparation Module, the Data Annotation Module and the Cloud Interface.

Data preparation consists of three modules, which are data aggregation, data filtering and data structuring (Al-Osta *et al.*, 2017). The data aggregation module receives data from sensor nodes and stores them in a temporary file for analyzing. The data filtering module reduces the amount of data to be processed at the annotation stage, thereby saving resources needed. This module also includes a rule engine which contains a set of predefined rules for removing duplicated and unnecessary data. The data structuring module receives filtered sensor data from the filtering sub module, and utilizes a technique to convert the data into an Extensible Markup Language (XML) file for data annotation. Data annotation converts the resulting data in XML format to Resource Description Framework (RDF) format which is sent to the cloud for storage.

With the aid of Semantic Web Technologies (SWTs), data and other information on the web and their relationships are described as resources. Because of this, resources on the web are easily interpreted, understood and integrated by machines. Recently, SWTs have been extended to the

IoT domain to promote interoperability and to enhance the quality of data(Barnaghi *et al.*, 2012). This is achieved by modeling IoT data based on common terminologies that can be interpreted by different software agents. This process is referred to as semantic annotation, which implies the involvement of several SWTs such as Web Ontology Language (OWL), Resource Description Framework Schema (RDFS), and RDF to build conceptual models (i.e. Ontology) to describe application domain concepts and the relationships that exist between them (Aggarwal *et al.*, 2013).

1.2 Motivation

IoT application developers depend on data stored on the cloud which are interpretable by machines so as to develop interoperable, effective and efficient applications using a lower computing resource. Also, IoT applications need to be able to synchronize regardless of the source where their data was generated. In order to ensure data stored on the cloud are consumable by machine and to ensure the burden of data preparation and annotation at the gateway is alleviated, data processing is needed at the storage level. This research would also advance the knowledge of using Hadoop framework for distributed data storage on HDFS, the Hadoop storage layer and parallel processing on MapReduce, the computation layer.

1.3 Problem Statement

IoT system components are of different varieties. This heterogeneity characteristic of IoT systems is reflected on the data they produce, which in turn affects the task of IoT application to interpret data and effectively utilize them. This also makes data integration difficult, which leads to the lack of interoperability among different IoT systems. Thus, limiting the development of applications that can benefit from data generated from diverse domains. In addition to the heterogeneity aspect of IoT data, these data are continuously streaming. Consequently, huge

amounts of data are regularly generated and sent to gateway platforms for further processing and filtering. According to Zachariah *et al.* (2015), IoT gateway problem exists in part because today's gateways fuse network connectivity, in-network processing, and user interface functions, which limit their capabilities to manipulate such amount of data being generated from different sources and at high velocity. Also, the process consumes a considerable amount of resources, where in some cases bottleneck-like processing at the gateway occurs due to the increasing amount of data which results to high latency and low throughput.

Data filtering at either the sensor node or the gateway node inadvertently discards sensitive data which could be very useful for IoT applications. In addition, preprocessing data at the gateway overloads the route between the sensors and gateway, leaving the route between the cloud and IoT gateways idle. These have led to network traffic overloading and latency issues that might influence time-sensitive services. On a similar note, as proposed by Al-Osta *et al.*(2017) the filtering mechanism and annotation algorithms applied at the gateway level based on a rule engine could result into a delayed data-transfer from the Gateway. Applying semantic annotation algorithms on large amount of data at the gateway device level will result in extensively consuming its resources because of the technologies involved.

Finally, despite data filtering, another problem with data processing at the gateway is, sometimes the gateway ends up sending data that are not complete to the cloud for storage. For instance, if a sensor is expected to send its *measuredValue* and *timestamp* and only the *measuredValue* is available while the *timestamp* is empty, the gateway still sends the empty data for annotation. Therefore, this work seeks to address these challenges, by enhancing and performing the annotation at the storage layer.

1.4 Aim and Objectives

The aim of this work is to develop an enhanced data annotation and filtering technique for data generated by IoT Sensors.

The specific objectives of the research are to:

- a. Design an enhanced data preparation and annotation technique.
- b. Simulate the implementation of the technique using Apache Hadoop.
- c. Evaluate the implementation in objective (b) based on Data Size and Processing Time and compare with Al-Ostaet *al.* (2017).

1.5 Research Methodology

Designing an enhanced data preparation and annotation technique involves the following steps:

- i. Download Smart Infrastructure Monitoring Sensor dataset from data.gov
- ii. Preprocess the dataset such that unwanted attributes are removed
- iii. Design a distributed data storage architecture
- iv. Design a parallel data processing architecture
- v. Develop a MapReduce algorithm for data structuring and annotation
- vi. Design the proposed system architecture.

Similarly, to simulate the implementation of the algorithm and distributed data storage, these steps can be followed:

- i. Install the following software components serially:
 - a. Ubuntu 16.04 Long Term Support (LTS) Operating System
 - b. Java Development Kit
 - c. Eclipse Oxygen Integrated Development Environment

- d. Apache Hadoop: MapReduce and HDFS Framework
- ii. Store preprocessed datasets on HDFS
- iii. Implement the MapReduce algorithm on Hadoop framework
- iv. Conduct four (4) different experiments using preprocessed datasets.
- v. Store resulting data of each experiment on HDFS for use by IoT application.

Finally, to evaluate the proposed system and compare it with the existing system, the following steps are followed:

- i. Retrieve the time taken for executing dataset of respective experiments
- ii. Retrieve the size of processed datasets in each experiment.
- iii. Compare processing time and data size of proposed system with Al-Ostaet *al.*, (2017).

1.6 Contribution of this Dissertation

The following contributions are made in this research:

- i. This work develops a MapReduce algorithm for data annotation on a distributed cluster of commodity hardware which reduces data processing time.
- ii. An enhanced rule engine is also developed in this system. This efficiently filter records before they are processed.
- iii. The proposed approach keeps a log of incomplete data instead of discarding them. These data could be used for further decision making, thus enhancing the overall data annotation proposed by Al-Osta *et al.* (2017).

1.7 Organization of Dissertation

The rest of the work is organized as follows:

Chapter Two: In this chapter, several literatures were reviewed which includes the introduction to Internet of Things (IoT), Characteristics of IoT and the general structure of IoT architecture. This chapter also has a review of Semantic web which includes Semantic web technologies such as XML, RDF, RDFS and OWL. Roles of XML and RDF and the need for Domain Ontology were also explained. Furthermore, this chapter also includes a review of Apache Hadoop, MapReduce and Hadoop Distributed File System (HDFS). The chapter ends with summary of other related works and their limitations. Detail review of the base research of this work and its limitations are enumerated. Finally, the source and collection of Data are discussed.

Chapter Three: The chapter discussed research methodology which starts with Data Preparation and Preprocessing (Extraction, Cleaning and Transformation of Data), Distributed Data Storage and Parallel Data Processing and System Architecture.

Chapter Four: In this chapter the experimental results and analysis were presented; it comprises result of the distributed storage of dataset before being processed. The chapter also contain result of dataset processing with proposed system and with existing system, comparing the results based on file size and processing time. Finally, the distributed storage of the processed data is also discussed.

Chapter Five: The summary, conclusion and future work of this research are presented in this chapter. Contribution to knowledge and limitation of the work are also stated.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

Data preparation and annotation addresses the complexity of data integration by high-level IoT applications and it also eliminates lack of interoperability between IoT applications. Removing unnecessary data enhances network traffic between sensor nodes and gateway nodes. Storing ready-made data on the cloud reduces the computing resources needed for its processing and analyzing and its machine understandable nature empowers software developers to create more applications that will make possible impact on human daily activity.

2.2 Internet of Things (IoT)

The Internet of Things (IoT) refers to the use of smartly connected devices and systems in machines and other physical objects to leverage data collected by embedded sensors and actuators. IoT is expected to spread rapidly over the coming years and this convergence will trigger a new dimension of services that will improve the quality of life of consumers and the productivity of businesses (GSMA, 2014). According to Ovidiu and Peter (2014), the Internet of Things refers to the overall concept of things, most specifically everyday objects that can be found, contacted, identified and interpreted via a data sensing system and/or controllable via the Internet, either wireless LAN, wide area networks or other means. Everyday objects include not only the electronic devices we are exposed to and higher-technology products such as vehicles and equipment, but also things we don't usually think of as technological at all such as meat, clothes, table, pet, tree and water. The Internet of Things (IoT) is a global information

technology network that facilitates digital technologies by interconnecting (physical and virtual) things based on current and emerging interoperable ICT (Keyur and Sunil, 2016).

2.2.1 Characteristics of Internet of Things

The fundamental characteristics of the IoT as stated by Ovidiu and Peter (2014) are as follows:

- a. **Interconnectivity:** With regard to the IoT, anything can be interconnected with the global information and communication infrastructure.
- b. **Things-related services:** The IoT is capable of providing thing-related services within the constraints of things, such as privacy protection and semantic consistency between physical things and their associated virtual things. In order to provide thing-related services within the constraints of things, both the technologies in physical world and information world will change.
- c. **Heterogeneity:** The devices in the IoT are heterogeneous since they are based on different hardware platforms and networks. They can interact with other devices or service platforms through different networks.
- d. **Dynamic changes:** The state of devices change dynamically, e.g., sleeping and waking up, connected and/or disconnected as well as the context of devices including location and speed. Moreover, the number of devices can change dynamically.
- e. **Enormous scale:** The number of devices that need to be managed and that communicate with each other will be at least an order of magnitude larger than the devices connected to the current Internet. Even more critical will be the management of the data generated and their interpretation for application purposes. This relates to semantics of data, as well as efficient data handling.

- f. Safety: This includes the safety of personal data and the safety of physical well-being. Securing the endpoints, the networks, and the data moving across all of it means creating a security paradigm that will scale.
- g. Connectivity: Connectivity enables network accessibility and compatibility. Accessibility is getting on a network while compatibility provides the common ability to consume and produce data.

2.2.2 Internet of Things Architecture

IoT architecture consists of different layers of technologies supporting IoT. It serves to illustrate how various technologies relate to each other and to communicate the scalability, modularity and configuration of IoT deployments in different scenarios.

The functionality of each layer in Keyur and Sunil(2016) are as explained in the following sections:

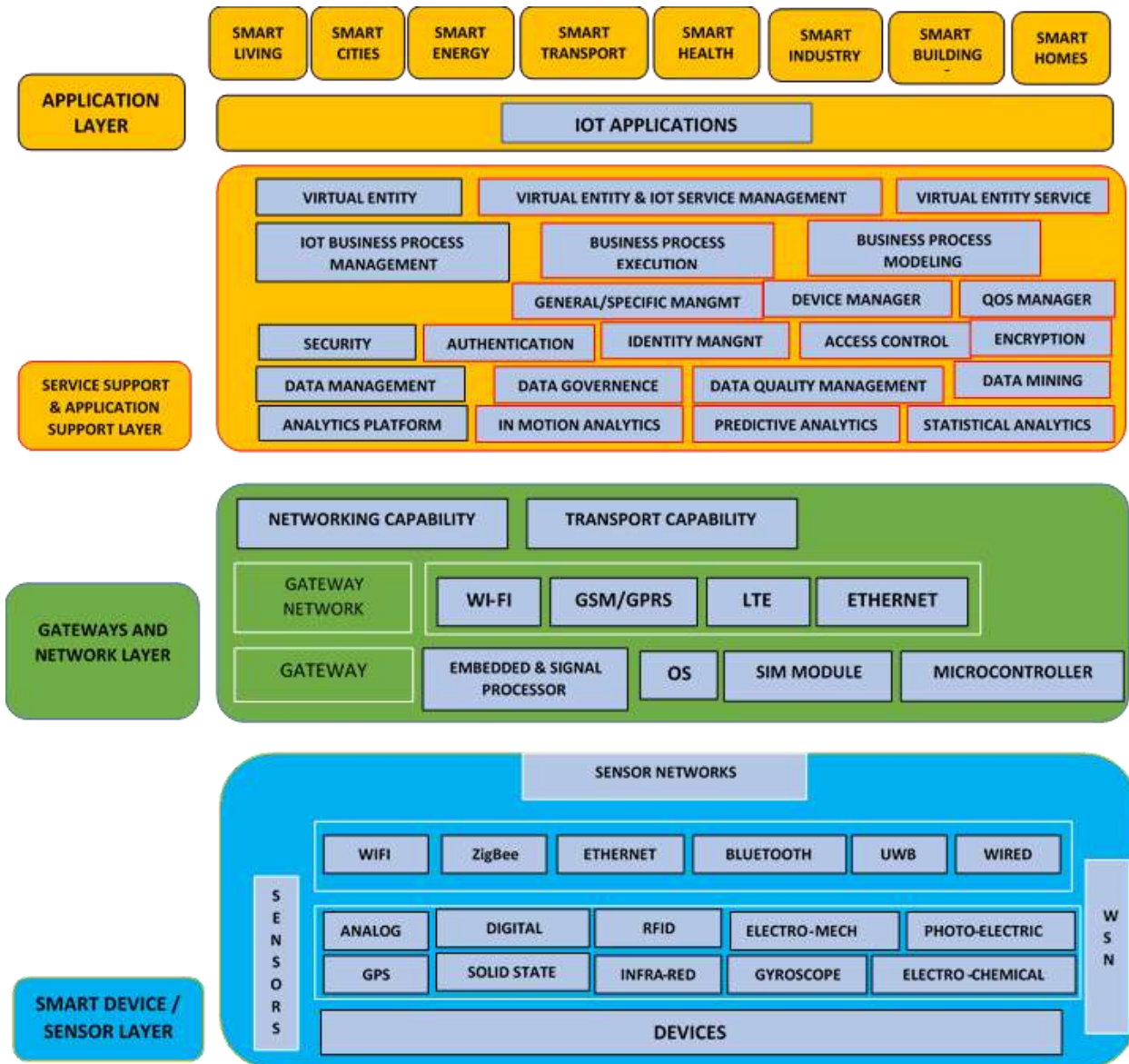


Figure 2.1: IoT Architecture (Keyur and Sunil, 2016)

a. Smart device / Sensor layer:

As shown in Figure 2.1, the lowest layer consists of smart objects embedded with sensors. The sensors make it possible to interconnect the physical and digital realms and capture and process information in real time. For different purposes, there are different types of sensors. The sensors are capable of measuring temperature, air quality, velocity, humidity, pressure, flow, movement, and electricity. They may also have a degree of memory in some cases, which allows them to record a number of measurements. A sensor can calculate and translate the physical property into a signal that an instrument can recognize. Sensors are classified according to their specific function, such as climate sensors, body sensors, sensors for home appliances and telematics sensors for cars. Most sensors require access to sensor gateways. This may take the form of a Local Area Network (LAN) such as Ethernet or Wi-Fi or Personal Area Network (PAN) such as ZigBee, Bluetooth or Ultra-Wideband (UWB). For sensors that do not need access to sensor aggregators, Wide Area Network (WAN) can provide their connectivity to backend servers and applications. Sensors that use low power and low connectivity data rate typically form wireless sensor networks (WSNs). WSNs are gaining popularity as they can accommodate much more sensor nodes while maintaining a sufficient battery life and covering wide areas.

b. Gateways and Networks

These tiny sensors produce a large volume of data, which requires a reliable and high-performance wired and wireless network infrastructure as a means of transport. Present networks were used to serve machine-to-machine (M2M) networks and their applications, often connected to very different protocols. With the demand required to support a broader range of IoT infrastructure and devices such as high-speed transactional services, context-

aware applications, numerous networks with different technologies and communication protocols are needed to work in a large and diverse environment with one another. Such networks may be in the form of private, public and hybrid systems and are designed to support latency, bandwidth and security connectivity requirements.

c. Management Service Layer

The management service makes it possible to process information through monitoring, security controls, application analysis and device management. The business and system rule engines are one of the important features of the management service layer. IoT puts together entities and devices that communicate and collaborate to provide information in the form of incidents and situational data such as product temperature, current location or traffic data. Some of these incidents require sorting and routing to post-processing systems such as collecting regular sensory data, while others require response to urgent circumstances such as responding to crises on the health conditions of patients. The rule engines support decision logic formulation and activate interactive and computerized processes to allow a much more functional IoT system.

d. Application Layer

The IoT application encompasses “smart” environments/spaces in areas such as: Transportation, Building, City, Lifestyle, Retail, Agriculture, Factory, Supply chain, Emergency, Healthcare, User interaction, Culture and tourism, Environment and Energy.

However, as described in Figure 2.1, the proposed technique in this research would be on the Service Support and Application Support Layer.

2.3 Semantic Web

The web was designed for individuals and organizations irrespective of their geographical locations to access information and transmit data from one location to another. In other words, humans have the ability to understand and control web contents unlike the computer machine. Due to this divergence the semantic web technology was invented to assist develop a web that can be processed by humans and computer systems.

Semantic web is an extension of the current web as it's composed of enhanced features that enable users from different works of life, irrespective of their geographical locations to achieve timely results on the web. The key idea of the Semantic Web is to always technically associate or link data with a meaningful context (Arnulf, 2012). In most cases this data is often called Meta data whereby it is use as a link to access a web page that would describe in detail its meaning. For instance, the word "Science" can be said to be a Meta data that links to web page containing the general information about science (Edje, 2016).

According to Laura and Pamela(2013), Semantic web is a set of data that computer software can process automatically rather than a collection of documents for people to consume. The Semantic Web is motivated by the initiative of the present Web which has been in the background since its inception

2.3.1 Semantic Web Technologies

Semantic Web technology is defined as a method of linking data between systems or entities that allows for rich, self-describing interrelations of data available across the globe on the web (Edje, 2016). This means that the web is a combination of the existing hypertext markup language (HTML) contents and contents from computer generated programming software. Due to

Semantic Web's capabilities in annotating data, several semantic web technologies have been recently adopted to promote data integration and interoperability in the IoT field.

- a. Resource Description Framework (RDF) is a standard language for representing information about Web resources as XML format. It provides a unified framework for exchanging information between applications without loss of meaning. Data in RDF are stored in the form of triples, each triple is consisted of subject, property and object. All of the elements of the triple are resources with the exception of the last element, object that can be also a literal. Literal in the RDF sense is a constant string value such as string or number. Literals can be either plain literals without type or typed literals typed using XML Data types (Deepa, 2016). In typical IoT application consist of set of devices generating set of data, devices are semantically represented by the subject; while the property represents the measured quantity, and the object represents the measured value.
- b. Resource Description Framework Schema (RDFS) is an extension of RDF vocabulary, which enables more detailed description taxonomies of classes and properties. In another word, RDFs can be perceived as an expressive meta-model used to describe the vocabulary used in an RDF document.
- c. Web Ontology Language (OWL) represents more expressive way to model data on the semantic web. It was essentially developed to overcome some RDF and RDFs limitations such as the lack of a clear way for domain or range constraints description, and the lack of the ability of representing closure, inverse or transitive properties (Aggarwal, 2013).

RDF, RDFs, and OWL can be perceived as meta-meta models that encompass set of vocabularies used to define new domain specific schemas and ontologies.

2.3.2 Role of XML and RDF

XML and RDF are the current standards for establishing semantic interoperability on the Web, but XML addresses only document structure. RDF better facilitates interoperation because it provides a data model that can be extended to address sophisticated ontology representation techniques.

2.3.3 Domain Ontology

Ontologies as defined by Dillon *et al.*(2012) are formal, explicit specifications of a shared semantic conceptualization that are machine-understandable. Ontologies promote sharing of unified understanding of domain-specific structured information among software agents. Thus, enabling systems to consume data based on predefined concepts and relations in the ontology.

To facilitate interoperability and data exchange between IoT resources, recent activities to design ontologies to be used for several purposes including the description of sensor and sensor networks, IoT resources and services, smart Things. SSN semantic sensor network ontology is the most adopted by IoT projects. It was developed by W3C to describe three major concepts: systems, processes, and observations. The ontology can describe sensors, their accuracy and capabilities, observations and methods used for sensing.

Ontologies are considered one of the supports of the Semantic Web. Ontologies provide Semantic Web agents with background knowledge about domain concepts and their relationships. Ontologies can also be instantiated to create individuals that describe Semantic Web resources or real-world entities. For example, individuals of an ontology for Real estate agents could represent specific site destinations or activities. In such a scenario, a Semantic Web repository would provide instance data about these individuals, and agents can use their

ontological knowledge useful for applications in which knowledge plays a key role, but they can also trigger a major change in current Web contents (Antoniou, 2004).

2.4 Apache Hadoop

Hadoop is an open source project from the Apache Software Foundation, it provides a framework for distribution and parallel execution of applications and programs on cluster of servers (Data-Flair, 2018). Hadoop was inspired by Google's MapReduce programming model as well as the model's file system, the Google File System.

Apart from Hadoop, there are several MapReduce-like implementations for distributed systems such as Dryad from Microsoft, HPCC from LexisNexis and Disco from Nokia (Isardet *al.*, 2007). However, Hadoop is the most well-known and popular open source implementation of MapReduce. Hadoop is written in Java. Hadoop can be setup on single machine, that is a single node cluster but the real strength of Hadoop comes with a cluster of machines which is vertically and horizontally scalable.

Hadoop is made up of two major components which include the Hadoop Distributed File System (HDFS) and the Map-Reduce. The duo forms the storage layer and the processing or computation layer respectively.

Hadoop uses master/slave architecture and obeys the same overall procedure, for executing programs (Saeed and Saeed, 2014), which mean the master nodes oversee the affairs of the slave nodes. The master node stores metadata of data stored on the slave nodes and also keeps information about the health of the slaves. By default, Hadoop stores input and output files on its distributed file system, HDFS while its computation is done using the MapReduce programming model. Its worth-knowing that both parallel computation and distributed storage of data are carried out on the slave nodes with the approval of the master node.

2.4.1 MapReduce

MapReduce is a programming model that enables the processing of big data in parallel on a set of commodity hardware (Saeed and Saeed, 2014). Before the MapReduce model was introduced, large scale data processing was very difficult as the process required management of hundreds or thousands of processors. Parallelization and distribution of data alongside Input/Output scheduling were major challenges of processing large dataset. Apparently, this led to lack of fault tolerance in parallel computing (Data-Flair, 2018). Hence, the need for a model that provides the aforementioned features.

The MapReduce programming model was invented by Google and can be implemented in multiple programming languages such as Java, C, C++, Ruby, Groovy, Perl and Python. The model divides work into small parts, each of which can be done in parallel on group of computers called servers. MapReduce is highly scalable and can be used across many computers. Many small machines can be used to process jobs that normally could not be processed by a large machine (Data-Flair, 2018).

MapReduce defines computation as two functions: map and reduce. The input is a set of Key/Value Pairs (KVPs), and the output is a list of KVPs. The map function takes an input pair

and returns a set of intermediate KVPs called Intermediate Outputs (IOs). The reduce function takes an intermediate key and intermediate values associated to that key as its input, and returns a set of final KVPs as the output (Saeed and Saeed, 2014). Execution of a MapReduce program involves two phases. In the first phase called the Mapper Phase (MP) each input pair is given to a map function and a set of output pairs is produced. Afterwards in the second phase called the Reducer Phase (RP), all of the intermediate values that have the same key are aggregated into a list, and each intermediate key and its associated intermediate value list is given to a reduce function. The execution of a MapReduce program obeys the same two-phase procedure. Usually, distributed MapReduce is implemented using master/slave architecture (Dean and Ghemawat, 2010). In some cases, a job might require only the MP without the RP, such jobs are called Map Only Jobs. This is because in-between map and reduces phases there is key, sort and shuffle phase. According to Data-Flair (2018), Sort and Shuffle takes care of ordering the keys in ascending order as well as grouping values according to their keys. This process is very costly and should be skipped if Reduce Phase is not necessary, because avoiding Reduce Phase would also eliminate the phase of Sort and Shuffle. This also avoids network congestion as in shuffling.

In a MapReduce job, the output of mapper is written to local disk before sending to reducer but in Map Only job, this output is directly written to HDFS. This further save time and reduces cost as well (Data-Flair, 2018).

The master machine is responsible for the assignment of tasks and controlling the slave machines. The input is stored over a shared storage like distributed file system, and is split into chunks. First, a copy of map and reduce functions' code is sent to all workers, that is the slave nodes. Then, master assigns map and reduce tasks to workers. Each worker assigned a map task, reads the corresponding input split and passes all of its pairs to map function and writes the

results of the map function into intermediate files. After the map phase is completed, the reducer nodes read intermediate files and pass the intermediate pairs to reduce function and finally the pairs resulted by reduce tasks are written to final output files.

2.4.1.1 Map Reduce Terminologies

- a. Job: a job is a complete program which needs the execution of a Mapper and/or Reducer. A job consists of the input data, MapReduce program and configuration information.
- b. Task: This is an execution of a Mapper or a Reducer on a slice of data. Master node divides job into task and schedules it on the slaves.

2.4.2 Hadoop Distributed File System (HDFS)

HDFS is a file system designed for storing very large files running on cluster of non-expensive, low-end hardware used for daily purpose. It is designed on the principle of storage of a smaller number of large files. However, it provides fault tolerant storage layer for Hadoop and its other components (Data-Flair, 2018). HDFS stores data reliably even in case of hardware failure and it provides high-throughput access to application data.

HDFS has two types of nodes which works in master-slave fashion, which include the HDFS master node and the HDFS slave node.

2.5 Related Works

Gopinath and Sagayaraj (2011) proposed an approach to extract the methods of a project and store the metadata about the methods in the OWL. OWL stores the structure of the methods in it. Then the code will be stored in the distributed environment so that the software company located in various geographical areas can access. To reuse the code, a tool can be created that can extract the metadata such as function, definition, type, arguments, brief description, author, and so on

from the source code and store them in OWL. This source code can be stored in the HDFS repository. For a new project, the development can search for components in the OWL and retrieve them at ease.

However, this work focuses on extracting information about metadata only. Processing the observed or measured data is not performed, this limits the proposed approach.

Nasullahet *al.*(2011) presented a MapReduce based distributed Support Vector Machine (SVM) algorithm for automatic image annotation. MapReduce Sequent Minimal Optimization (MRSMO) is an SVM algorithm for automatic image annotation, using Google's MapReduce, a distributed computing framework that facilitates data intensive processing. MRSMO is built on the Sequent Minimal Optimization (SMO) algorithm and implemented using the Hadoop implementation of MapReduce. The framework facilitates a number of important functions such as partitioning the input data, scheduling the program's execution across a cluster of participating nodes, handling node failures, and managing the required network communications. In the approach Nasullahet *al.* (2011) partition the training data according to the dataset size as well as the number of MapReduce mappers to be employed.

The approach lacks efficient load balancing technique for optimal resource utilization as multiclass classification uses the one to many techniques instead of the one to one technique.

Christophe(2012)proposed a distributed framework composed of geographically distributed nodes managing a pool of semantically described IoT resources to enable scalable search and management for the IoT resources. The framework relies on the location as key parameter when searching the IoT resources, where nodes publish their location, and then a managing node based on neighboring nodes creates federations. By doing this, the author aims at managing data

produced by nodes locally. Where each node can make reasoning over received data and the produced knowledge stored locally.

The implementation of this approach in resource constraint environment will be difficult as it requires a lot of resources for local processing and storage (Al-Ostaet *et al.*, 2017).

Gyrard (2013) developed a sensor measurement ontology (SenMESO) for data annotation. SenMESO ontology is a combination of various domain ontologies covering the sensor data and features of interest. The sensors send the measured sensor values in SenML format to the gateways. The gateway nodes use a mechanism to process the sensed data to an XML file and send it to the aggregation gateways which use the stored ontologies to annotate the sensor data and thereby allow different applications to use it.

Sensor Web Enablement (SWE) are difficult to setup and definitely not suitable for resource-constrained environments (Khan *et al.*, 2015).

Kamburugamuveet *et al.* (2014) developed a distributed data processing platform called IoTCloud. The platform helps to connect devices to frameworks deployed in the cloud. The platform also allows for deployment of custom frameworks for distributed data processing without concern for how the data is processed. To demonstrate the effectiveness of the system, a robot navigation framework and application were developed. However, the system lacks the implementation of SWT for resource description.

Khan *et al.* (2015) proposed data annotation architecture for semantic applications in virtualized Wireless Sensor Network (WSN) environments. A base ontology was developed by extending the SSN ontology. Furthermore, a domain ontology was developed for a fire monitoring semantic application that was prototyped. The fire monitoring semantic application receives

annotated data and uses the fire domain ontology, along with a reasoner, to infer knowledge. An end-user can query over the annotated data to get the real-time information of the fire event, such as its status and location. The application is developed and deployed in the cloud using Google App Engine (GAE) and works in a heterogeneous virtualized WSN environment.

These steps are performed concurrently during the implementation of the application, which is definitely not suitable for resource-constrained environments and likely to increase network traffic especially in a wireless network.

Jutamard and Wiwat (2016) proposed a supporting tool to perform large RDF map data transfer and query. The system was developed using the Hadoop Framework to reduce access time and response time to queries. In Hadoop Distributed File System (HDFS) data nodes, the RDF / XML and related data is translated into a huge set of N-triples and submitted to Hadoop space. The RDF graph query in terms of SPARQL is evaluated and translated to a particular N-triple format to use Jena Algebra to find the answer. The MapReduce algorithm was designed to manipulate the RDF map in a specific manner.

Chen et al. (2019) proposed a model for learning semantic annotation of tabular data. The research proposed a deep prediction model that could fully exploit the contextual semantics of a table, including table position features learned from a Hybrid Neural Network (HNN), and intercolumn semantics features learned from a knowledge base (KB) search and query response. This work assumes that a table comprises of cells structured by columns and rows, without metadata such as names of columns or row identities. The input is a table whose type is to be predicted with a target column. The model showed good results on individual table sets, as well as when it was moved from one table to another.

Al-Ostaet *al.*(2017) stated that IoT system components are majorly classified into three main elements, which involve the Sensor node, Gateway and Cloud Platforms.

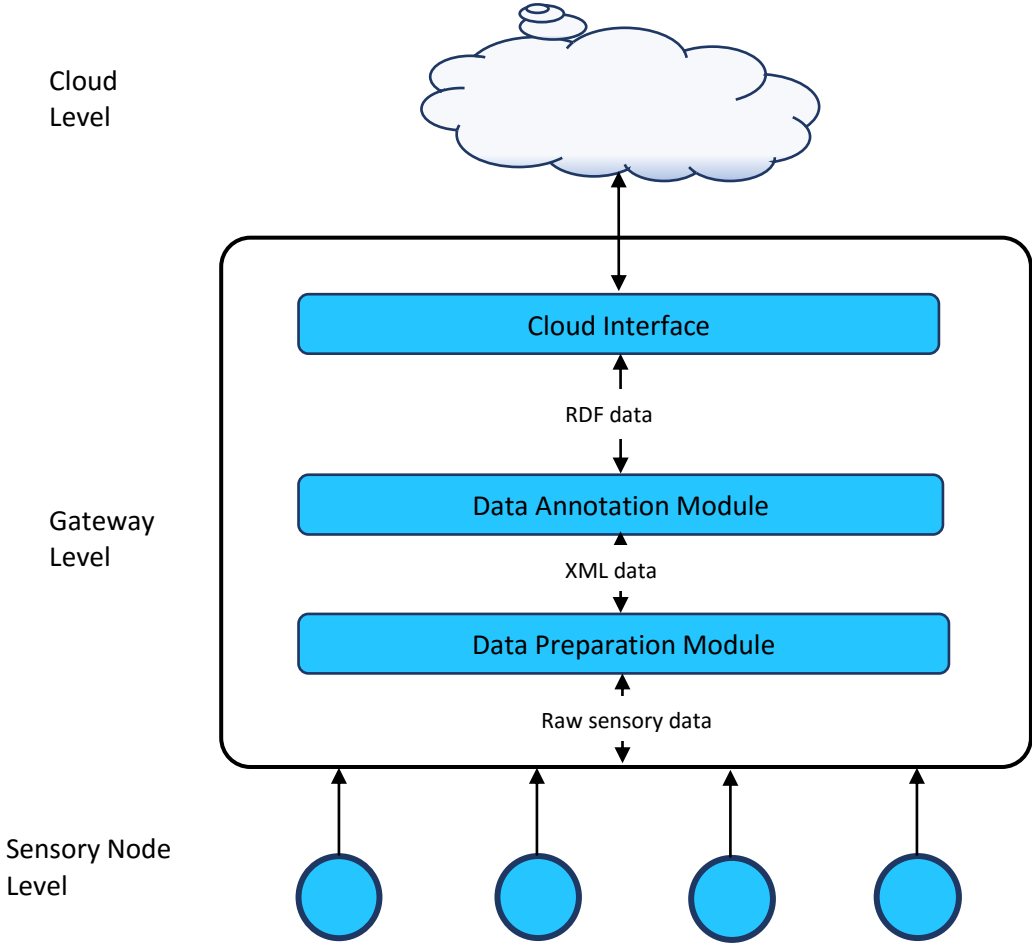


Figure 2.2: IoT Gateway Architecture (Al-Ostaet *al.*, 2017)

As depicted in Figure 2.2 sensornodes being the lowest level is made up of limited resources such as the sensors and the microcontrollers which collect data from surrounding environment and send them to the Gateway. Unlike the Sensor node, devices at the Gateway use more

computing resources at the node level because the Gateway interfaces with both the sensor node and the cloud. The Gateway also serves as the collection point for sensor readings.

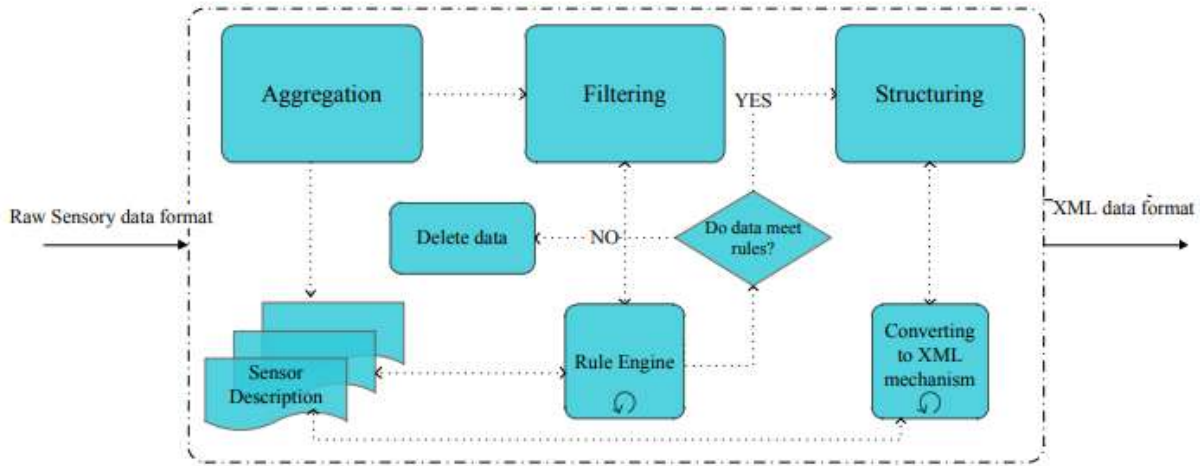


Figure 2.3: Data Preparation Module (Al-Ostaet *al.*, 2017)

Al-Ostaet *al.*(2017) proposed a gateway consisting of three (3) modules as shown in Figure 2.2. They are Data preparation, Data annotation, and Cloud interface. The data preparation module analyzes and formulates raw data sent by sensor nodes by filtering out duplicate and redundant data and converting the remaining data to XML format. The module is composed of three sub-modules; the data aggregation sub-module upon receiving raw sensory data from sensor nodes, stores them in a file temporarily for analysis. When a sensor is being read for the first time, its SensorID, type and model are saved in a file called the sensor description file. In subsequent readings, only the sensor ID, observation value and time stamp are recorded.

The data filtering sub-module on the other hand as depicted in Figure 2.3 uses a rule engine to reduce the amount of data to be annotated. This rule engine is composed of a predefined rule which filters the aggregated data. Two categories of rules were considered, amongst which are the sensor type-based rule and the interval time-based rule. The sensor-type based rule uses the

SensorID to access the sensor description file in order to extract the sensor type which determines the rules that will be used to perform the sensor’s filtering process. Interval time-based rule checks at interval set by the user based on the sensor type prevents the annotation of data outside a threshold. The data structuring sub-module by means of a conversion mechanism generates an XML file which contains the filtered data packets.

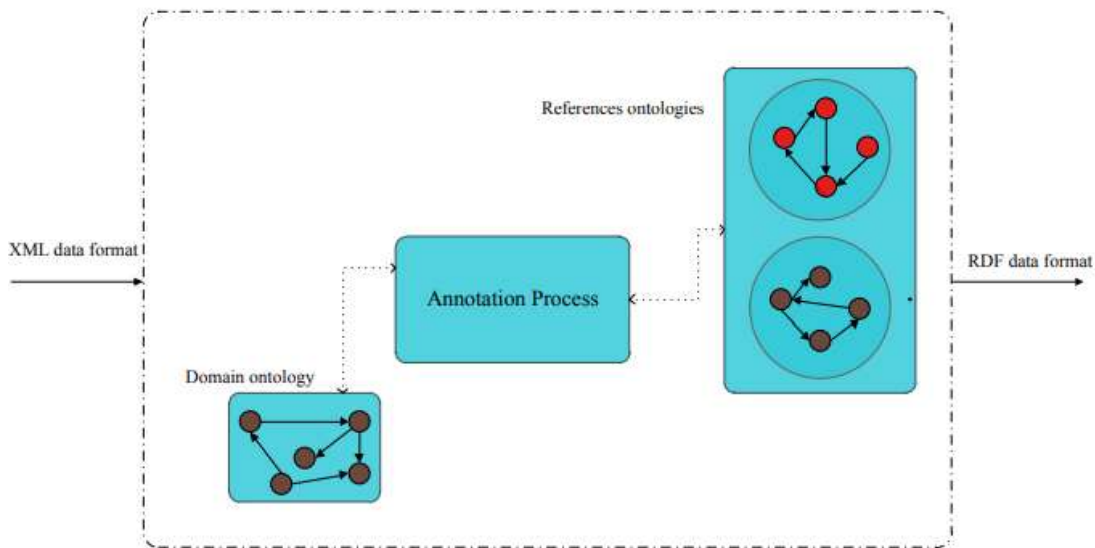


Figure 2.4: Data Annotation Module (Al-Ostaet *al.*, 2017)

The data annotation module used a Semantic Sensor Network (SSN) ontology and Description Ontology for Linguistic and Cognitive Engineering (DOLCE) Ultralite (DUL) ontology together with a Sensor domain ontology (SDO) developed by Al-Ostaet *al.* (2017) to convert the data in XML format obtained from the Data Preparation module to RDF format as illustrated in Figure 2.3. The RDF file is further transmitted to the Cloud Interface module for storage in the cloud. The system is posed with the following limitations:

- a. Devices at the gateway use more computing resources at the node level because the gateway interfaces with both the sensor node and the cloud, hence this approach cannot be implemented in a resource constraint environment.
- b. This approach would lead to high latency due to bottleneck processing at the gateway.
- c. The approach could lead to the removal of data that are seen as duplicate or redundant.
- d. The system does not make provision for logging incomplete data that are sent to gateway, perhaps by a faulty sensor.

2.6 Literature Gap

The reviewed systems, frameworks and approaches developed by researchers have quite a number of limitations, such as concentrating on the interoperability between devices rather than data. Another key issue in a resource constraint environment is the enablement of semantic web technologies on the gateway which obviously will require quite a number of resources. Filtering and annotating gigabytes and petabytes of data on a device such as the gateway can lead to bottleneck processing and high latency with low throughput. This is because the gateway has to process each data it received from the sensor node regardless of whether the data is complete or not, thus increasing resource utilization, inconsistency and network traffic at the gateway.

2.7 Data Source

Data.gov is an online repository of the United State(U.S.)'s open data. It is a warehouse to find data, tools, and resources to conduct research, develop web and mobile applications and design data visualizations. Data.gov is managed and hosted by the U.S. General Services Administration, Technology Transformation Service (Data.gov, 2018).

On the platform, there are datasets on variety of topics such as Agriculture, Climate, Consumer, Ecosystems, Education, Energy, Finance, Health, Local government, Manufacturing, Maritime,

Ocean, Public safety, Science and Research. As of June 2017, the approximately 200,000 datasets reported as the total on Data.gov represents about 10 million data resources (Data.gov, 2017). While as of the time of conducting this research Data.gov has Three thousand and one, six hundred and sixty-eight (301,668) datasets (Data.gov, 2018).

The Data.gov team typically works with a designated open data point of contact as a liaison for each agency. Therefore, to publish data on Data.gov, Data publishers consults with their agency point of contact to include datasets on Data.gov. Every dataset displayed on the platform follows the Project Open Data schema which is a set of required fields which includes Title, Description, Tags, Last Update, Publisher and Contact Name.

This dataset is results from a 2017-2018 project of City-installed Smart Green Infrastructure Monitoring Sensors (SGIMS), measuring water runoff from streets and sidewalks in the city of Chicago. These data can be used to measure the impact of sustainable green infrastructure on flooding. These sensors also captured weather data.

The file contains over 18 Million sensor records which amounts to 1.5 GigaByte (GB) of file size. Each row corresponds to a sensor measurement at a specific time and location. Each row is a different sensor, which can be determined from the "Measurement Title" column. The value for each measurement is always numeric and available in the "Measurement Value" column. The corresponding unit of measurement is in the "Units" column. Data may be missing at times due to sensors not being available (Levy, 2018).

The raw data was exported to Microsoft Excel in comma separated values (CSV) format, the data also contains various elements that are not required for this study as only five elements are required for this research, these elements are sensor ID, sensor model, sensor type, measured

value and timestamp. In order to carry out the analysis intended for this research, the relevant features representing earlier cited elements were extracted from the original dataset to produce well-formed dataset suitable for the intended analysis. Figure 3.1 shows the format in which the data are stored after being generated by sensors in CSV file format.

C	D	E	F	G	H	I	J	K
Measurement Type	Measur	Measurement Time	Measurement Value	Units	Units Abb	Measure Data	Street Resource	
ation TimeWindowBounda	NA	5/4/2018 7:00	61	universal	UTC	Instantiar	33264	65
ation TimeWindowBounda	NA	5/3/2018 19:00	61	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	5/3/2018 19:00	0.3	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	5/3/2018 13:00	0.08	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	5/3/2018 7:00	59	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	5/3/2018 7:00	0.12	universal	UTC	Instantiar	33265	65
ation F TimeWindowBounda	NA	5/3/2018 1:00	0.57	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	5/2/2018 19:00	85	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	5/2/2018 19:00	0.21	universal	UTC	Instantiar	33265	65
ation F TimeWindowBounda	NA	5/2/2018 13:00	0.03	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	5/2/2018 7:00	41	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	5/2/2018 7:00	0	universal	UTC	Instantiar	33265	65
ation F TimeWindowBounda	NA	5/2/2018 1:00	0	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	5/1/2018 19:00	7	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	5/1/2018 19:00	0	universal	UTC	Instantiar	33265	65
ation F TimeWindowBounda	NA	5/1/2018 13:00	0	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	5/1/2018 7:00	3	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	5/1/2018 7:00	0	universal	UTC	Instantiar	33265	65
ation F TimeWindowBounda	NA	5/1/2018 1:00	0	universal	UTC	Instantiar	33265	65
ation TimeWindowBounda	NA	4/30/2018 19:00	0	universal	UTC	Instantiar	33264	65
ation F TimeWindowBounda	NA	4/30/2018 19:00	0	universal	UTC	Instantiar	33265	65
ation F TimeWindowBounda	NA	4/30/2018 13:00	0	universal	UTC	Instantiar	33265	65

Figure 2.5: Smart Green Infrastructure Monitoring Sensors sample data (Data.gov, 2018)

```

076_..HistoricalCSV X
: Description,Measurement Type,Measurement Medium,Measurement Time,Measurement Value,Units,Units Abbrevi
ID,Measurement ID,Record ID,Latitude,Longitude,Location
ity of Precipitation, TimeWindowBoundary, NA, 5/4/2018 7:00, 61, universal, coordinated
65153, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 5/3/2018 19:00, 61, universal, coordinated
65153, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/3/2018 19:00, 0.3, universal, coordinated
65148, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/3/2018 13:00, 0.08, universal, coordinated
65147, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 5/3/2018 7:00, 59, universal, coordinated
65152, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/3/2018 7:00, 0.12, universal, coordinated
65146, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/2/2018 1:00, 0.57, universal, coordinated
65145, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 5/2/2018 19:00, 85, universal, coordinated
65151, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/2/2018 19:00, 0.21, universal, coordinated
65144, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/2/2018 13:00, 0.05, universal, coordinated
65144, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 5/2/2018 7:00, 41, universal, coordinated
65150, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/2/2018 7:00, 0, universal, coordinated
65142, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 5/1/2018 19:00, 7, universal, coordinated
65150, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/1/2018 19:00, 0, universal, coordinated
65144, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/1/2018 13:00, 0, universal, coordinated
65143, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 5/1/2018 7:00, 3, universal, coordinated
65150, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 5/1/2018 7:00, 0, universal, coordinated
65142, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
ity of Precipitation, TimeWindowBoundary, NA, 4/30/2018 19:00, 0, universal, coordinated
65150, 2.51877E+18, 3.32643E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 4/30/2018 19:00, 0, universal, coordinated
65142, 2.51877E+18, 3.32653E+23, 41.90715, -87.653996, POINT (-87.653996 41.90715)
itive Precipitation Forecast, TimeWindowBoundary, NA, 4/30/2018 13:00, 0, universal, coordinated

```

Figure 2.6: Smart Green Infrastructure Monitoring Sensors sample data - CSV (Data.gov, 2018)

CHAPTER THREE

DESIGN OF AN ENHANCED DATA FILTERING AND ANNOTATION TECHNIQUE FOR IOT SENSORS

3.1 Introduction

In this chapter the method designed for preprocessing datasets so as to remove unnecessary attributes used in this research is discussed. In addition, a distributed data storage architecture

and MapReduce algorithm for data structuring and annotation of preprocessed data are also developed. The chapter concludes with the design of proposed system architecture.

3.2 Data Preparation and Preprocessing Method

Data preparation and preprocessing deals with preparing the raw dataset or cleaning it to obtain the required data that will be suitable for analysis. The preprocessing tasks in this context include removal of unwanted fields in the dataset. Data cleaning task was carried out to remove data that are of no interest in the research. These include information about the sensors, location of the sensor and measurement attributes that are not part of the attributes needed. The major task involved in data preprocessing include Data extraction, Data cleaning and Data transformation. Figure 3.2 shows the data preparation tasks used in this research.

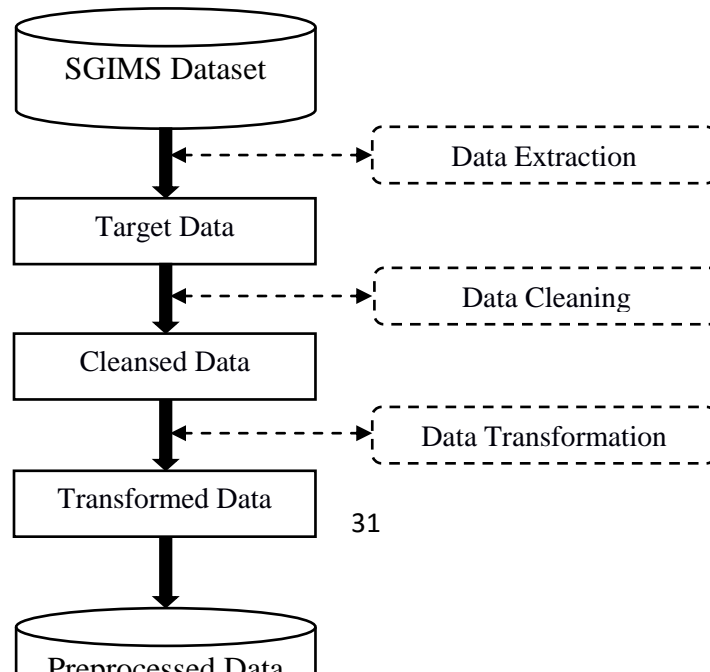


Figure 3.1: Data Preparation and Preprocessing Tasks

3.2.1 Data Extraction

Smart Green Infrastructure Monitoring Sensors (SGIMS) dataset has huge amount of records comprising of several attributes such as Measurement Title, Measurement Description, Measurement Type, Measurement Medium, Measurement Time, Measurement Value, Units, Units Abbreviation, Measurement Period Type, Data Stream ID, Resource ID, Measurement ID, Record ID, Latitude, Longitude and Location. However, only Resource ID, Measurement Title, Measurement Description, Measurement Value and Measurement Time were selected because these were the same attributes used by Al-Ostaet *al.* (2017) whose work is being enhanced in this research. These would ensure that both the existing work and the proposed system are tested with the same dataset. It is worth noting that at some point, Measurement Description values for sensors are missing in the dataset. This is as a result of the sensor's unavailability sometimes, as stated by Levy (2018). Table 3.1 and Figure 3.3 gives the descriptions of the selected attributes.

Table 3.1: Attributes Description

S/No	Attributes	Description
1	Resource ID	Unique identity of the sensor
2	Measurement Title	Information about sensor name, model and lab
3	Measurement Description	Category or type of sensor being used
4	Measurement Value	Sensor measured value

5	Measurement Time	Date and time the measurement was captured
---	------------------	--

3.2.2 Data Cleaning

Data cleaning has to do with data enhancement, where data is made more complete by adding related information. In this case a number of missing values obtained as a result of unavailability of sensors were left untouched. These would aid the proposed system's ability to determine faulty sensors. Furthermore, at this stage, data attribute names were renamed to ensure uniformity with the model used in the existing system. Another reason is because these names would be used for annotating the attribute values at a later stage. Finally, the attributes and respective values were rearranged in a particular order. Although this does not have any impact on the intending analysis, but for clarity. Table 3.2 shows the cleaned and ordered attributes.

Table 3.2: Cleaned Attributes

S/No	Attribute	Cleaned Attribute
1	Resource ID	Sensor ID
2	Measurement title	Sensor model
3	Measurement Description	Sensor type
4	Measurement value	Measured value
5	measurement time	Timestamp

3.2.3 Data Transformation

Data transformation is part of data preprocessing task which involve normalization and aggregation, it is used to improve the quality of data. The data transformation technique used in this dissertation is aggregation which is the procedure of bringing data closer to the requirements of the algorithms, or to preprocess data so as to ease the algorithm's task.

By default, the SGIMS data is comma delimited which means attribute values in a record are comma separated. However, the algorithm used in the proposed system is applicable on tab delimited records. As a result, the sample data had to be transformed from a comma delimited values to tab delimited values. Sublime text, a multipurpose text editor was used to convert the sample data from Comma separated to Tab delimited using regular expression.

3.3 Distributed Data Storage and Parallel Data Processing

Data streams received from the gateway node are stored in a distributed fashion in the cloud using Hadoop Distributed File System for reliability, availability, fault tolerance and efficient processing. Cloud-based program for parallel filtering and annotation of data is deployed.

The huge amount of data collected by the sensors can be processed, analyzed, and stored using the computational and data storage service of the cloud. In this architecture, the sensor data can be efficiently stored and processed by different commodity hard-ware in a cluster. This transforms data into a format that is interpretable by machines.

3.3.1 Distributed Data Storage on HDFS

HDFS is a file system designed for storing very large files running on cluster of non-expensive, low-end hardware used for daily purpose. It is designed on the principle of storage of a smaller number of large files. However, it provides fault tolerant storage layer for Hadoop and its other components. HDFS stores data reliably even in case of hardware failure and it provides high-throughput access to application data. HDFS has two types of nodes which works in master-slave fashion, these include the HDFS master node and the HDFS slave node.

In this work, a single node cluster approach is adopted, this implies that the slave node daemons and the master node daemons run on a single machine. As shown in Figure 3.2, immediately

large data streams are received by the master node for storage, the Hadoop framework automatically split the large dataset into blocks. A block has maximum dataset size of 128 megabyte (MB). Each block is then stored on different slave nodes. By default, the Hadoop framework creates 2 replicas of each block and store the blocks on different slaves. This allows for data availability in case of hardware failure. The number of blocks is determine using the following formula:

$$NB = FS/128$$

Where NB is the Number of Blocks and FS is the File Size.

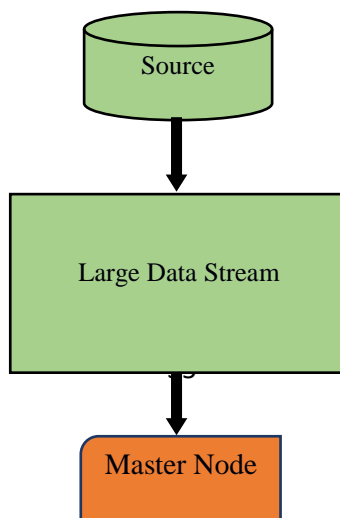


Figure 3.2: Distributed Data Storage Architecture

To demonstrate the workability of this approach, a sensor dataset of size 1.42GB was sent for storage on HDFS. The Hadoop framework split the dataset into 11 blocks of 128MB each, with the 12th block having size 49.55MB as shown in Figure 3.3.

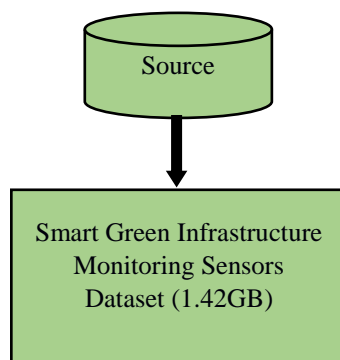


Figure 3.3: Distributed Storage of 1.42GB GSIMS Data

3.3.2 Parallel Data Processing using MapReduce Model

In the Chapter 2 of this work, it was established that the MapReduce model is implemented using master/slave architecture (Dean and Ghemawat, 2010). A job might require only the Map Phase without the Reduce Phase because in reduce process basically what happens is an aggregation of values or rather an operation on values that share the same key. Such jobs that do not require a reduce process are called Map Only Jobs. The implementation of the proposed system in this

research requires only the Map Phase because no sorting or aggregation of annotated sensor record is required.

The master machine is responsible for the assignment of tasks and controlling the slave machines. At this point the data stream chunks stored over the Hadoop distributed file system are fetched by slave nodes in a Key-Value Pair (KVP) fashion for processing. First, a copy of map functions' algorithm or code is sent to all slave nodes as shown in Figure 3.4. Then, master node assigns map tasks to workers. Each worker assigned a map task, reads the corresponding input split and passes all of its pairs to map function and writes the results of the map function into intermediate files.

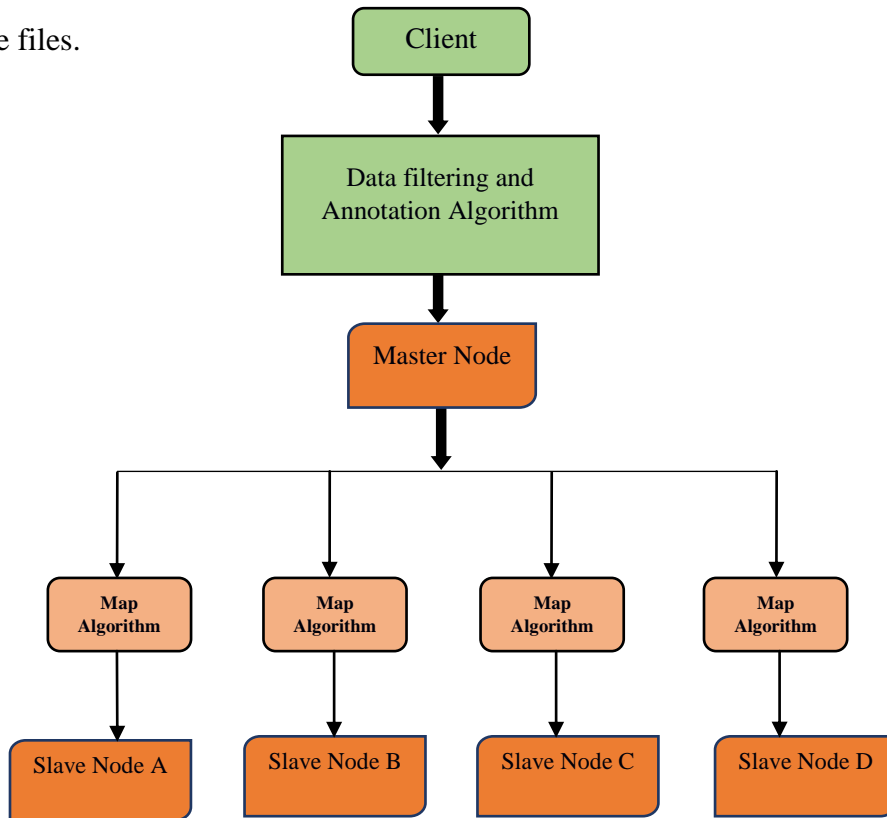


Figure 3.4: Parallel Data Processing Architecture

Algorithm 1 depicts the step by step procedure of the Hadoop map function where sensor data

Algorithm 1: Proposed Algorithm for Parallel Filtering and Annotation of Sensor Data

1. **input:** sensorRecord = data obtained from sensors (sensor id, model, type, measured value, timestamp)
2. **output:** processedSensorRecord
3. FOR EACH sensorRecords as sensorRecord
4. tokens = tokenize sensorRecord by tab
5. count = count the number of tokens
6. IF (count is equal to 5)
7. structuredRecord = structure (sensorRecord)
8. annotatedRecord = annotate(structuredRecord)
9. processedSensorRecord = annotatedRecord
10. store processedSensorRecord
11. ELSE
12. log sensorRecord as invalid
13. store sensorRecord
14. END IF
15. END FOR EACH

filtering and annotation is performed.

Algorithm 1 illustrates the steps involved in parallel processing of sensor data. Lines 1-2 outline the expected input and output in the pseudo code. It is assumed that each sensor has been programmed to published its ID, model and type. Subsequently, sensors send their ID alongside the value measured from their environment and the time the data was captured. While line 3 reads data record one at a time, line 4 tokenizes the record in-to tokens using tab character as delimiter (assuming sensor Id, model, type, measured value and timestamp are tab delimited). Line 5 counts the number of tokens obtained to validate the record for further step.

The expected number of tokens is 5. If number of tokens generated from a record equals five (5), the record is processed and its output stored as shown in lines 7-10. Otherwise, the record is taken to be invalid and logged for further decision making (lines 12-13).

3.4 System Architecture

The representation of the system architecture of the entire model is given in the Figure 3.5. Firstly, preprocessed dataset and algorithm are passed to the master node. The master node then splits the dataset into blocks which are distributedly stored in slave nodes. The master node also sends a copy of the MapReduce algorithm to each of the slaves for parallel processing of data blocks after which resulting processed records on each slave are stored on HDFS. Components within dotted lines were adopted from the existing work and improved on.

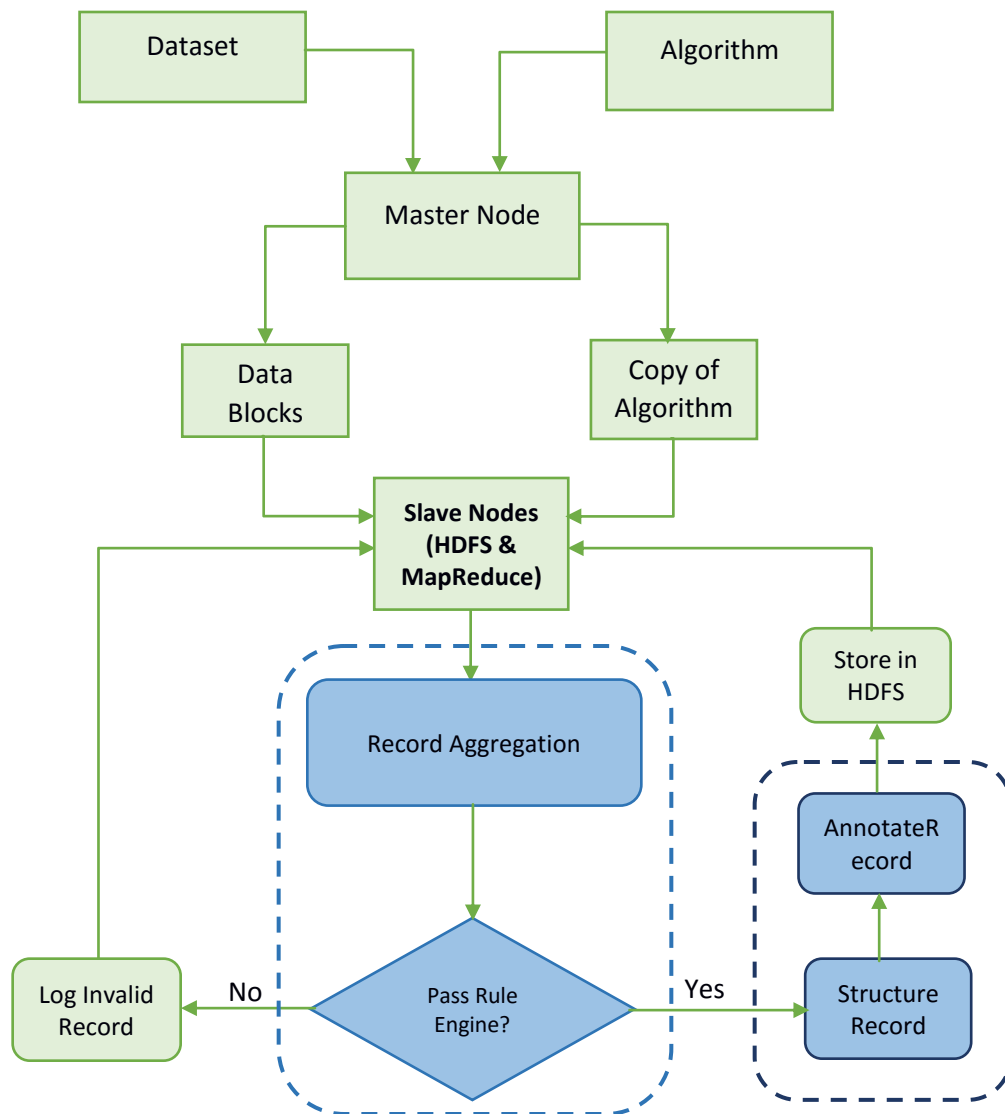


Figure 3.5: System Architecture

CHAPTER FOUR

IMPLEMENTATION, RESULT AND ANALYSIS

4.1 Introduction

HDFS is a distributed storage system that provide high-throughput access to application data such as the GSIMS data. Its ability to store data distributedly makes it amongst the most reliable storage system for large data. As stated in chapter two of this work, before the experiment for this research was conducted, the dataset being processed by proposed system had to be stored in a distributed fashion. As soon as the data was received for storage, the namenode (master node) created splits (blocks) based on the size of dataset provided and store each split on a datanode (slave node). Number of blocks created for 1.42GB dataset is shown in Figure 4.1. Details of the splits are shown in Table 4.1.

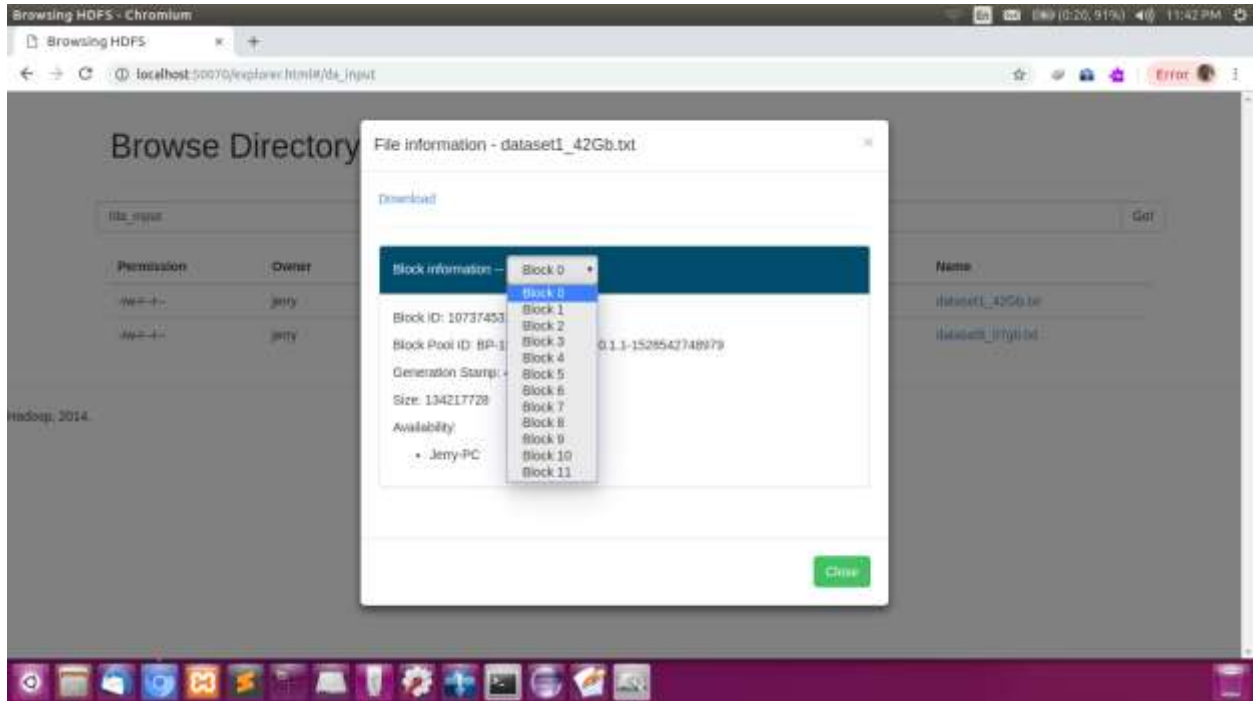


Figure 4.1: 1.42GB Blocks Information

4.2 Block Information

Table 4.1 shows the information of blocks created on storage of 1.42GB dataset. Each block has a default maximum size of 128MB. Hence, the dataset was divided into 12 blocks.

Table 4.1: Block information for 1.42GB dataset storage

Block No	Block ID	Block Size (Bytes)	Block Size (MB)	Generation Stamp
Block 0	1073743748	134217728	128	2927
Block 1	1073743749	134217728	128	2928
Block 2	1073743750	134217728	128	2929
Block 3	1073743751	134217728	128	2930
Block 4	1073743752	134217728	128	2931
Block 5	1073743753	134217728	128	2932
Block 6	1073743754	134217728	128	2933
Block 7	1073743755	134217728	128	2934
Block 8	1073743756	134217728	128	2935
Block 9	1073743757	134217728	128	2936
Block 10	1073743758	134217728	128	2937
Block 11	1073743759	51957271	49.55	2938

Each split created comes with a Block No, Block ID, Block Size (Bytes) and Generation Stamp. From Table 4.1 it can be observed that 11 blocks created had a Block Size of 128MB respectively. However, the twelfth block has a Block Size of 49.55. This is because the sensor record remaining after the eleventh block was created was not up to 128MB. Hence only a block of 49.55MB was created. Table 4.2, 4.3 and 4.4 shows the Block Information for 2.14GB, 3.02GB and 5.07GB dataset respectively. According to Suhasini (2019), resulting blocks of processed data can be queried using Apache Pig and Hive.

Table 4.2: Block information for 2.14GB dataset storage

Block No	Block ID	Block Size (Bytes)	Block Size (MB)	Generation Stamp
Block 0	1073743645	134217728	128	2824
Block 1	1073743646	134217728	128	2825
Block 2	1073743647	134217728	128	2826
Block 3	1073743648	134217728	128	2827
Block 4	1073743649	134217728	128	2828
Block 5	1073743650	134217728	128	2829
Block 6	1073743651	134217728	128	2830
Block 7	1073743652	134217728	128	2831
Block 8	1073743653	134217728	128	2832
Block 9	1073743654	134217728	128	2833
Block 10	1073743655	134217728	128	2834
Block 11	1073743656	134217728	128	2835
Block 12	1073743657	134217728	128	2836
Block 13	1073743658	134217728	128	2837
Block 14	1073743659	134217728	128	2838
Block 15	1073743660	134217728	128	2839
Block 16	1073743661	134217728	128	2840
Block 17	1073743662	10827042	10.33	2841

Table 4.3: Block information for 3.02GB dataset storage

Block No	Block ID	Block Size (Bytes)	Block Size (MB)	Generation Stamp
Block 0	1073743999	134217728	128	3178
Block 1	1073744000	134217728	128	3179
Block 2	1073744001	134217728	128	3180
Block 3	1073744002	134217728	128	3181
Block 4	1073744003	134217728	128	3182
Block 5	1073744004	134217728	128	3183
Block 6	1073744005	134217728	128	3184
Block 7	1073744006	134217728	128	3185
Block 8	1073744007	134217728	128	3186
Block 9	1073744008	134217728	128	3187
Block 10	1073744009	134217728	128	3188
Block 11	1073744010	134217728	128	3189
Block 12	1073744011	134217728	128	3190
Block 13	1073744012	134217728	128	3191
Block 14	1073744013	134217728	128	3192
Block 15	1073744014	134217728	128	3193
Block 16	1073744015	134217728	128	3194
Block 17	1073744016	134217728	128	3195
Block 18	1073744017	134217728	128	3196
Block 19	1073744018	134217728	128	3197
Block 20	1073744019	134217728	128	3198
Block 21	1073744020	134217728	128	3199
Block 22	1073744021	134217728	128	3200
Block 23	1073744022	134217728	128	3201
Block 24	1073744023	26523012	25.29	3202

Table 4.4: Block information for 5.07GB dataset storage

Block No	Block ID	Block Size (Bytes)	Block Size (MB)	Generation Stamp
Block 0	1073744581	134217728	128	3766
Block 1	1073744582	134217728	128	3767
Block 2	1073744583	134217728	128	3768
Block 3	1073744584	134217728	128	3769
Block 4	1073744585	134217728	128	3770
Block 5	1073744586	134217728	128	3771
Block 6	1073744587	134217728	128	3772
Block 7	1073744588	134217728	128	3773
Block 8	1073744589	134217728	128	3774
Block 9	1073744590	134217728	128	3775
Block 10	1073744591	134217728	128	3776
Block 11	1073744592	134217728	128	3777
Block 12	1073744593	134217728	128	3778
Block 13	1073744594	134217728	128	3779
Block 14	1073744595	134217728	128	3780
Block 15	1073744596	134217728	128	3781
Block 16	1073744597	134217728	128	3782
Block 17	1073744598	134217728	128	3783
Block 18	1073744599	134217728	128	3784
Block 19	1073744600	134217728	128	3785
Block 20	1073744601	134217728	128	3786
Block 21	1073744602	134217728	128	3787
Block 22	1073744603	134217728	128	3788
Block 23	1073744604	134217728	128	3789
Block 24	1073744605	134217728	128	3790

Block No	Block ID	Block Size (Bytes)	Block Size (MB)	Generation Stamp
Block 25	1073744606	134217728	128	3791
Block 26	1073744607	134217728	128	3792
Block 27	1073744608	134217728	128	3793
Block 28	1073744609	134217728	128	3794
Block 29	1073744610	134217728	128	3795
Block 30	1073744611	134217728	128	3796
Block 31	1073744612	134217728	128	3797
Block 32	1073744613	134217728	128	3798
Block 33	1073744614	134217728	128	3799
Block 34	1073744615	134217728	128	3800
Block 35	1073744616	134217728	128	3801
Block 36	1073744617	134217728	128	3802
Block 37	1073744618	134217728	128	3803
Block 38	1073744619	134217728	128	3804
Block 39	1073744620	134217728	128	3805
Block 40	1073744621	76045818	72.52	3806

4.3 Experimental Results for Data Processing on Existing and Proposed System

This section describes the result obtained after several datasets were processed using the existing system (without enhanced data processing) and on the proposed system (with enhanced data processing). As stated in section 3.6, comparison between the two systems was also carried out based on size of the resulting processed file and the time taken for processing the dataset on both systems.

4.3.1 Evaluation and Analysis of Data Processing on Existing System

This section analysis observation made while processing four different datasets of size 1.42GB, 2.14GB, 3.02GB and 5.07GB on the existing systems. Recall that major metrics of interest are Processing Time and RDF File Size. Following subsections (a) and (b) analyze and discuss the results and observations made in detail.

a. Processing Time

Table 4.5: Time taken for processing datasets on existing System

Dataset Size (GB)	Processing Time (minutes)
1.42	82.8
2.14	212.4
3.02	372
5.07	992.2

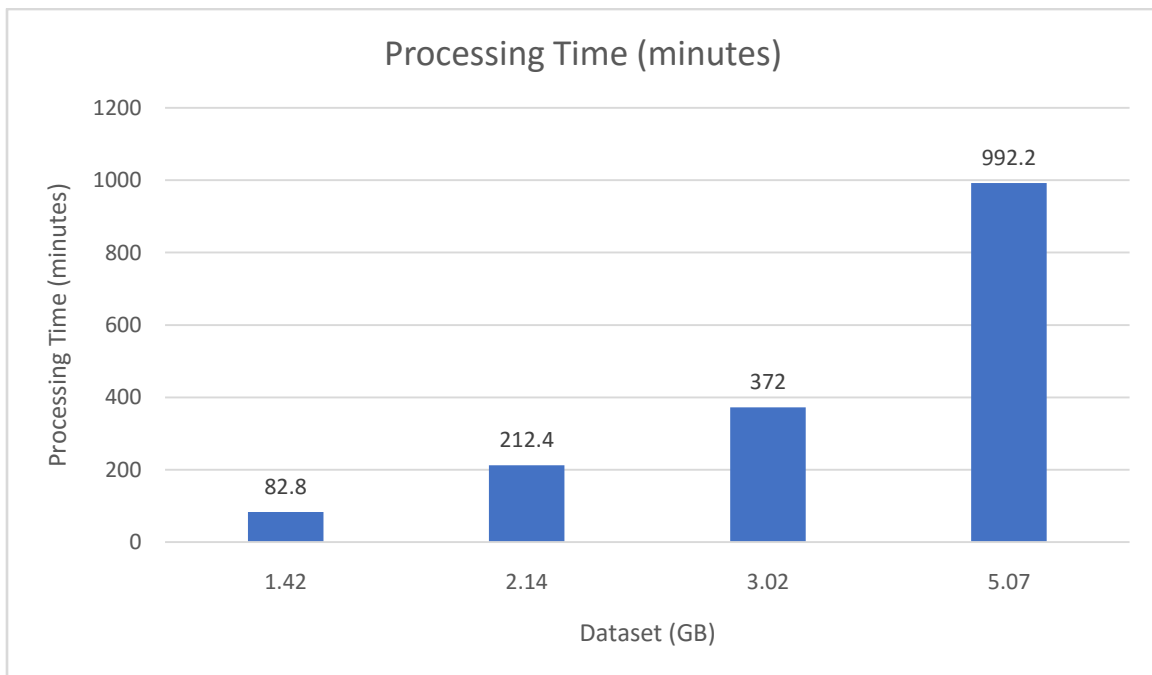


Figure 4.2: Time taken for processing datasets on existing System

As shown in Table 4.5 and Figure 4.2, each dataset takes a respective amount of time to be processed. The more the size of the dataset, the more processing time it requires. Collectively, the four datasets being used sums up to 11.65GB. Processing this data on the existing system took 1659.4 minutes which is also equivalent to 27.66 hours.

b. File Size

Table 4.6: Processed file size on existing System

Dataset Size (GB)	Processed File Size (MB)	Processed File Size (GB)
1.42	3589.6	3.59
2.14	5384.43	5.38
3.02	7627.94	7.63
5.07	12547.76	12.55

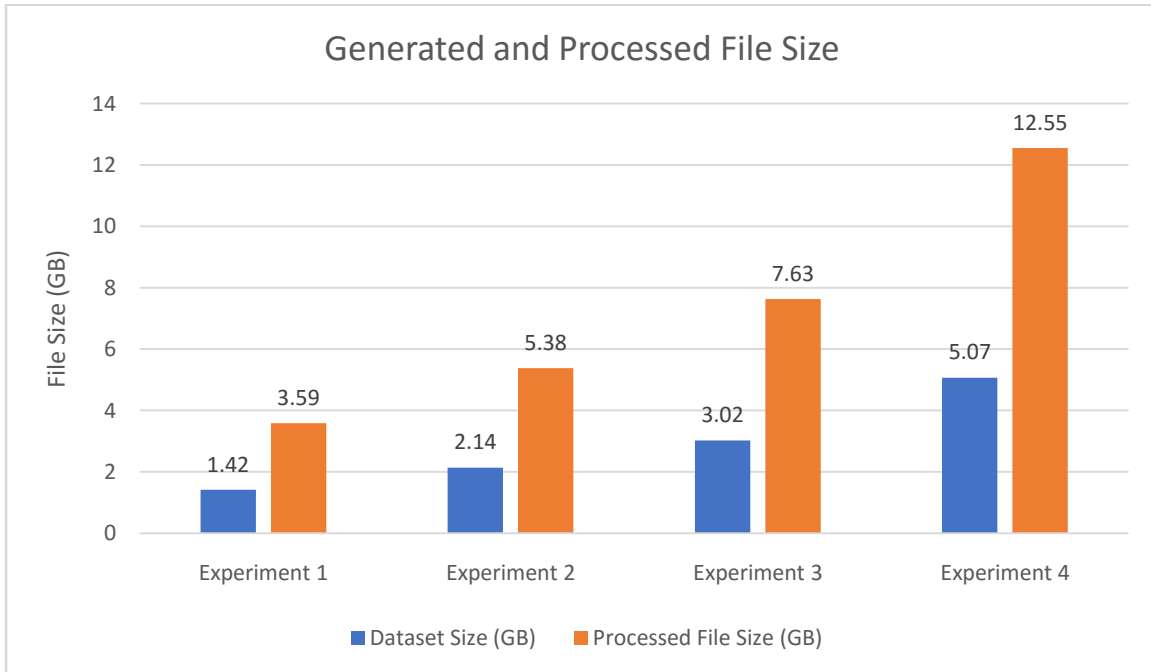


Figure 4.3: Processed file size on existing System

Also, as shown in Table 4.6 and Figure 4.3, processing a dataset of 11.65GB on the existing system generates an RDF document of about 29.15GB, which is almost 3 times the actual size of the raw sensor data.

4.3.2 Evaluation and Analysis of Data Processing on Proposed System

a. Processing Time

Table 4.7: Time taken for processing datasets on Proposed System

Dataset Size (GB)	Processing Time (minutes)
1.42	53.5
2.14	82.4
3.02	114.5
5.07	190.5

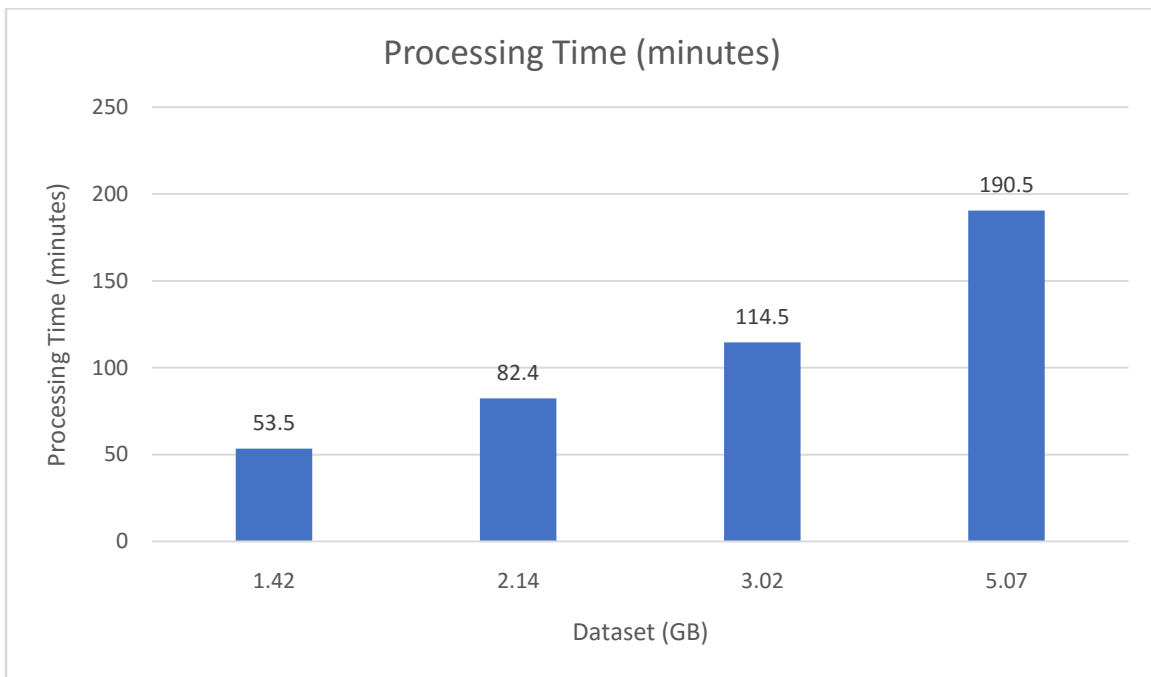


Figure 4.4: Time taken for processing datasets on Proposed System

As shown in Table 4.5 and Figure 4.4, processing a dataset of 11.65GB on the existing system takes only 58.4 minutes which is less than an hour.

b. File Size

Contrary to the procedure of generating RDF document in the system proposed by Al-Ostaet *al.* (2017) which processes any sensor record regardless of whether it’s a valid record or not. The proposed system in this work sidelines any incomplete record as described in section 3.3.2 of this work. Table 4.8 and 4.9 highlights the size of RDF document generated and the size of incomplete sensor data recorded respectively.

Table 4.8: Processed File Size on the Proposed System

Dataset Size (GB)	Processed File Size (MB)	Processed File Size (GB)
1.42	3099.73	3.09
2.14	4649.61	4.64
3.02	6586.94	6.58
5.07	11023	11.02

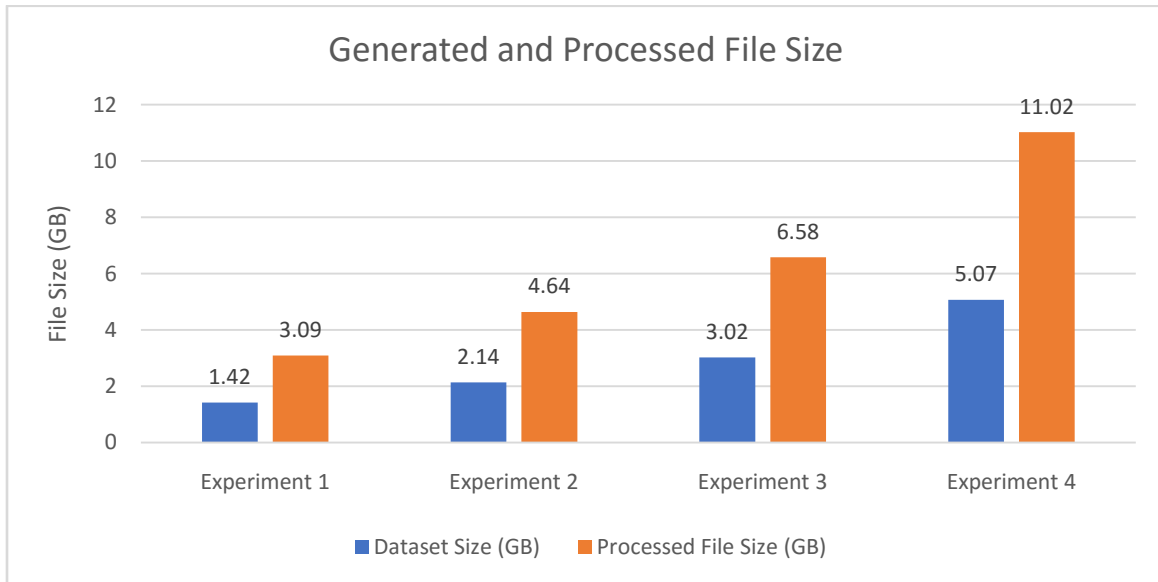


Figure 4.5: File size generated by Proposed System

Also, as shown in Table 4.8 and Figure 4.5, processing a dataset of 11.65GB on the proposed system generates an RDF document of about 25.33GB.

Table 4.9: Invalid sensor data size

Dataset Size (GB)	Invalid Record Size (MB)	Invalid Record Size (GB)
1.42	489.87	0.48
2.14	734.82	0.73
3.02	1041	1.04
5.07	1524.76	1.52

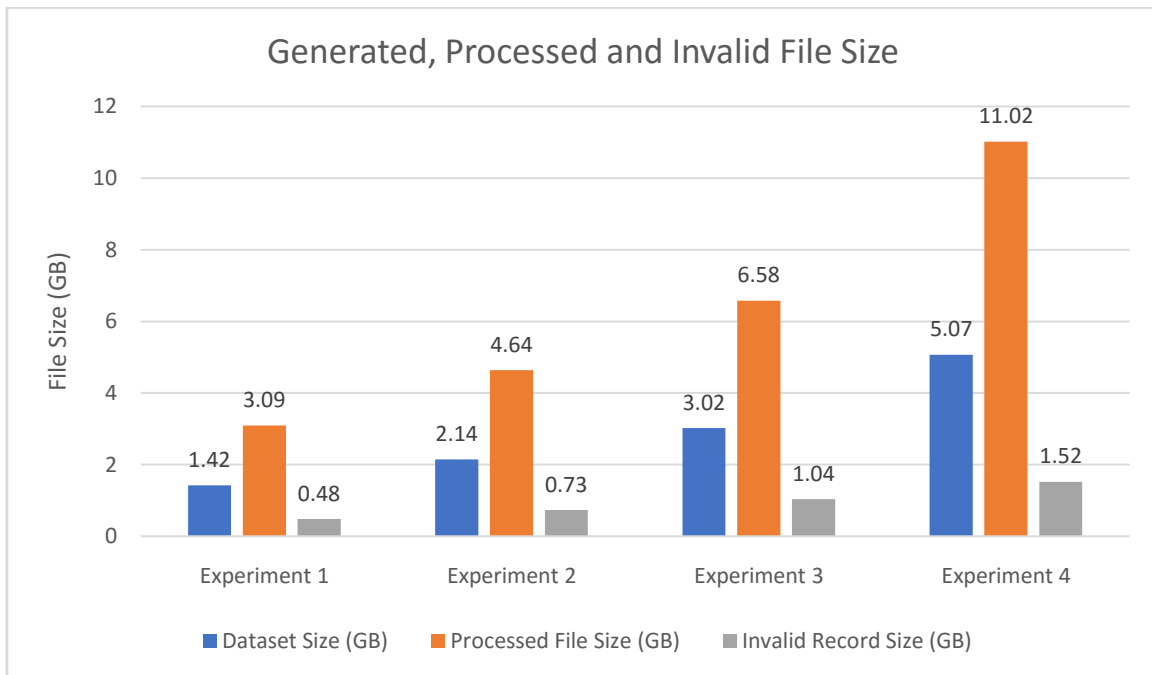


Figure 4.6: Processed and Invalid document size

As shown in Table 4.9, each processed dataset has a number of incomplete sensor record, these sensor records were extracted and stored in a separate file for further decision making. Result of the experiment shows that processing a huge dataset of 11.65GB generates about 3.77GB invalid record which would save the machine an extra time of processing and memory space. Figure 4.6

outlines the dataset size used in each experiment, the processed file size and the size of incomplete sensor record.

4.4 Comparative Analysis of Existing System and Proposed System

In this section, analysis of the Level of Enhancement (LoE) between the proposed system and existing system is described. In order to determine the LoE, percentage increase for each experiment was derived. These were used to determine average LoE for both Processing Time and File Size comparison metrics.

4.4.1 Processing Time Comparative Analysis

Table 4.10: Time taken for processing datasets on Existing and Proposed System

Dataset Size (GB)	Processing Time (minutes)		Enhancement Level (%)
	Existing System	Proposed System	
1.42	82.8	53.5	35.4
2.14	212.4	82.4	61.2
3.02	372	114.5	69.2
5.07	992.2	190.5	80.8
Total: 11.65	Total: 1659.4	Total: 58.4	Average: 61.65%

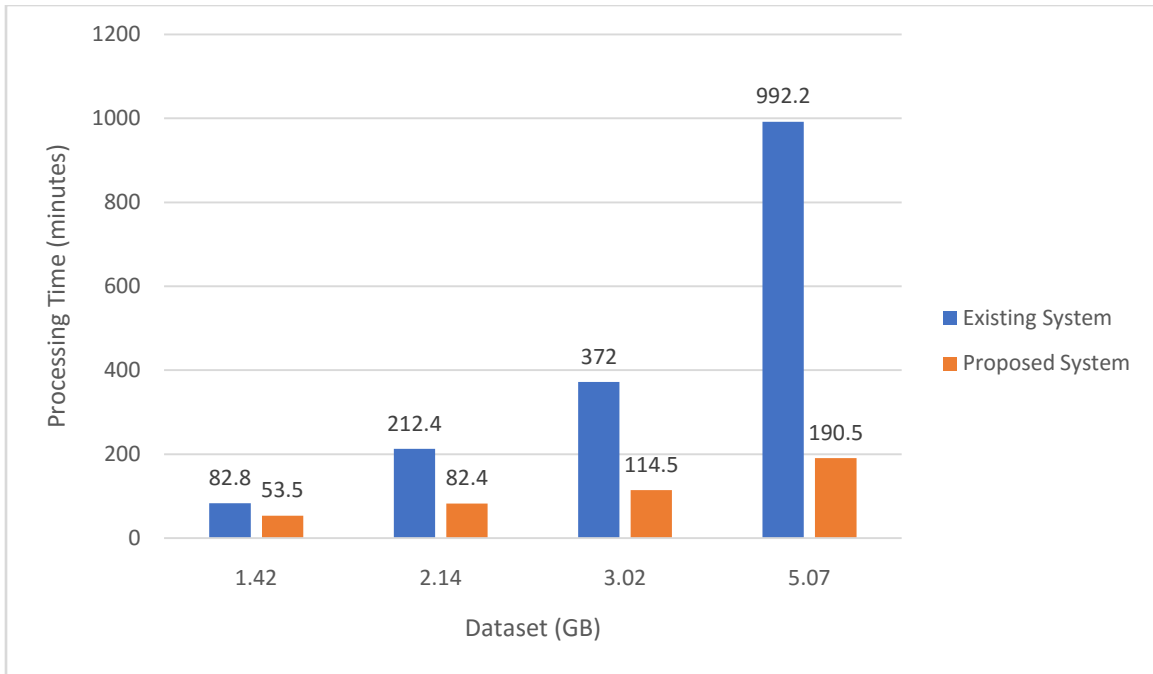


Figure 4.7: Processing Time on Existing and Proposed System

Table 4.10 indicates that existing system took 82.8 minutes while processing the 1.42GB dataset, whereas same amount of dataset was processed in 53.5 minutes on the proposed system thus having a 35.4% processing time percentage increase over existing system. Similarly, while it took the proposed system 82.4 minutes to process 2.14GB dataset, the existing system spent about 4hours processing same dataset. As a result, the proposed system recorded 61.2% increase. As shown in Table 4.10, subsequent experiments with 3.02GB and 5.07GB datasets revealed that the proposed system had enhancement increase of 69.2% and 80.8% respectively over the system developed by Al-Ostaet *al.*(2018). Ultimately, on an average the proposed system recorded 61.65% processing time enhancement over the existing system.

It is worth noting that the processing time of the proposed system is the sum of preprocessing time of raw sensory data and total time the system takes to generate RDF files that are consumable by IoT applications.

4.4.2 File Size Comparative Analysis

Table 4.11: Processed File Size on Existing and Proposed System

Dataset Size (GB)	Processed File Size (GB)		Enhancement Level (%)
	Existing System	Proposed System	
1.42	3.59	3.09	13.93
2.14	5.38	4.64	13.76
3.02	7.63	6.58	13.76
5.07	12.55	11.02	12.19
Total: 11.65	Total: 29.15	Total: 25.33	Average: 13.41%

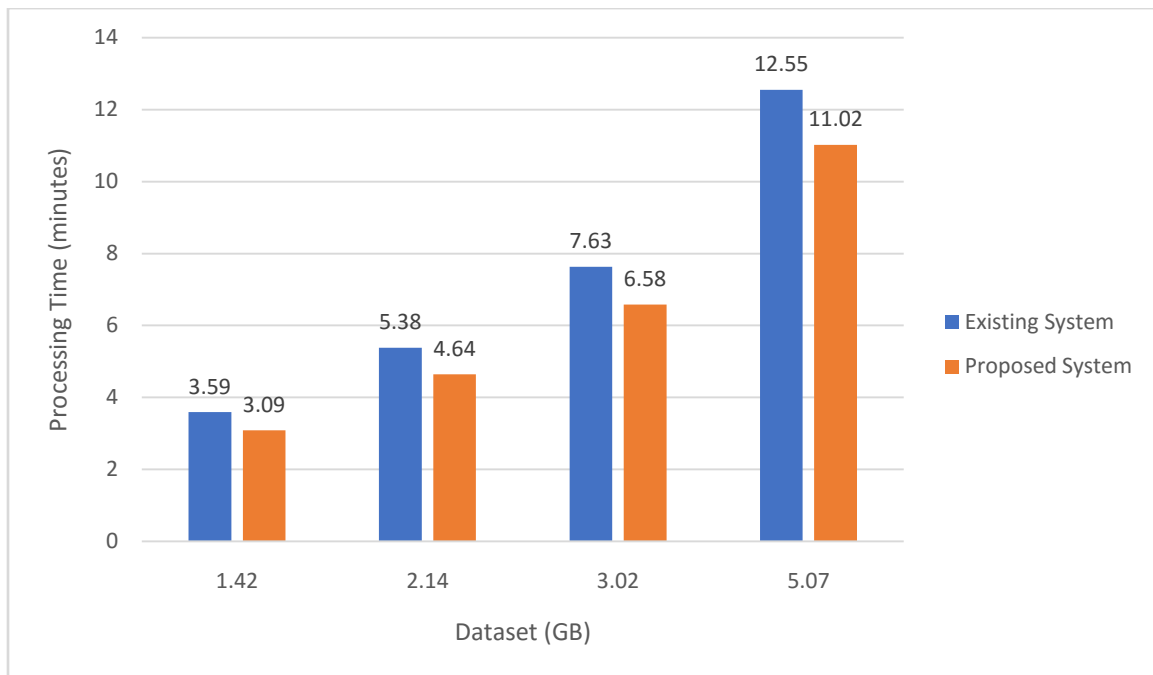


Figure 4.8: File size generated on Existing and Proposed System

While the experiments were being conducted, the size of resulting RDF document generated were observed and recorded as outlined in Table 4.11. This is to enable seamless evaluation of proposed enhancement. It is worth noting that only the Generated RDF document file size is

being considered, Invalid record are of no interest in this section. It's assumed that invalid record is used for better informed decision making and recommendations. Table 4.11 indicates that existing system generated RDF document of size 3.59GB from processing 1.42GB dataset, whereas same amount of dataset generated 3.09GB on the proposed system thus having a 13.93% percentage increase over existing system. Furthermore, while the proposed system produced 4.64GB resulting document from processing 2.14GB dataset, the existing system output 5.38GB worth of document processing same dataset. As a result, the proposed system recorded 13.76% increase. As shown in Table 4.11, subsequent experiments with 3.02GB and 5.07GB datasets revealed that the proposed system had enhancement increase of 13.76% and 12.19% respectively over the system developed by Al-Ostaet *al.*(2018). In summary, the proposed system had 13.41% enhancement over the existing system.

CHAPTER FIVE

SUMMARY, CONCLUSION AND FUTURE WORK

5.1 Summary

Huge amount of data is being generated by sensors on daily basis and most often these data are store in cloud for consumption by IoT applications and IoT developers. IoT applications encounter challenges consuming these data because of its heterogeneity. Despite several solutions developed, a few challenges such as high latency in data processing, storage and querying of invalid data are still encountered. Hence, an enhanced storage-based data annotation technique was developed in this dissertation. The proposed system involved the design of a MapReduce algorithm which was implemented on Apache Hadoop for parallel processing and distributed storage on HDFS. To evaluate the feasibility of the proposed approach, data generated by sensors were stored on Hadoop Distributed File System (HDFS) and were processed by a MapReduce job. Semantic Web technologies such as Extensible Markup Language (XML) and Resource Description Framework (RDF) were employed for the data annotation. Two categories of experimentations were conducted and comparison between the proposed system and that of Al-Osta *et al.* (2017) were carried out based on data size and processing time.

5.2 Conclusion

The MapReduce based parallel processing algorithms developed in this dissertation uses a filtering mechanism to filter out invalid sensor record, which could be used for decision making, whereas valid sensor records were processed. Processed valid record are readily made available for consumption by IoT application. This dissertation also compares the data size and processing time of the proposed system and that of Al-Osta *et al.* (2017) to determine the level of enhancement of the proposed system. Four datasets of size 1.42GB, 2.14GB, 3.02GB and

5.07GB were used in experimenting the efficiency of the proposed system. Percentage increase, that is, the enhancement level of the proposed system over the existing for each dataset were calculated based on the results obtained from both systems and the average level of enhancement was calculated. This dissertation concludes that the proposed system has 61.65% processing time and 13.41% data size enhancement respectively over the existing system.

However, this work was not implemented and tested on multiple node cluster. Considering that the strength of Hadoop is in a cluster of commodity master/slave nodes, implementing this system on a single commodity node will reduce its performance. Also, this system does not consider the security of the data being processed or generated by sensors.

5.3Future Work

This dissertation can be further improved through the following suggestions:

1. IoT applications could be developed to query the processed data.
2. Also, as future work, the system developed in this dissertation can be deployed and tested on a multiple node cluster of several commodity hardware. This research focuses on using a single node cluster.
3. Larger datasets such as Terabytes and Petabytes with the necessary features should be used to conduct similar experiment in order to obtain more conclusive results.
4. Future research could also be channeled towards the security of the system as sensitive data could be part of the data being processed.
5. Finally, invalid sensor records could be further processed to trigger an alarm or notification once a number of invalid records is generated by a sensor.

REFERENCES

- Aggarwal, C. C., Ashish, N. and Sheth, A. (2013). *The internet of things: A survey from the data-centric perspective*. Managing and mining sensor data, Springer. DOI 10.1007/978-1-4614-6309-2_12: 383–428.
- Al-Osta, M., Ahmed, B. and Abdelouahed, G. (2017). A lightweight semantic web-based approach for data Annotation on IoT gateways. *The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017)*. Canada, pp. 186-193
- Antoniou, G. and van Harmelen, F. (2004). *A Semantic Web Primer*. MIT Press, Cambridge, MA.
- Arnulf, C. (2012). *Introduction to Semantic Web Technology and Geodata*. Retrieved From: [http://arnulf.us/publications/Introduction to Semantic Web Technology and Geodata v4.pdf](http://arnulf.us/publications/Introduction%20to%20Semantic%20Web%20Technology%20and%20Geodata%20v4.pdf) on 23/5/2018
- Barnaghi, P., Wang, W., Henson, C. and Taylor, K. (2012). Semantics for the internet of things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS)* 8 (1): 1–21.
- Borgia, E. (2014). The Internet of Things vision: Key features, applications and open issues. *Computer Communications* 54, pp. 1–31.
- Christophe, B. (2012). Managing massive data of the internet of things through cooperative semantic nodes. *International Journal of Semantic Computing* 6 (04): 389–408.
- Chen, J., Jimenez-Ruiz, E., Horrocks, I., and Sutton, C. (2019). Learning Semantic Annotations for Tabular Data. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. Pp. 2088 – 2094.
- Chun, S., Seo, S., Oh, B. and Lee, K. H. (2015). Semantic description, discovery and integration for the internet of things. *IEEE International Conference on Semantic Computing (ICSC)*, pp. 272–275.
- Cisco (2016). *Internet of Things at a Glance. Connected Means Informed*. Retrieved From: <https://www.cisco.com/internet-of-things> on 11/10/2018
- Data-Flair (2018). Deep Dive into MapReduce in Big Data Hadoop. *MapReduce Basics*. DataFlair Web Services Limited, India, pp. 1-31.
- Data.gov (2017). *Smart Green Infrastructure Monitoring Sensors – Historical*. data.cityofchicago.org. Retrieved From: <https://catalog.data.gov/dataset/sustainable-green-infrastructure-monitoring-sensors>

- Dean, J. and Ghemawat, S. (2010). MapReduce: a flexible data processing tool. *Communications of the ACM* 53 (1), pp. 72–77.
- Deepa, R. and Chezian, R. M. (2016). An Ontological Approach for the Semantic Web Search and the Keyword Similarity Metrics. *International Journal of Advanced Research in Computer and Communication Engineering* 5 (3): 678-682
- Desai, P., Sheth, A. and Anantharam, P. (2015). Semantic gateway as a service architecture for IoT interoperability. *IEEE International Conference on Mobile Services (MS)*, pp. 313–319.
- Dillon, T., Chang, E., Singh, J. and Hussain, O. (2012). Semantics of cyber-physical systems. *International Conference on Intelligent Information Processing*, Springer, pp. 3–12.
- Ding, Z., Yang, Q. and Wu, H. (2011). Massive heterogeneous sensor data management in the internet of things, in: Internet of Things (iThings/CPSCoM). *IEEE International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pp. 100–108.
- Edge, E. A. (2016). Overview of Semantic Web Technology: The Formulation of Semantic Web Agent System Model to Assist the Blind and Visually Impaired. *International Journal of Science and Technology* 6 (1): 1-31
- Evans, D. (2015). The internet of things: How the next evolution of the internet is changing everything. *Cisco*.
- Gopinath, G. and Sagayaraj, S. (2011). To Generate the Ontology from Java Source Code. (*IJACSA*) *International Journal of Advanced Computer Science and Applications*, 2 (2).
- GSMA (2014). *Understanding the Internet of Things (IoT)*. Retrieved From:
https://www.gsma.com/iot/wp-content/uploads/2014/08/cl_iot_wp_07_14.pdf on 12/3/2018
- Gyrard, A. (2013). An architecture to aggregate heterogeneous and semantic sensed data. *Extended Semantic Web Conference*, Springer, pp. 697–701.
- Hiba, A. and Shady, E. (2017). Effective searching of RDF knowledge graphs. *Web Semantics: Science, Services and Agents on the World Wide Web* 48: 66-84
- Isard, M., Budiu, M., Yu, Y., Birrell, A. and Fetterly, D. (2007). *Dryad: distributed data-parallel programs from sequential building blocks*. *SIGOPS Operating System Review*, 41(3):59–72.
- Levy, J. (2018). Dataset. Retrieved From: <https://catalog.data.gov/dataset?tags=iot>

- Jutamard, K. and Wiwat, V. (2016). A development of RDF data transfer and query on Hadoop Framework. *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*. DOI: 10.1109/ICIS.2016.7550760
- Kamburugamuve, S., Christiansen, L. and Fox, G. (2014). A Framework for Real-Time Processing of Sensor Data in the Cloud. *School of Informatics and Computing and Community Grids Laboratory*. Indiana University, Bloomington IN 47408 USA.
- Keyur, K. P. and Sunil, M. P. (2016). Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. *International Journal of Engineering Science and Computing* 6 (5): 6122-6131
- Khan, I., Jafrin, R., Errounda, F. Z., Glitho, R., Crespi, N., Morrow, M. and Polakos, P. (2015). A data annotation architecture for semantic applications in virtualized wireless sensor networks. *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 27–35.
- Kotis, K. and Katsanov, A. (2012). Semantic interoperability on the web of things: The semantic smart gateway framework. *IEEE Sixth International Conference Complex, Intelligent and Software Intensive Systems (CISIS)*, pp. 630–635.
- Laura, P. and Pamela, V. (2013). *Semantic Web in Action*. Retrieved From: http://www.globant.com/sites/default/files/pdf_white_papers/semantic-web.pdf on 7/6/2018
- Manyika, J. (2015). The Internet of Things: Mapping the value beyond the hype.
- Nasullah, K. A., Maozhen, L., Yang, L. and Suhel, H. (2011). *A MapReduce-based distributed SVM algorithm for automatic image annotation*. Computers and Mathematics with Applications. Pergamon Press, Inc. Tarrytown, NY, USA.
- Ovidiu, V. S. and Peter, F. (2014). Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems. *River publishers' series in communications*.
- Saeed, S. and Saeed J. (2014). *Beyond Batch Processing: Towards Real-Time and Streaming Big*.
Tarbiat Modares University (TMU), Tehran, Iran. pp. 1-11.
- Suhasini (2019). *Big Data and the Internet of Things (IoT)*. Retrieved From: <https://blogs.mastechinfotrellis.com/big-data-internet-things-iot>

Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N. and Dutta, J. (2015). *The Internet of Things Has a Gateway Problem*. ACM 978-1-4503-3391-7/15/02. DOI <http://dx.doi.org/10.1145/2699343.2699344>

APPENDIX A

I. Parallel Data Processing Algorithm

Algorithm for Parallel Filtering and Annotation of Sensor Data

16. **input:** sensorRecord = data obtained from sensors (sensor id, model, type, measured value, timestamp)
17. **output:** processedSensorRecord
18. FOR EACH sensorRecords as sensorRecord
19. tokens = tokenize sensorRecord by tab
20. count = count the number of tokens
21. IF (count is equal to 5)
22. processedSensorRecord = process (structure and annotate) sensor Record
23. store processedSensorRecord
24. ELSE
25. log sensorRecord as invalid
26. store sensorRecord
27. END IF
28. END FOR EACH

II. Mapper Class Code Snippet

```
package com.df.wc;
//Java Imports
import java.io.BufferedReader;
import java.io.IOException;
import java.util.StringTokenizer;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
```

```

import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

//Map reduce import
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Mapper.Context;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.Resource;
import org.w3c.dom.DOMImplementation;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

//input key, input value, output key, output value
public class DataAnnotationMapper extends Mapper <Object, Text, NullWritable, Text>
{
    private static String sensor_id = null;
    private static String sensor_type = null;
    private static String sensor_model = null;
    private static String sensor_value = null;
    private static String timestamp = null;

    // create Source and Namespace
    static String source = "http://jerryemmanuel.com/sdo";
    static String myNameSpace = source + "#" ;
    private Text invalid = new Text ("Invalid sensor data => ");

```



```

String processedRecord = "";
private MultipleOutputs<NullWritable, Text>mos = null;
public void setup (Context context){
    mos = new MultipleOutputs (context);
}
public void cleanup (Context context) throws IOException, InterruptedException{
    mos.close();
}
public void map(Object key, Text value, Context context) throws IOException,
InterruptedException
{
    String[] sensorRecord = value.toString().split("\t");
    String tempRecord = "\t<rdf:Descriptionrdf:about = '"+ myNameSpace +"'"+
sensorRecord[0] + ">\n";
    String[] tags = {"sdo:type", "sdo:model", "sdo:value", "sdo:timestamp"};

    int count = 0;
    if(sensorRecord.length == 5){
        for(int i = 0; i<tags.length; i++){
            if(sensorRecord[i+1].equals("")){
                count++;
                mos.write("BadRecords", NullWritable.get(), new Text
("Invalid sensor data => "+value));
                break;
            }else{
                tempRecord = tempRecord +
performDataAnnotation(sensorRecord[i+1], tags[i]);
            }
        }
        tempRecord += "\t</rdf:Description>";
        if(count == 0){
            processedRecord = tempRecord;
            mos.write("ParsedRecords", NullWritable.get(), new
Text(processedRecord));

```

```

        }
    }else{
        mos.write("BadRecords", NullWritable.get(), new Text ("Invalid sensor
data => "+value));
    }
}

public String performDataAnnotation(String value, String tagName){
    String tsValue = "\t\t<"+tagName+">"+value+"</"+tagName+">\n";
    return tsValue;
}
}
}

```

III. Driver Class Code Snippet

```

/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements. See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership. The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License. You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.df.wc;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

```

```

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class DataAnnotationDriver
{
    public static void main(String[] args) throws Exception
    {
        Path inputDir = new Path(args[0]);
        Path outputDir = new Path(args[1]);
        Configuration conf = new Configuration();
        Job job = new Job(conf);
        job.setJarByClass(WordCount.class);
        job.setJobName("Proposed System");

        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);

        job.setMapperClass(TokenizerMapper.class);

        FileInputFormat.setInputPaths(job, inputDir);
        FileOutputFormat.setOutputPath(job, outputDir);

        MultipleOutputs.addNamedOutput(job, "ParsedRecords", TextOutputFormat.class,
NullWritable.class, Text.class);
        MultipleOutputs.addNamedOutput(job, "BadRecords", TextOutputFormat.class,
NullWritable.class, Text.class);

        job.waitForCompletion(true);
    }
}

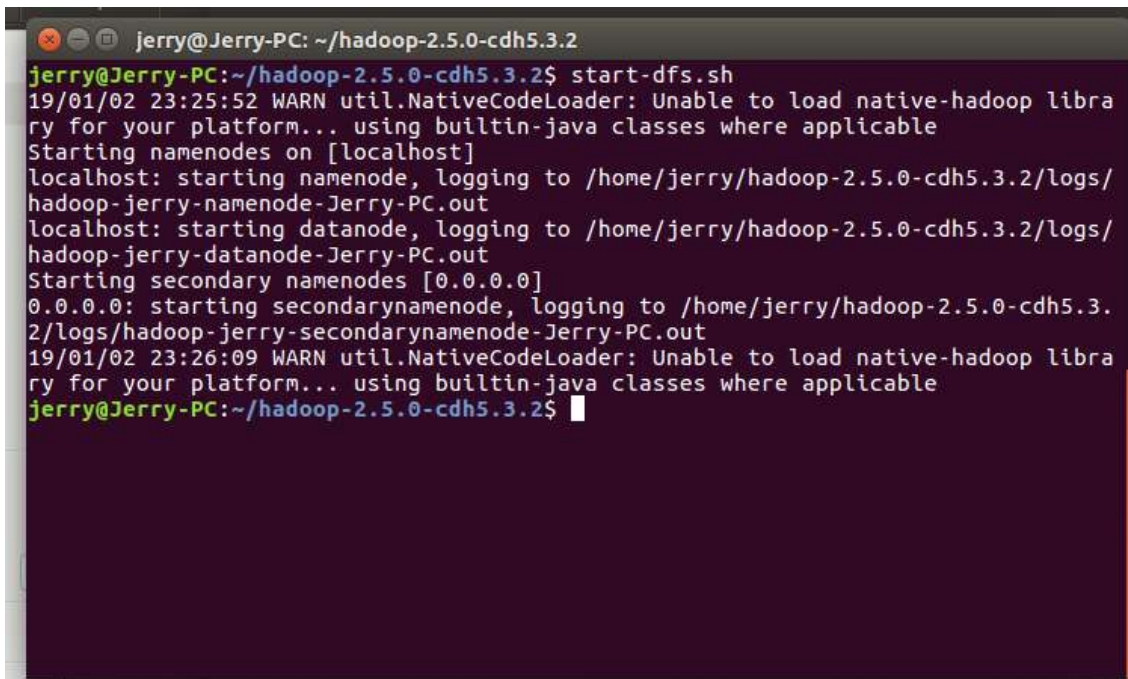
```

}

APPENDIX B

I. Distributed Dataset Storage using Ubuntu terminal

- a. Starting HDFS daemons using start-dfs.sh command



```
jerry@Jerry-PC: ~/hadoop-2.5.0-cdh5.3.2
jerry@Jerry-PC:~/hadoop-2.5.0-cdh5.3.2$ start-dfs.sh
19/01/02 23:25:52 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/jerry/hadoop-2.5.0-cdh5.3.2/logs/
hadoop-jerry-namenode-Jerry-PC.out
localhost: starting datanode, logging to /home/jerry/hadoop-2.5.0-cdh5.3.2/logs/
hadoop-jerry-datanode-Jerry-PC.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/jerry/hadoop-2.5.0-cdh5.3.
2/logs/hadoop-jerry-secondarynamenode-Jerry-PC.out
19/01/02 23:26:09 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
jerry@Jerry-PC:~/hadoop-2.5.0-cdh5.3.2$
```

- b. Initiating MapReduce daemons using start-yarn.sh command

```
jerry@Jerry-PC: ~/hadoop-2.5.0-cdh5.3.2
jerry@Jerry-PC:~/hadoop-2.5.0-cdh5.3.2$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/jerry/hadoop-2.5.0-cdh5.3.2/logs/yarn
-jerry-resourcemanager-Jerry-PC.out
localhost: starting nodemanager, logging to /home/jerry/hadoop-2.5.0-cdh5.3.2/lo
gs/yarn-jerry-nodemanager-Jerry-PC.out
jerry@Jerry-PC:~/hadoop-2.5.0-cdh5.3.2$
```

II. Processing 1.42GB dataset

```
jerry@Jerry-PC: ~/hadoop-2.5.0-cdh5.3.2
19/01/02 23:29:43 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
jerry@Jerry-PC:~/hadoop-2.5.0-cdh5.3.2$ bin/yarn jar ../jars/wcjob9.jar com.df.w
c.WordCount /da_input/dataset1_42Gb.txt /da_output/da_output_1_42gb
19/01/02 23:33:11 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
19/01/02 23:33:12 INFO client.RMPProxy: Connecting to ResourceManager at /0.0.0.0
:8032
19/01/02 23:33:13 WARN mapreduce.JobSubmitter: Hadoop command-line option parsin
g not performed. Implement the Tool interface and execute your application with
ToolRunner to remedy this.
19/01/02 23:33:14 INFO input.FileInputFormat: Total input paths to process : 1
19/01/02 23:33:14 INFO mapreduce.JobSubmitter: number of splits:12
19/01/02 23:33:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_15
46468047669_0001
19/01/02 23:33:15 INFO impl.YarnClientImpl: Submitted application application_15
46468047669_0001
19/01/02 23:33:15 INFO mapreduce.Job: The url to track the job: http://Jerry-PC:
8088/proxy/application_1546468047669_0001/
19/01/02 23:33:15 INFO mapreduce.Job: Running job: job_1546468047669_0001
19/01/02 23:33:25 INFO mapreduce.Job: Job job_1546468047669_0001 running in uber
mode : false
19/01/02 23:33:25 INFO mapreduce.Job: map 0% reduce 0%
```

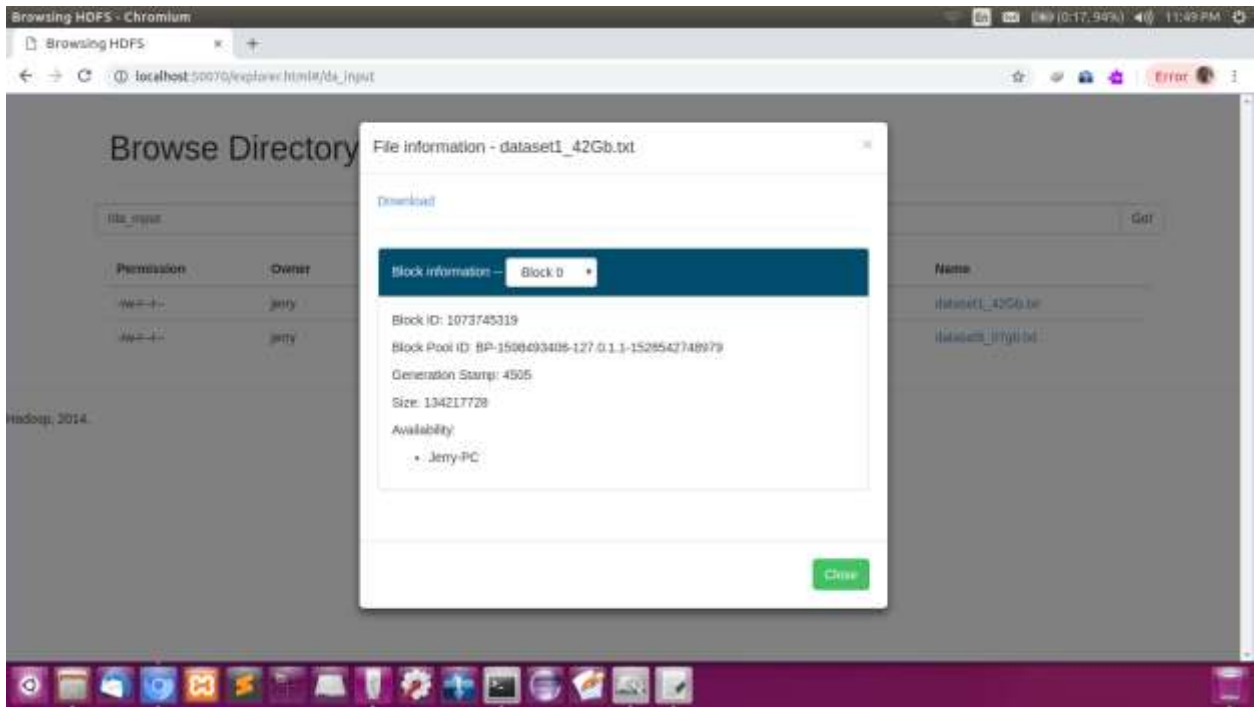
```
jerry@Jerry-PC: ~/hadoop-2.5.0-cdh5.3.2
46468047669_0001
19/01/02 23:33:15 INFO mapreduce.Job: The url to track the job: http://Jerry-PC:
8088/proxy/application_1546468047669_0001/
19/01/02 23:33:15 INFO mapreduce.Job: Running job: job_1546468047669_0001
19/01/02 23:33:25 INFO mapreduce.Job: Job job_1546468047669_0001 running in uber
mode : false
19/01/02 23:33:25 INFO mapreduce.Job: map 0% reduce 0%
19/01/02 23:33:57 INFO mapreduce.Job: map 1% reduce 0%
19/01/02 23:34:05 INFO mapreduce.Job: map 2% reduce 0%
19/01/02 23:34:11 INFO mapreduce.Job: map 3% reduce 0%
19/01/02 23:34:16 INFO mapreduce.Job: map 4% reduce 0%
19/01/02 23:34:20 INFO mapreduce.Job: map 5% reduce 0%
19/01/02 23:34:23 INFO mapreduce.Job: map 6% reduce 0%
19/01/02 23:34:27 INFO mapreduce.Job: map 7% reduce 0%
19/01/02 23:34:30 INFO mapreduce.Job: map 8% reduce 0%
19/01/02 23:34:34 INFO mapreduce.Job: map 9% reduce 0%
19/01/02 23:34:37 INFO mapreduce.Job: map 10% reduce 0%
19/01/02 23:34:40 INFO mapreduce.Job: map 11% reduce 0%
19/01/02 23:34:44 INFO mapreduce.Job: map 12% reduce 0%
19/01/02 23:34:47 INFO mapreduce.Job: map 13% reduce 0%
19/01/02 23:34:50 INFO mapreduce.Job: map 14% reduce 0%
19/01/02 23:34:54 INFO mapreduce.Job: map 15% reduce 0%
19/01/02 23:34:57 INFO mapreduce.Job: map 16% reduce 0%
```

```
jerry@Jerry-PC: ~/hadoop-2.5.0-cdh5.3.2
19/01/02 23:38:28 INFO mapreduce.Job: map 95% reduce 31%
19/01/02 23:38:31 INFO mapreduce.Job: map 96% reduce 31%
19/01/02 23:38:34 INFO mapreduce.Job: map 97% reduce 31%
19/01/02 23:38:35 INFO mapreduce.Job: map 100% reduce 31%
19/01/02 23:38:36 INFO mapreduce.Job: map 100% reduce 100%
19/01/02 23:38:37 INFO mapreduce.Job: Job job_1546468047669_0001 completed succe
ssfully
19/01/02 23:38:38 INFO mapreduce.Job: Counters: 50
  File System Counters
    FILE: Number of bytes read=6
    FILE: Number of bytes written=1361216
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=1528398691
    HDFS: Number of bytes written=3763994041
    HDFS: Number of read operations=63
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=38
  Job Counters
    Killed map tasks=1
    Launched map tasks=13
    Launched reduce tasks=1
    Data-local map tasks=13
```

```
jerry@Jerry-PC: ~/hadoop-2.5.0-cdh5.3.2
Reduce shuffle bytes=72
Reduce input records=0
Reduce output records=0
Spilled Records=0
Shuffled Maps =12
Failed Shuffles=0
Merged Map outputs=12
GC time elapsed (ms)=564710
CPU time spent (ms)=416810
Physical memory (bytes) snapshot=3124555776
Virtual memory (bytes) snapshot=7718252544
Total committed heap usage (bytes)=2331246592
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=1528397335
File Output Format Counters
  Bytes Written=0
jerry@Jerry-PC:~/hadoop-2.5.0-cdh5.3.2$
```

APPENDIX C

I. Sample Block Information



II. Output of Processed 1.42GB Dataset

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--	jerry	supergroup	43.33 MB	1	128 MB	BadRecords-m-00005
-rw-r--	jerry	supergroup	43.36 MB	1	128 MB	BadRecords-m-00006
-rw-r--	jerry	supergroup	42.53 MB	1	128 MB	BadRecords-m-00007
-rw-r--	jerry	supergroup	43.35 MB	1	128 MB	BadRecords-m-00008
-rw-r--	jerry	supergroup	42.54 MB	1	128 MB	BadRecords-m-00009
-rw-r--	jerry	supergroup	43.34 MB	1	128 MB	BadRecords-m-00010
-rw-r--	jerry	supergroup	17.09 MB	1	128 MB	BadRecords-m-00011
-rw-r--	jerry	supergroup	273.46 MB	1	128 MB	ParsedRecords-m-00000
-rw-r--	jerry	supergroup	271.25 MB	1	128 MB	ParsedRecords-m-00001
-rw-r--	jerry	supergroup	273.37 MB	1	128 MB	ParsedRecords-m-00002
-rw-r--	jerry	supergroup	271.5 MB	1	128 MB	ParsedRecords-m-00003
-rw-r--	jerry	supergroup	273.3 MB	1	128 MB	ParsedRecords-m-00004
-rw-r--	jerry	supergroup	271.59 MB	1	128 MB	ParsedRecords-m-00005
-rw-r--	jerry	supergroup	271.25 MB	1	128 MB	ParsedRecords-m-00006
-rw-r--	jerry	supergroup	273.41 MB	1	128 MB	ParsedRecords-m-00007
-rw-r--	jerry	supergroup	271.46 MB	1	128 MB	ParsedRecords-m-00008
-rw-r--	jerry	supergroup	273.33 MB	1	128 MB	ParsedRecords-m-00009

III. Sample Annotated Sensor Record

