Thesis no: MSEE-2016:35

Performance Evaluation of Time series Databases based on Energy Consumption

Sanaboyina Tulasi Priyanka

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author:
Tulasi Priyanka Sanaboyina
E-mail: tulasi.priyanka@gmail.com,
tusa15@student.bth.se

University advisor:
Asst. Prof. Dr. Dragos Ilie
School of Computing

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

The vision of the future Internet of Things is posing new challenges due to gigabytes of data being generated everyday by millions of sensors, actuators, RFID tags, and other devices. As the volume of data is growing dramatically, so is the demand for performance enhancement. When it comes to this big data problem, much attention has been given to cloud computing and virtualization for their almost unlimited resource capacity, flexible resource allocation and management, and distributed processing ability that promise high scalability and availability. On the other hand, the variety of types and nature of data is continuously increasing. Almost without exception, data centers supporting cloud based services are monitored for performance and security and the resulting monitoring data needs to be stored somewhere. Similarly, billions of sensors that are scattered throughout the world are pumping out huge amount of data, which is handled by a database. Typically, the monitoring data consists time series, that is numbers indexed by time. To handle this type of time series data a distributed time series database is needed.

Nowadays, many database systems are available but it is difficult to use them for storing and managing large volumes of time series data. Monitoring large amounts of periodic data would be better done using a database optimized for storing time series data. The traditional and dominant relational database systems have been questioned whether they can still be the best choice for current systems with all the new requirements. Choosing an appropriate database for storing huge amounts of time series data is not trivial as one must take into account different aspects such as manageability, scalability and extensibility. During the last years NoSQL databases have been developed to address the needs of tremendous performance, reliability and horizontal scalability. NoSQL time series databases (TSDBs) have risen to combine valuable NoSQL properties with characteristics of time series data from a variety of use-cases.

In the same way that performance has been central to systems evaluation, energy-efficiency is quickly growing in importance for minimizing IT costs. In this thesis, we compared the performance of two NoSQL distributed time series databases, OpenTSDB and InfluxDB, based on the energy consumed by them in different scenarios, using the same set of machines and the same data. We evaluated the amount of energy consumed by each database on single host and multiple hosts, as the databases compared are distributed time series databases. Individual analysis and comparative analysis is done between the databases. In this report we present the results of this study and the performance of these databases based on energy consumption.

Keywords: Time series Databases, Energy Consumption,

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my supervisor, Asst. Prof. Dragos Ilie for his valuable guidance and encouragement throughout the period of the thesis work. His supervision helped me during my thesis research and composing of the thesis document immensely.

I would also like to thank my thesis committee: Prof. Kurt Tutschku and Asst.Prof. Patrik Arlos including others, for their insightful comments and encouragement.

On the whole, I would like to the Department of Communication Systems for this educational opportunity which has tested and pushed me beyond my abilities.

Furthermore, I would like to thank my parents and friends for their support at every step of my education at Blekinge Tekniska Hogskolan.

Thank you all

Contents

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

IoT	Internet of Things
SQL	Structured Query Language
DBMS	Database Management System
NSIT	Netaji Subhas Institute of Technology
IaaS	Infrastructure as a service
PaaS	Platform as a service
SaaS	Software as a service
MIT	Massachusetts Institute of Technology
API	Application program interface
GUI	Graphical user interface
RDMS	Relational database management system
HTTP	Hyper Text Transfer Protocol
TSDB	Time series Database
TCP	Transmission Control Protocol
CLI	Command Line Interface
JSON	JavaScript Object Notation
UTC	Universal time Coordinated
HDFS	Hadoop Distributed File System
GFS	Google File System
WUI	Web user Interface
QLFU	Queued Least-Frequently-Used
VM	Virtual Machine
RQ	Research Question

1 INTRODUCTION

1.1 Overview

Internet of Things (IoT) is a concept that envisions all objects around us as a part of the Internet and that leverages on the power of networks to create ubiquitous sensor-actuator networks. It refers to a world of physical and virtual objects (things) which are uniquely identified and capable of interacting with each other, with people, and with the environment. IoT coverage is very wide and includes a variety of objects like smartphones, tablets, digital cameras, sensors, etc. [1]. Once all these devices are connected with each other, they enable more and more smart processes and services that support our basic needs, economies, environment and health. IoT allows people and things to be connected at anytime and anyplace, with anything and anyone. Communication among the things is achieved by exchanging the data and information sensed and generated during their interactions. Such enormous number of devices connected to the Internet provides many kinds of services and produces a huge amount of data and information. The types of data transmitted in the Internet of Things are of a huge variety. They can be input by humans or auto-generated, they can also be either discrete, where the number of possible data points is countable, or continuous where the data points are infinite but can be acquired through sampling.

Everyday gigabytes of data which can be text, picture, videos or more is pumped out by systems from different domains such as manufacturing, social media, or cloud computing. According to a study in Digital Universe in 2012, for every two years, the data in the world is getting doubled, and the same study has forecasted that by 2020, the data can be 50 times more than that of 2010. More than 40% of the data described above will be stored in a cloud [2]. This is because of the massive data generation and also due to the number of IoT objects. Besides, all this data is updated in real-time and across multiple nodes.

In business computing, Cloud computing is an emerging model and the systems aim for data computations and procession. A powerful network service such as a super computer can be emulated by the Cloud computing system. To make calculation capability, storage space and software services accessible to all the applications, computing tasks are distributed to a large number of computers in cloud computing technology [3]. Network providers use the cloud computing technology to accord with millions and billions of information within tiny time intervals. Through the emerging cloud computing technology, the independent and personal computing i.e., users that spurt on private ownership of expensive hardware would migrate to a cloud where they can rent computing resources from a cloud provider when needed.

Big Data deals with the volume of the data, the velocity of the data accumulated and a variety of data. Big Data and Cloud are generally bundled together as they provide faster access anywhere, elasticity and scalability. The volumes of data handled in cloud environments coupled with demands for scalability, availability and multitenancy cannot be handled well by existing database architectures. This is the reason why we have witnessed the growing popularity of NoSQL databases for handling data in the cloud.

1.2 Motivation

The rapid growth in the amount of data is mostly related to radical changes in the data types and how data is generated, collected, processed, and stored. With different new sources of data, data tend to be distributed across multiple nodes it is no longer conform to some predefined schema definition. For example, unstructured and semi-structured data make up 90% of the total digital data space, including text messages, log files, blogs and more. [2]. This problem has promoted the creation of new technologies which can handle the data growth while improving system performance. Several database management systems (DBMS) have been developed and characterized to this end. The classic and dominant type, SQL databases among database systems have raised the question whether they fit well with all the periodic data. As a solution for this upcoming problem, a new class of database referred to as NoSQL databases are developed. NoSQL databases store data very differently from the traditional relational database systems. They are meant for data of schema free structure and are claimed to be easily distributed with high scalability and availability. These properties are actually needed to realize the vision behind Internet of Things data. The data that is sampled at a particular time interval is called time series data.

1.3 Problem Statement

The challenge of the IoT (Internet of Things) is not in the functionality of a smart object but in the extreme number of billions or even trillions of smart objects that generate large amounts of data which need storage backend [4]. Billions of sensors that are scattered throughout the world are pumping out a huge amount of data that is to be handled. Different data centers report periodical data that needs to be stored. The big question is how to manage the data system in an efficient and cost-effective way. This can be solved by proper planning in the selection of which database management system either the traditional system or the newly emerged NoSQL system, is needed, to store periodic data. As mentioned before, a variety of databases are currently available, including SQL and NoSQL databases [2].

As the data is periodic in nature time series databases fit the most for this problem. For any storage system, reliability and scalability are most important factors as periodic data needs to be stored for a long time and needs to be the updated periodically. So a distributed time series database is the best fit to store all these large amounts of data that is generated throughout the world. On the other hand, energy-efficient computing combined with environmental concerns is making energy efficiency a prime technological and societal challenge. As the rate of change of energy is power, that implies energy is directly proportional to power. It is important to identify which services or processes consume a large amount of energy when complaining about data centers. However, physical power meters and components with embedded energy sensors are often missing, and they require significant investment and efforts to be deployed posterior in a data center. Additionally, these hardware facilities usually only provide system-level or device-level granularity. Hence, software-based power estimation is becoming an economical alternative in measuring energy consumption. This helps in the identification of devices that consumes considerably more energy. Identified devices can further be optimised to decrease the energy consumption. [5]. As we have a varied number of distributed time series databases that are efficient reliable and scalable it is also important to see which database consumes less energy. For this, we have taken two distributed time series databases OpenTSDB, InfluxDB that are most widely used and their performance is compared in terms of energy consumption during different operations such as read and write.

1.4 Research Questions

RQ1: How much energy is consumed on average by a node of the distributed time series database?

RQ2: How much energy is consumed by the database when synchronization among multiple nodes is considered?

RQ3: What is the estimated energy of a read and write operation, respectively, executed by the database?

1.5 Contribution

The contributions of the thesis:

a) A clear analysis is given about the different behaviours of the distributed time series databases under load conditions.
b) It gives the analysis of the flexibility of the database when different data are processed.
c) It gives a quantitative analysis of energy consumed by each database while storing and accessing data.

A comparative analysis is given between different databases in terms of energy consumption in different scenarios.

1.6 Document Outline

Chapter 1, it provides the motivation for this thesis, the problem at hand, research questions and contribution of this work.

Chapter 2 provides an overview of the background and elucidates the technologies involved in this work.

Chapter 3 deals with previous research work related to this thesis.

Chapter 4 presents the methodology, it illustrates the design and scenarios of the experiment with a detailed account of the experiment setup.

Chapter 5 presents the results of the experiment.

Chapter 6 deals with the analysis of the obtained results.

Chapter 7 includes conclusions derived from the experiments, its results and analysis.

2 BACKGROUND

2.1 Cloud Computing and IoT

2.1.1 Cloud Computing

Cloud computing is the new driver of IT revolution; as new IT services are being developed. There are also changing the ways of access, usage, maintenance and financing services on demand [6].The definition provided for Cloud Computing by the National Institute of Standard and Technologies (NIST) is: ‘‘Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction’’ [7].

The definition provided by Gartner defines is like “style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies”. Cloud computing is any online activity for everyday users of the Internet and computers. Data is been accessed or small software projects are performed from different devices regardless of the on-ramp to the Internet [8]. The data or software applications are not stored on the user's computer, but rather are accessed through the web from any device at any location a person can get web access.

For end users all the applications like obtaining software licenses, updating or upgrading existing software, data synchronization, etc. are available in the Cloud service. Cloud computing means that you don't have to worry about maintaining hardware or purchase new equipment [9]. Scalability, Mobility and Platform independency are the characteristics of Cloud computing [6].

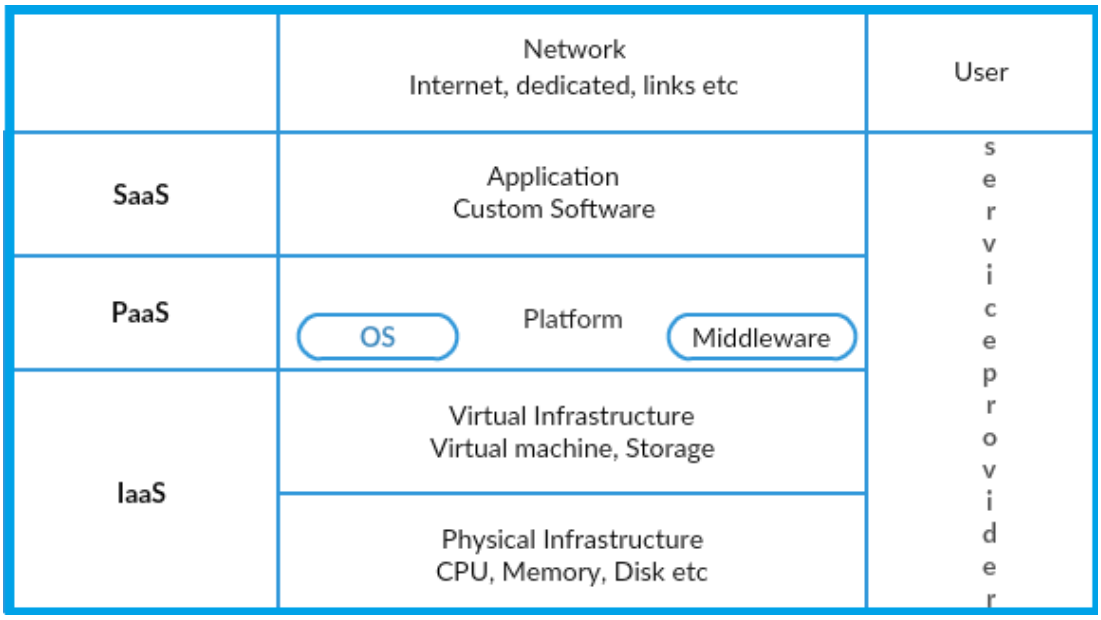

Figure 1: Cloud computing

As depicted in figure 1, there are three types of cloud computing: Infrastructure as a Service (IaaS) is the hardware component with different forms of virtual technology rentals, platform

as a service (PaaS) involves the use of the operating system and development tools in the cloud and software as a service (SaaS) which refers to the use various web-based applications that run and execute on the server [6].

IaaS:

For elimination of initial investment installations effectively in subscribers business, an IaaS service provider invests on infrastructure, deploys and maintains them to offer physical or virtual hardware [6]. For configuring and monitoring resources and providing an opportunity to install independent OS, middleware and custom applications by subscribers, the provided infrastructures are accessed remotely or graphically as a major feature of IaaS cloud computing. Public clouds (e.g. Amazon Elastic Compute Cloud – EC2), virtual private clouds (e.g. T-Systems) and implementation tools (e.g. vCloud, and OpenStack) for private clouds are offered by IaaS Service Providers.

PaaS:

An execution and development environment on top of a cloud infrastructure is offered by PaaS as it provides a wide spectrum of detailed application-level services. A platform for implementing and uploading custom applications by developers is established in PaaS whereas the cost and convolution of configuring, managing, and monitoring cloud infrastructure are eliminated [6]. For custom codes that are developed by customers (e.g. Google App Engine), some PaaS clouds offer a cloud execution environment. For other PasS clouds, a configuration file should be applied before coding, this will permit customers to develop extensions for cloud-based software (e.g. Force).

SaaS:

Cloud-based applications are the highest level of applications that cloud computing provides. For Subscribers the applications like hardware installation, license payments, middleware configurations, and system administrations are eliminated and enhance the acceleration of software installation, configuration, and customization for them. For handling service delivery, user customization, and user scalability the major three specifications are used which are Centralized management, data isolation, and united multi-tenancy [6]. Furthermore, a SaaS application consists of a domain container for wrapping applications from a single supplier within the shared platform and application integration for establishing appropriate communication when needed.

2.1.2 Internet of things

In 1999 concept “Internet of Things” was coined by Kevin Ashton of Massachusetts Institute of Technology (MIT) [10]. Definition of Internet of Things as per Kevin Ashton is as follows “all things are connected to the Internet via sensing devices such as Radio Frequency Identification (RFID) to achieve intelligent identification and management”. From figure 3, it can be explained as in IoT the local environment contains the connected objects and the local pickup points. All these elements communicate through wired technologies (Ethernet, optic fiber, etc.) or wireless links [6] (Bluetooth Low Energy, Wi-Fi, ZigBee, etc.). Smartphones, small computers and other objects are the local pickup points which are optional [10]. To reach the infrastructure by objects which are not that powerful enough (battery, computing power. etc.), these local pickup points act as a gateway for them. Sometimes, direct user interaction with the objects is also possible (an application on a smartphone for example).

As depicted in figure 2, IoT is subdivided into layers or levels where the objects or local pickup points are allowed to communicate with the command servers in the transport level [10]. The processing of the data is allowed by the storage and data mining which generally takes place

in the cloud. Data is accessed by the users, or other systems through APIs or GUIs. It can be seen that only the first level – the local environment – is specific to the IoT, the other three can be found anywhere where a massive amount of data are being generated or treated.

From the above we can define connected object as: "Sensor(s) and/or actuator(s) carrying out a specific function and that are able to communicate with other equipment. It is part of an infrastructure allowing the transport, storage, processing and access to the generated data by users or other systems." Then, IoT can be defined as: "Group of infrastructures interconnecting connected objects and allowing their management, data mining and the access to the data they generate" [10].

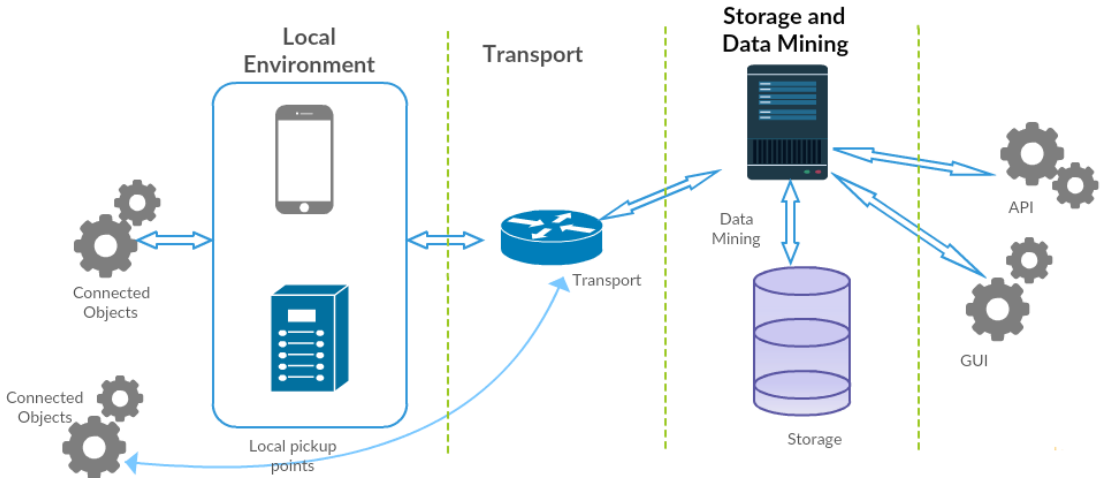

Figure 2: Internet of Things

The broad future vision of IoT is to make the things able to react to physical events with suitable behaviour, to understand and adapt to their environment, to learn from, collaborate with and manage other things, and all these are autonomous with or without direct human intervention [2]. To achieve such a goal, numerous researches have been carried out. The three main concrete visions of the IoT that most of the researches are focusing on are:

- Things-oriented vision: Originally, the IoT started with the development of RFID (Radio Frequency Identification) tagged objects that communicate over the Internet. RFID along with the Electronic Product Code (EPC) global framework is one of the key components of the IoT architecture. However, the vision is not limited to RFID. Many other technologies are involved in the things-vision of IoT. Those in conjunction with RFID are to be the core components that make up the Internet of Things. Applying these technologies, the concept of things has been expanded to be of any kind: from human to electronic devices such as computers, sensors, actuators, phones. In fact, any everyday object might be made smart and become a thing in the network. For example, TVs, vehicles, books, clothes, medicines, or food can be equipped with embedded sensor devices that make them uniquely addressable, be able to collect information, connect to the Internet, and build a network of networks of IoT objects.
- Internet-oriented vision: A focus of the Internet-oriented vision is on the IP for Smart Objects (IPSO) which proposes to use the Internet Protocol to support smart objects connection around the world. As a result, this vision poses the challenge of developing the Internet infrastructure with an IP address space that can accommodate the huge number of connecting things. Another focus of this vision is the development of the Web of Things, in which the Web standards and protocols are used to connect embedded devices installed on everyday objects.
- Semantic-oriented vision: The heterogeneity of IoT things along with the huge number of objects involved impose a significant challenge for the interoperability among them.

Semantic technologies have shown potential for a solution to represent, exchange, integrate, and manage information in a way that conforms with the global nature of the Internet of Things. The idea is to create a standardized description for heterogeneous resources, develop comprehensive shared information models, provide semantic mediators and execution environments, thus accommodating semantic interoperability and integration for data coming from various sources.

2.1.3 Types of data

The scope of the IoT is wide and it provides applicability and profits for users and organizations in a variety of fields. For instance, face recognition to analyse passing shoppers, identify their gender and age range is used by the digital billboards and by this the advertisement content is changed accordingly. Similarly, a smart refrigerator keeps track of food items' availability and expiry date, then autonomously orders new ones if needed. To monitor crop conditions during farming, to control farming equipment's and more works are smartly handled by a small sensor network. Examples of such applications of IoT are countless [2]. With countless applications of IoT, the data generated from all these applications could be a lot more than expected and the types of data transmitted in the Internet of Things can also be of extreme variety The type of data that is transmitted could be either discrete or analogue, input by humans or auto-generated. Generally, IoT data include the following:

- Radio Frequency Identification Data
- Sensor Data
- Multimedia Data
- Positional Data
- Descriptive Data
- Metadata about Objects or Processes and Systems
- Command Data

All the above data types have one thing in common, which is that they can be reported periodically. The type of data that is reported at a particular interval with a respective time index can be called as time series data.

Time series data refers to the composition of metrics and tags. A metric consists of a title and several time-value pairs which are a successive time order arrangement of numerical data. Usually, time series data is enriched by tags. Tags are made up of tag keys and tag values. Both tag keys and tag values are stored as strings and record metadata. The tag set is the different combinations of all the tag key-value pairs. Tags are optional and indexed unlike fields [11]. The field of time series analysis defines different approaches to investigate past data to gather meaningful statistics. The time series forecasting acts as a field of predicting prospective developments based on meaningful analytics [9]. There are four types of components for time series data which are trends, cyclical variation, seasonal variation and irregular variation.

- Secular Trend: A long-term increase or decrease in the data indicates a trend. It does not need to be straight.
- Cyclical variation: The second component of a time series is the cyclical variation that happens when any pattern demonstrating an up and down movement around a given trend is recognized.
- Seasonal Variation: Seasonality happens when the time series displays consistent fluctuations during the same month (or months) every year, or during the same quarter consistently. Seasonality is dependably of a settled and known period.
- Irregular variation: This component is unpredictable. Every time series has some eccentric component that makes it an arbitrary variable. In the forecast, the goal is to

model all the components to the point that the main component that remains unexplained is the random component[4].

Below we show how time series data is constructed:

- Time series data = metrics + tags
- Metric: an arrangement of numerical data in a successive time order.
- Tag: metadata to structurally enrich a metric with additional information.
- Consider the data point: “sys.cpu.user 1234567890 42 host=web01 cpu=0”
- In the following data point, “sys.cpu.user” is the Metric name, “1234567890” is the timestamp, “42 is the metric value, “host=web01” and “cpu=0” are the tags.

2.2 Databases

A database is an organized collection of data. It is a collection of schemas, tables, queries, views and other objects. Large amounts of data are stored in a database such that the data is organized to model aspects of reality in a way that supports processes requiring information. Access to this data is usually provided by a database management system that consists of an integrated set of computer software. This software allows the users to interact with one or more databases and provides access to all of the data contained in the databases. Database management systems are often classified according to the database model that they support. A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized and manipulated. The four different types of data models are Flat Model, Relational model, Hierarchal model and Network model. In flat database system, data is stored in a single row and the values are separated by delimiters such as commas or tabs. Data is physically represented in the text file. They are also called flat file databases. Due to its limitations of single row data, this database is not generally used in software applications. Relational model data base system is data is stored in the form of rows and columns. It is a mathematical model which uses predicate logic and set theory to maintain relations among tables. In hierarchical model, data is organized in tree-like structure where all the elements are linked to one primary record. This type of data model uses one-to-many relationship architectures. The network model is an extension of the hierarchical structure, this model allows many-to-many relationships where data can have multiple parent nodes. Based on these data models we have different types of databases such as Operational databases, End user databases, Centralized databases, Distributed databases, Personal databases and Commercial databases. Based on the Query Language used databases can also be classified as SQL Databases and NoSQL Databases.

In this section firstly CAP theorem is stated which has been used as the paradigm to explore the variety of distributed systems as well as database systems. Thereafter, SQL and NoSQL databases are presented along with the main differences between them. In the end, the chapter describes the main features of Time series databases and gives a clear view of the two databases that are targeted in the performance tests.

Cap theorem ACID vs. Base:

The CAP theorem proposed by Eric Brewer [12] states that all the three characteristics stated below cannot be guaranteed by a shared data system at the same time:

- Consistency: This means that once an update operation is finished, everyone can read that latest version of the data from the database. A system where the readers cannot view the new data right away does not have strong consistency and is referred to as eventual-consistent.

- Availability: This is achieved normally by deploying the database as a cluster of nodes, using replication or partitioning data across multiple nodes so if one node crashes, the other nodes can still continue to work which means the system needs to provide continuous operations.
- Partition tolerance: This means that the system can continue to operate even if a part of it is inaccessible (e.g. due to the network connection, or maintenance purpose). This can be accomplished by redirecting writes and reads to nodes that are still available. This property is meaningless for a system of one single node though, it only works for a cluster [12].

The most traditional Relational Database Management Systems were initially meant to be on a single server and thus focus on Consistency, thus having the so-called ACID properties:

- Atomicity: the transactions are all-or-nothing, which means that the database state changes only when the transaction is fully completed.
- Consistency: Here the consistency is different from that we have already defined in the CAP theorem. It ensures that any transaction brings the database from one stable state to another. In CAP, consistency means the stability of the system i.e. the system stays in a stable state before and after the transaction. If a failure occurs, the system reverts to the previous state. [2].
- Isolation: Without any interference or objection transitions are carried away. It ensures that the concurrent state obtained due to transactions is obtained due to serial transactions
- Durability: This guarantees that committed transactions will not be lost as the database keeps track of all the changes made (in logs) so that the system can recover from an abnormal termination.

Consistency is essential for databases that are used for banking or accounting data. However, there are ones that favour availability and partition tolerance over consistency. For instance, social networks, blogs, wikis, and other large scale websites with high traffic and low-latency requirement focus on availability and partition-tolerance[2]. For these systems, it is hard to achieve ACID approach, and hence BASE approach is more likely to be applied:

- Basic Availability: It indicates that the system does guarantee availability in terms of CAP theorem.
- Soft-state: Due to the eventually consistency model, it indicates that the state of the system may change over time even if there is no input.
- Eventual consistency: During the state where the input is not provided, then the system will become consistent over time.

The system with BASE approach does not have to be strictly available and consistent all the time but is more fault-tolerant. NoSQL databases have been using the CAP theorem as an argument against the traditional ones [2].

2.2.1 SQL Databases

SQL (Structured Query Language) was introduced by IBM in 1970's from then it has become the Standard Query language for Relational Database Management Systems. In the relational model, the data is generally organized into relations where each relation is represented by a table consisting of rows and columns. The list of columns makes up the header of the table whereas the set of rows represents the body of the table. Each column represents an attribute of the data and each row is an entry of data which is a tuple of its attributes. A key is an essential concept in the Relational model that is used to map the data to other relations. Primary key is the most important key of a table which is used to uniquely identify each row in a table.

In order to access a relational database, SQL is used to make queries to the database such as creating, reading, updating and deleting data. To speed up reading operations, creating views and other features for database optimization and maintenance SQL supports indexing mechanism [13]. SQL as their standard Query Language is used by MYSQL, Oracle and SQL Server which are different relational databases. SQL databases follow the ACID rules to ensure the reliability of the data. This is one of the key difference between SQL and NoSQL databases [14].

SQL databases usually support isolated transactions, with two-phase commit and rollback mechanism to achieve the data integrity. The above feature contributes to the processing overhead [15]. The normal sources of processing overhead are:

- Logging: To ensure system durability and consistency, so the system can recover from failures SQL databases write everything twice, once to the database itself and once to the log.
- Locking: Before making a change to a record, a transaction must set a lock on it and the other transactions cannot interfere before the lock is released.
- Latching: For preventing data from unexpected modification a latch can be understood as a "lightweight, short-term lock". However, latches are only maintained during the short period while locks are kept during the entire transaction, the short period is when a data page is moved between the cache and the storage engine.
- Besides, when shared data structures have used the index and buffer management also require significant CPU and I/O operations, especially (e.g., index B-trees, buffer pool). Hence, they also cause processing overhead.

Relational databases which are originally designed to focus on data integrity are nowadays facing challenges of scaling to meet the growing data volume and workload demand [2].

2.2.2 NoSQL Databases

Relational databases have matured very well because of their prolonged existence and are still good for various use cases. Unfortunately for much of the today's software design which contains large data sets and dynamic schemas, relational databases show their age and are not giving a good performance [16]. Similarly, nowadays there is a rapid demand for database technologies in the following aspects such as high concurrent of reading from and writing to the database with low latency, efficient big data storage and access requirements, high scalability and high availability and limited capacity [17]. These changes in requirements along with various other reasons described above led to the development of non-relational databases known as NoSQL databases.

NoSQL can also be interpreted as the abbreviation of NOT ONLY SQL to show the advantage of NoSQL [17]. There is much disagreement on this name as it does not depict the real meaning of non-relational, non-ACID, schema-less databases since SQL is not the obstacle as implied by the term NoSQL. The term "NoSQL" was introduced by Carlo Strozzi in 1998 as a name for his open-source relational database that did not offer a SQL interface [18]. The term was re-introduced in October 2009 by Eric Evans for an event named no:sql(east) organized for the discussion of open source distributed databases [19]. Different from SQL databases, NoSQL databases do not divide data into relations, nor do they use SQL to communicate with the database[2].

2.2.2.1 NoSQL properties

It is likely to put SQL into perspective when it comes to NoSQL definition because the origin for NoSQL movements is to eliminate the weak points of relational databases which are widely

considered as the traditional and popular type of database. NoSQL databases are known to be non-relational, horizontally scalable and distributed. Common characteristics of NoSQL databases are listed below, which show the motivations for the rise of such databases.

Non-relational:

There are various types of NoSQL databases, including document, graph, key value, and column family databases, but the common point is that they are non-relational. A principal engineer at Java toolmaker SpringSource, Jon Travis said "Relational databases give you too much. They force you to twist your object data to fit a RDBMS" [20]. Relational model only fits a portion of data whereas many data need a simpler structure or a flexible one.

In NoSQL databases, there are no limitations on the data structure. More data types apart from the normal primitive types are supported, for example, nested documents or multi-dimensional arrays. Unlike SQL, each record does not necessarily hold the same set of fields, and a common field can even have different types in different records. Hence, NoSQL databases are meant to be schema-free and suitable to store data that is simple, schema-less, or object-oriented [21].

Unstructured data (e.g., email body, multimedia, metadata, journals, or web pages) is more easily and efficiently handled by NoSQL databases. Moreover, when it comes to data of dynamic structure, the benefit of a schema-free data structure also stands out.

Horizontal scalability:

Most of the SQL databases were initially designed to run on a single large server. For operating in a distributed manner, several servers are joined together which is a difficult work for relational databases [22]. The idea of "one size fits it all", however, is not feasible to fulfil current demand. Portioning of data across multiple machines is a suitable idea. Unlike SQL databases, most NoSQL databases are not relying much on hardware capacity and are able to scale well horizontally. Without causing any interruption in system operation, cluster nodes can be added or removed in NoSQL databases. This provides higher availability and distributed parallel processing power that increases performance, especially for systems with high traffic. If the concept of broken glass, you can get the concept of Sharding – breaking your database down into smaller chunks called "shards" and spreading those across a number of distributed servers. Many time series databases can auto-shared data over multiple servers and keep the data load balanced among them, thus distributing query load over multiple servers [2].

Availability over Consistency:

One main characteristic of SQL databases is that they conform to ACID rules (Section 2.2), which mainly focus on consistency. Many NoSQL databases have dropped ACID and adopted BASE for higher availability and performance. Applications used for bank transactions, for example, require high reliability and therefore, consistency is vital for each data item. Social network applications such as Facebook have a priority to serve millions of users at the same time with the lowest possible latency which do not require such high data integrity. One method to reduce query response time for database systems is to replicate data over multiple servers, thus distributing the load of reads on the database. Once a data is written to the master server, that data will be copied to the other slave servers. An ACID system will have to lock all other threads that are trying to access the same record. This is not an easy job for a cluster of machines, and will lengthen the delayed time. BASE systems will still allow queries even though the data may not be the latest. Hence, it can be said that NoSQL databases prefer to drop the expense for data integrity to trade for better performance, when integrity is not critical [2].

2.2.2.2 NoSQL categories

NoSQL databases can be classified as follows [19]:

- Key-Value stores
- Document stores
- Column Family stores
- Graph databases
- Multi model Databases
- Object Databases
- Grid & Cloud Database Solutions
- XML Databases
- Multidimensional Databases
- Multi value Databases
- Event Sourcing
- Time Series / Streaming Databases

Key-Value stores:

Key-Value Databases have a very simple data model where data is organized as an associative array of entries consisting of key-value pairs. Each key is unique and is used to retrieve the values associated with it. These databases can be visualized as relational databases having multiple rows and only two columns- key and value. Key-based lookups result in shorter query execution time. Also, since values can be anything like objects, hashes etc. it results in a flexible and schema-less model appropriate for today's unstructured data. Key-value databases are highly suitable for applications where schema is prone to evolution. Unlike keys, there is no limit on length of value to be stored. Most key-value stores favour high scalability over consistency and therefore most of them also omit rich ad-hoc querying and analytics features for example join and aggregate operations [16].

Document stores

Document Databases stores documents as data. Documents are grouped together in form of collections and different documents can have different fields. These databases are flexible in nature as any number of fields can be added to the documents without wasting space by adding same empty fields to the other documents in a collection. Compared to relational databases, collections correspond to tables and documents to records. But there is one big difference, in relational databases, every record in a table have the same number of fields, while documents in a collection can have completely different fields. Documents are addressed in the database via a unique key that represents that document. Document-oriented databases are one of the categories of NoSQL databases that are appropriate for web applications which involve storage of semi-structured data and execution of dynamic queries [16].

Column Family stores:

Column-oriented or Wide-table data stores are designed to address following three areas: the huge number of columns, sparse nature of data and frequent changes in the schema. In relational databases, row elements are stored contiguously but in column-oriented databases, column elements are stored contiguously. This change in storage design results in better performance for some operations like aggregations, support for ad-hoc and dynamic query etc. These databases deal with only a few specific columns. Hence, these are best suited for analytical purposes. For each column, row-oriented storage design deals with multiple data types and limitless range of values, thus making compression less efficient overall [16].

Graph databases:

Graph databases model the database as a network structure containing nodes and edges. Nodes may also contain properties that describes the real data contained within each object. Similarly, edges, connecting nodes to express relationship amongst them may also have their own properties. A relationship connects two nodes and is identified by their names and can be traversed in both the directions and may be directed where direction adds meaning to the relationship. Comparing with Entity-Relational Model(ER Model), a node corresponds to an entity, property of a node to an attribute and relationship between entities to relationship between nodes [16].

2.3 Time Series Databases

For software with complex logic or business rules and high transaction volume for time series data, traditional relational database management systems may not be practical. A time series database is a software system that is optimized for handling time series data, arrays of numbers indexed by time. It can be also said that for a particular object the list of changing values sampled at a particular time interval is called a data sequence. A set of data sequences stored in a database is called a time series database. Time series databases make it possible to predict future values of an object by analysing its past values.

It is not easy to store data whose nature is unpredictable, so a time series database is profitable in this case. At large scales, time-based queries can be implemented as large, contiguous read-write operations that are extremely effective if the information is stored appropriately in a time series database. In addition, a non-relational time series database (TSDB) in a NoSQL system can be expected to give adequate scalability for large amounts of information.[4]. There is a lot of periodic or time series data positioned in industries, but these industries are still placing the data in relational databases. There are three main objectives related to TSDBs in this thesis. The first is to make a platform to stream time series data into a distributed time series database. The second is to focus on the energy consumption of the formed TSDB cluster. The third is to analyse and compare the energy consumed by each distributed time series databases mentioned below.

2.3.1 InfluxDB

InfluxDB is a time series database built from the ground up to handle high write and query loads. For any use case involving large amounts of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics InfluxDB is meant to be used as a backing store [11].

Here are some of the key features that InfluxDB currently supports:

- It has a customised high performance data store written specifically for time series data and has high import speed and data compression functionality.
- It is written entirely in Go and it compiles into a single binary with no external dependencies.
- It has simple and high performing write and query HTTP(S) APIs.
- It supports plugins for other data ingestion protocols such as Graphite, collected, and OpenTSDB.
- Relay gives high availability support.
- To easily query aggregated data, it supports SQL-like query language.
- For fast and efficient queries, Tags allow series to be indexed.
- Retention policies efficiently auto-expire stale data.

- Automatically computation of aggregate data is done by Continuous Queries to make frequent queries more efficient.
- Built in web admin interface.

There are many ways to write data into InfluxDB including the command line interface, client libraries and plugins for common data formats such as Graphite. New measurements, tags, and fields can be added at any time. If data is written with a different type than previously used InfluxDB will reject those data [11].

The HTTP API is the primary means for querying data in InfluxDB, alternative ways are the command line interface and client libraries. InfluxDB returns JSON. The results of our query appear in the "results" array. If an error occurs, InfluxDB sets an "error" key with an explanation of the error and timestamp [11]. Some of the key concepts and common terminology that is needed to work with InfluxDB are Field key, Field set, Field value, Measurement, Point, Retention Policy, Series, Tag key, Tag set, Tag value, Timestamp.

Fields are made up of field keys and field values. Field keys are strings and they store metadata. The collection of field-key and field-value pairs make up a field set. Field values are the data we want to store. They can be strings, floats, integers, or Booleans, and, because InfluxDB is a time series database, a field value is always associated with a timestamp. The root thing for everything that we do in InfluxDB deals with the factor Time. All data in InfluxDB have a separate column for the time. Time stores timestamps and the timestamp shows the date and time, in RFC3339 UTC, associated with particular data [11].

Tags are made up of tag keys and tag values. Both tag keys and tag values are stored as strings and record metadata. The tag set is the different combinations of all the tag key-value pairs. Tags are optional and indexed unlike fields [11].

The measurement acts as a container for tags, fields, and the time column, and the measurement name is the description of the data that are stored in the associated fields. Measurement names are strings, and, when compared to SQL a measurement is conceptually similar to a table. A single measurement can belong to different retention policies (RPs). The part of Influx DB's data structure that describes for how long InfluxDB keeps data (duration), how many copies of those data are stored in the cluster (replication factor), and the time range covered by shared groups (shared group duration). RPs are unique per database and along with the measurement and tag set define a series. When you create a database, InfluxDB automatically creates a retention policy called default with an infinite duration, a replication factor set to one, and a shard group duration set to seven days. The attribute of the retention policy that determines how many copies of the data are stored in the cluster. InfluxDB replicates data across N data nodes, where N is the replication factor. A point is a field set in the same series with the same timestamp. An InfluxDB database is similar to traditional relational databases and serves as a logical container for users, retention policies, continuous queries, and time series data. Databases can have several users, continuous queries, retention policies, and measurements. InfluxDB is a schema-less database which means it's easy to add new measurements, tags, and fields at any time [11].

2.3.2 OpenTSDB

OpenTSDB is an open source, distributed time series database designed to monitor large clusters of commodity machines at an unprecedented level of granularity [23]. It allows operation teams to keep track of all the metrics exposed by operating systems, applications and network equipment and makes the data easily accessible. We have chosen OpenTSDB because it is open-source, scalable, and interacts with another open-source distributed

database, HBase [24]. It retains time series for a configurable amount of time (defaults to forever), it creates custom graphs on the fly. The OpenTSDB secret ingredient that helps to increases its reliability, scalability and efficiency is asynchbase. It is a fully asynchronous, non-blocking HBase [24] client, written from the ground up to be thread-safe for server apps. It has far fewer threads and far less lock contention; it uses less memory and provides more throughput especially for write-heavy workloads. It is based on HDFS [25] and HBase [24] and provides a set of command line utilities used to manage the database. OpenTSDB runs on HDFS, that is the file system used in OpenTSDB to store large datasets.

Hadoop:

Hadoop is an open source programming model that provides both distributed storage and computational capabilities across a cluster of computers using simple programming model. Being developed mainly by Yahoo, it is now an Apache project. It is mostly inspired by Google published a paper that describes its novel distributed filesystem, the Google File System (GFS), and MapReduce. Big companies like Yahoo, Facebook, Cloudera and Amazon are currently using Hadoop. Hadoop is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Hadoop library itself is designed to detect and handle failures at the application layer instead of relying on hardware to deliver high-availability [25].

Hadoop platform generally consist of

- Hadoop common
- MapReduce and
- Hadoop Distributed File System (HDFS)

HDFS:

HDFS is a distributed, scalable, a portable file system written in Java for the Hadoop framework. It is a model based on Google File System (GFS) paper [25] [4]. A distributed file system (DFS) is the storage of a computer in a cluster of one unified file system. When file is stored on the DFS, it is partitioned into blocks and each block is replicated across the clusters. It provides a measure of fault-tolerance. HDFS is designed to store big data in terabytes and stream the min an efficient way. It is inspired by the Google file system (GFS). As shown in figure 3, each node in a Hadoop has a single NameNode and a cluster of Data Nodes to form the HDFS cluster. A Hadoop cluster will include master and multiple worker nodes. The master node consists of a JobTracker, TaskTracker, NameNode and Data Node. Clients use RPC to communicate each other. HDFS stores large files (ideally 64MB), across the clusters. It mainly separates file's metadata and application data. Metadata is stored in master (NameNode) and application data is stored in workers (Data Node) [25].

- **NameNode**: The NameNode is the master node. The filesystem namespace tree and map the location of all blocks to the Data Nodes are managed by the NameNode. When a client wants to read a file located in the system, it first contacts NameNode for the location of data block, which consists of different files and comprising that single file and then read block contents from the Data Node closer to the client. The job of the NameNode is to tell where to find a specific data block which stores the required file. Similarly, when the client wants to write data, it first queries NameNode to nominate a suite of three Data Nodes to host block replicas. The client then writes data to the Data Nodes in a pipeline fashion. The replication number three is by default. It can be changed to any numbers. For HBase NameNode is also known as Region Server [25] [4].

- **Data Node**: Data Nodes are the worker's node. They store real application data. They have to periodically send report the NameNode about the information on which blocks they are storing the data enabling the NameNode to update Meta data [25][4].

- **Secondary NameNode**: The Secondary NameNode is an assistant to NameNode for monitoring the state of the cluster's file system. Its main task is to take snapshots of the HDFS metadata from the NameNode memory structures. By doing this, it helps in preventing filesystem corruption and reducing loss of data, and thus over comes failures [25].

- **Job Tracker**: Job tracker accepts jobs from client and submit the jobs in clusters. It also helps to pipeline and distribute job across the cluster.

- **TaskTracker**: It maintains map and reduces task in Data Node.

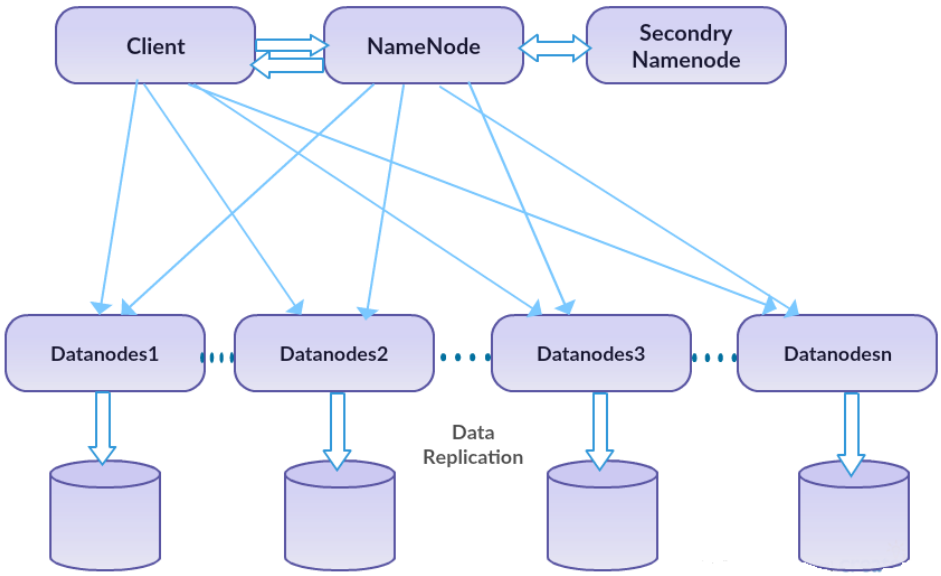

Figure 3: HDFS

OpenTSDB consists of a Time Series Daemon (TSD) as well as a set of command line utilities. Interaction with OpenTSDB is primarily achieved by running one or more of the TSDs. Each TSD is independent. There is no master, no shared state so you can run as many TSDs as required to handle any load you throw at it. Each TSD uses the open source database HBase to store and retrieve time-series data. The HBase schema is highly optimized for fast aggregations of similar time series to minimize storage space. Users of the TSD never need to access HBase directly. You can communicate with the TSD via a simple telnet-style protocol, an HTTP API or a simple built-in GUI. All communications happen on the same port (the TSD figures out the protocol of the client by looking at the first few bytes it receives).

HBase:

HBase is an Apache open source project implementation of Google's BigTable that provides Big Table storage capabilities for Hadoop. Big Table is designed to handle massive workloads at consistent low latency and high throughput, so it's a great choice for both operational and analytical applications, including IoT, user analytics, and financial data analysis. HBase is a distributed column-oriented and NoSQL database built on top of HDFS. One of the biggest utility is the one that is able to combine real-time HBase queries with batch MapReduce Hadoop jobs, using HDFS as a shared storage platform. HBase is extensively used by big companies like Facebook [25], Firefox and others. HBase can be efficient to use if we have

millions or billions of rows. All rows in HBase real ways sorted lexicographically by their row key. In lexicographical sorting, each key is compared on a binary level, byte by byte, from left to right. HBase provides java API for client interaction.

Data are logically organized into tables, rows and columns. Columns in HBase can have multiple versions of the same row key. The data model is similar to that of Big Table. Data are replicated across a number of nodes. Each table must have an element defined as a primary key, and all access attempts to HBase tables must use this primary key. A typical HBase cluster has one active master, one or several backup masters, and a list of regional servers [25].

- **HBaseMaster:** The HBaseMaster is responsible for assigning regions to HRegionServers. The first region is the ROOT region, which contains all the META regions to be assigned. It also monitors the health of HRegionServers and, if it detects a failure in HRegionServer recover it using replicated data [25]. Furthermore, the HBaseMaster is responsible for maintenance of the table, performing such tasks as on-/off-lining of tables and changes to table schema - adding and removing column families, etc.

- **HRegionServer:** The HRegionServer serves client read and write requests. It interacts with the HBaseMaster to obtain a list of regions to serve and to inform the master that it is working.

- **HBase Client**: The HBase client is responsible for investigating HRegionServers that serve the specific row range of interest. On interaction, the HBase client communicates with the HBaseMaster to find the location of the ROOT region.

Thus OpenTSDB is built on top of HBase, which allows us to collect thousands and thousands of metrics from thousands of hosts and applications, at a high rate. All the data is stored in HBase, and the simplified web user interface (WUI), enables users to query various metrics in a real time. OpenTSDB generally creates two special tables in HBase: tsdb and tsdb-uid. Tsdb is the massive table where all the OpenTSDB data points are stored by default. This is to take advantage of HBase's ordering and region distribution. All values are stored in the t column family. Tsdb-uid table stores UID mappings, both forward and reverse. Two columns exist, one named name that maps a UID to a string and another id mapping strings to UIDs. Each row in the column family will have at least one of three columns with mapping values. [26].

A metric ID is located at the start of the row key if a new set of busy metric are created, all writes for those metric will be on the same server until the region splits. With random ID generation enabled, the new metrics will be distributed across the key space and likely to wind up in different regions on different servers. OpenTSDB schema promotes the metric ID into the row key, forming the following structure:

<metric-id><base-timestamp>...

OpenTSDB does not hit the disk for every query, it has its own query cache called Varnish [25]. OpenTSDB also uses HBase caching mechanism called the Block Cache which provides quick access for fetch data point. OpenTSDB provides Java API and HTTP API (JSON). OpenTSDB is built using Java library and Asynchronous [25].

Row key is a combination of metric-id is 3 bytes, base-timestamp is 4 bytes, tag key is 3 byte and tag value is 3 bytes. The column qualifier is of the value 2 bytes. The first 12 bits are used to store an integer which is a delta in seconds from the timestamp in the row key, and the remaining 4 bits are flags. In 4 bit flag, the first bit indicates, the value is an integer value or a floating point value, the remaining 3 bits aren't really used at this time at version 1.0 but they

are to be used for variable-length encoding in the future. The value in the cell is 8 bytes [25] [26].

2.4 Power consumption

2.4.1 Data Centers

Modern data centers are continuously expanding as they attempt to accommodate the surging scientific and enterprise demand for computing resources [27]. The proliferation of Cloud computing has resulted in the establishment of large-scale data centers around the world containing thousands of compute nodes. Driven by this fast-paced demand, the cloud computing paradigm has emerged as a key to leveraging virtualization technologies and to enable a more efficient resource management. However, Cloud data centers consume huge amounts of electrical energy resulting in high operating costs [28]. In this context, energy consumption becomes a critical concern for large-scale datacenters, as well as for the growing cloud infrastructures they host. In the same way that performance has been central to systems evaluation (e.g., measuring completed tasks per unit of time, or Queries per Second), energy-efficiency (e.g., completed tasks per unit of energy, or Queries per Joule) is quickly growing in importance for minimizing IT costs. Most recent work in improving energy efficiency in large data centers is either hardware/platform oriented or workload-management oriented. In the database software field, there have only been a few preliminary studies in this direction [29]. Energy management has now become a key issue for servers and data center operations, focusing on the reduction of all energy-related costs, including capital, operating expenses, and environmental impacts [30]. Storing and managing large amounts of data are recognized as extremely energy inefficient[31].

Our aim in this thesis is to better understand the energy characteristics of database systems on modern hardware. The focus of this report is first assessing different databases and measure the energy consumption of each in different scenarios. We focus on the energy consumed by the database on a single node in a scale-out (shared-nothing, that is when database is not broken into chunks called shards) architecture to clusters (Auto-Sharding is used).

2.4.2 Power API

Energy-efficient computing is becoming increasingly important. Among the reasons, one can mention the massive consumption of large data centers, estimated to account for about 2% of global greenhouse gas and some of which consume as much as 180,000 homes [5]. This trend, combined with environmental concerns makes energy efficiency a prime technological and societal challenge.

Researchers and operators have been proposing solutions to increase energy efficiency at all levels, from application to runtime and to hardware. As surveyed by Org´erie et al. [5], examples include methods for energy-based task scheduling, energy-efficient software, dynamic frequency and voltage scaling, and energy-aware workload consolidation using virtualization. Virtualization offers environment and performance isolation and, hence, is the basis for many data center and cloud management frameworks. In order to improve their energy efficiency, such cloud management frameworks need to know the resource requirements of the running entities. For data center providers and users, it is particularly useful to identify which applications are the largest power consumers. However, physical power meters and components with embedded energy sensors are often missing, and they require significant investment and efforts to be deployed some posterior in a data center.

Additionally, these hardware facilities usually only provide system-level or device-level granularity. Hence, software-based power estimation is becoming an economical alternative [36]. Power estimation is relatively accurate when one has full control over the underlying hardware and detailed knowledge of its properties. It typically works by sampling the activity of applications and measuring the power consumption of the whole system using hardware specific probes.

PowerAPI is a tool to quantify this energy consumption, by providing an application programming interface (API) that monitors, in real-time, the energy consumed at the granularity of a system process [32]. PowerAPI does not require any external device to measure energy consumption. This is a purely software approach where the estimation is based on energy analytical models that characterize the consumption of various hardware components (e.g. CPU, memory, disk). PowerAPI is based on a highly modular architecture where each module represents a measurement unit for a specific hardware component.

One objective of PowerAPI is to provide a simple and efficient way to estimate the energy consumption of a given process. (1) Simple, because API is close to the user requirements. (2) Efficient, because the library is an actor-based framework in which the user builds the library by choosing modules to consider for the user's particular requirements. PowerAPI is thus limited to the user's needs, avoiding any extra computational cost [5] [32]. BITWATTS, a middleware solution to estimate the power consumption of software processes running in virtualized environments. Bitwatts is an extension of PowerAPI. It is a modular framework that can accommodate different power models (including running average power limit (RAPL) probes and power meters). It provides trustworthy power estimation within a few percent of actual measures when configured with the appropriate power model for the underlying hardware [27].

3 RELATED WORK

A large amount of monitored sensor data is obtained from industries to optimize industrial processes. Previous systems are built in such a way that they operate in closed, plant side IT infrastructures without horizontal scalability. Cloud technologies can also be used to enable higher scalability but their maturity for industrial applications with high responsiveness and robustness is not well understood. In paper [33], scalability and robustness of three open source time series databases namely OpenTSDB, KairosDB, Databus are compared for cloud-native monitoring systems. The cloud infrastructures with up to 36 nodes with workloads for realistic industrial applications have been considered. It is evident from the paper that KairosDB fulfils their initial hypothesis (linear scalability, support for industrial workloads, workload independence, resiliency and read/write independence) concerning the scalability and robustness.

The paper[34] proposes a novel mechanism for scalable storage and real-time processing of monitoring data. In this mechanism, two major cycles in the life cycle of monitoring data are considered. The two stages are short time processing and long-time archiving. In this paper data is stored in four different databases namely HBase, OpenTSDB, NFdump1, NFdump2. A comparison is carried out on the queries performance evaluation for these databases. In this paper, HBase outperforms OpenTSDB by an average factor of 87 and NFD1 by an average factor of 4472. Its performance is not comparable with NFD2 since it has a limited dataset that is NFD2 processes only a subset of the dataset with a time constraint. Finally, it concludes that long time queries are performed by MapReduce jobs and short time queries are executed through available scanning APIs.

Paper[35], introduces a new index structure called hierarchal index tree where each time series of database is divided into several segments with same length, each segment of which corresponds to a cell-tree similar to R*-Tree. R-tree is a balanced tree, which uses the minimum bounding rectangle to describe its nodes. Its main problem is that there are a huge number of overlapped tree nodes, which means that there are too many paths to complete the retrieval efficiently. Relevant insertion and retrieval algorithms are implemented for different time series data. The paper proves that there are no false dismissals for the algorithms and the hierarchal index tree is superior to the traditional retrieval methods when the dimension of time series is high. Generally, when Internet of things data is considered, a big question is how to manage the data system in an efficient and cost effective way. This depends on a proper planning on which DBMS (Database Management System) is used to store the concerned data and how it is configured to provide adequate performance.

This paper [2] focuses on evaluating and comparing NoSQL databases against SQL databases in performance, usage and complexity. It also focuses on different types of IoT data, mainly sensor readings and multimedia data. In this paper, four popular databases are considered. They are MySQL, MongoDB, CouchDB and Redis, with the main focus of comparison on MongoDB and MySQL. The sensor scale data showed good results for MongoDB database. The paper concludes that scalability is the actual key point that can potentially make NoSQL win over SQL databases.

This paper[25] focuses on the implementation of a tool for analyzing large time-series data. It describes a way to analyse the data stored by OpenTSDB. OpenTSDB is an open source distributed and scalable time series database. It has become a challenge for statisticians and data scientists to analyse such massive data sets with the same level of comprehensive details as is possible for smaller analyses. Currently, available tools for time-series analysis are time and memory consuming. Moreover, no single tool exists that specializes in providing an efficient implementation of analyzing time-series data through MapReduce programming model at massive scale. To address the above issues, the paper has designed an efficient and

distributed computing framework - R2Time. R2Time integrates R open source project for statistical computing and visualization with the OpenTSDB [1] and RHIPE [2] based on the MapReduce framework for the distributed processing of large data sets across a cluster. It creates the programming environment by integrating R and HBase for the data scientists. It also describes the architecture of R2Time framework. The usefulness of this framework is verified by the performance analysis based on carefully chosen types of statistical analysis for time-series data.

The frequency of circumstances in which data are being gathered as time series is expanding, as are the requirement for reliable, high-performance time series databases. It is observed that it is not a simple issue of what data to save, but rather when considering what data to maintain, a time series database is profitable. At extensive scales, time-based queries can be implemented as large, contiguous scans that are extremely effective if the information is stored appropriately in a time series database. In addition, if the amount of information is vast, a non-relational time series database (TSDB) in a NoSQL system can be expected to give adequate scalability. There are many time-series data deployed in industries that are still placing the data in relational databases. This paper [4], focuses on three main objectives. The first is finding an efficient method to migrate data into a time-series database. The second is to make a platform to stream time series data from a TSDB into a cluster-computing framework. The third is to use machine-learning approaches for analysis, modelling and forecasting of data.

This paper [23] proposes a novel monitoring architecture that, by combining a hierarchical approach with decentralized monitors, addresses the following challenge. Data centers supporting cloud-based services are characterized by a huge number of hardware and software resources often cooperating in complex and unpredictable ways. Understanding the state of these systems for reasons of management and service level agreement requeries scalable monitoring architectures that should gather and evaluate continuously large flows in almost real-time periods. In this context, fully centralized systems do not scale to the required number of flows, while pure peer-to-peer architectures cannot provide a global view of the system state. The paper evaluates the monitoring architecture for computational units of gathering and evaluation in real contexts that demonstrate the scalability potential of the proposed system.

Regardless of whether data is stored in a cluster, grid, or cloud, data management is being recognized as a significant bottleneck. Computing elements can be located far away from the data storage elements. The energy efficiency of the data centers storing this data is one of the biggest issues in data intensive computing. In order to address such issues, the paper [31] proposed a new method for designing and analyzing a series of energy efficient data aware strategies involving data replication and CPU scheduling. In this paper, a new strategy is demonstrated for data replication, called Queued Least-Frequently-Used (QLFU), and its performance is analyzed to determine if it is an energy efficient strategy or not. The paper also focuses on the benefits of using a data aware CPU scheduling strategy, called data backfilling, which uses job pre-emption in order to maximize CPU usage and allows for longer periods of suspension time to save energy. The paper investigates the performance of QLFU and existing replica strategies on a small green cluster to study the running time and power consumption of the strategies with and without data backfilling. Results from this study have demonstrated that energy efficient data management can reduce energy consumption without negatively impacting response time.

In contrast with the previous studies, where the performance played an important role in evaluating the Database Systems, this paper [32] focus on the importance of energy efficiency. The paper studies the energy efficiency in hardware/platform oriented and also in work load management oriented. This paper [30] focuses firstly on how to assess the database server and then explore ways to improve the energy efficiency of a single-machine instance of a database server in a scale-out (shared-nothing) architecture, with standard server-grade hardware components, running a wide spectrum of data management tasks. It focuses on understanding

the power performance trade-off for a single database node in a scale out (shared nothing) architecture. The paper presents several experiments to measure the power of systems components from idle state to fully utilize. It provides a good description of the power profile usage of the different hardware components in one configuration of an 8 core (dual CPU) test machine in the context of database operations. From the workload-management oriented, the paper presents the total CPU power consumption for several power use profiles of database operators, in different configurations (varying the number of cores used from 1 to 8, CPU frequency, storage system). The authors used a high performance, open–source database storage engine that supports both columns-oriented and row–oriented database scans. The authors of the paper, furthermore, used two different DBMSs: one open–source system (PostgreSQL) and one commercial (System - X).

In this paper [27] the authors explore the energy consumption patterns of Infrastructure-as-a-Service cloud environments under various synthetic and real application workloads. For each scenario, they have investigated the power overhead triggered by different types of virtual machines, the impact of the virtual cluster size on the energy-efficiency of the hosting infrastructure and the trade-off between performance and energy consumption of MapReduce virtual clusters through typical cloud applications. The main contributions of this paper can be stated in two ways. First, an evaluation of two well-known IaaS cloud platforms, Apache CloudStack and OpenNebula with respect to energy consumption is proposed. Second, an empirical study is conducted to discover the impact of virtual cluster management on the cloud power profile. The effects of executing applications in customized virtual clusters both in terms of application performance and energy usage are investigated. This paper contemplated mainly on providing users with a basis and a set of guidelines for making energy-aware decisions when selecting specific cloud infrastructures and VM configurations for typical cloud workloads. The results show how different work load types and configuration decisions affect the energy profile of each cloud. By considering the energy consumption of the entire cloud, the evaluations provided valuable insights on cloud computing potential to save energy.

Power estimation of software processes provides critical indicators to drive scheduling or power capping heuristics. State-of-the-art solutions can perform coarse-grained power estimation in virtualized environments, typically treating virtual machines (VMs) as a black box. Yet, VM-based systems are nowadays commonly used to host multiple applications for cost savings and better use of energy by sharing common resources and assets. In this paper [5], a fine-grained monitoring middleware providing real-time and accurate power estimation of software processes running at any level of virtualization in a system. In particular, the solution provided automatically learns an application-agnostic power model, which can be used to estimate the power consumption of applications. The middleware implementation in this paper, named BITWATTS, builds on a distributed actor implementation to collect process usage and infer fine-grained power consumption without imposing any hardware investment (e.g., power meters). BITWATTS instances use high-throughput communication channels to spread the power consumption across the VM levels and between machines. The outcomes of the experiments in this paper are based on CPU- and memory-intensive benchmarks running on different hardware setups which demonstrate that BITWATTS scales both in number of monitored processes and virtualization levels. This non-invasive monitoring solution therefore paves the way for scalable energy accounting that takes into account the dynamic nature of virtualized environments.

This paper [36] argues that hardware-only approaches are only part of the solution, and that data management software will be key in optimizing for energy efficiency. The problems arising from growing energy use in data centers and the trends that point to an increasing set of opportunities for software-level optimizations are considered. Using two simple experiments, the authors illustrated the potential of such optimizations, and, motivated by these examples, they have discussed the general approaches for reducing energy waste. Lastly, they mentioned the existing places within database systems that are promising for energy-

efficiency optimizations. They urge the data management systems community to shift focus from performance-oriented research to energy-efficient computing.

4 METHODOLOGY

4.1 Experiment Testbed

An Environment which has one experiment testbed is implemented for this thesis. The two databases i.e., InfluxDB, OpenTSDB are tested on the same testbed individually. The main aim of the thesis is to analyse and compare the energy consumed by each database under same hardware conditions. The testbed is a representation of the production server cluster, where OpenStack is deployed using the tool Mirantis Fuel. The production server cluster consists of three nodes as shown in figure 4. To simplify the installation of Mirantis OpenStack Fuel tool is being used. The operating system being used in the testbed is Ubuntu.

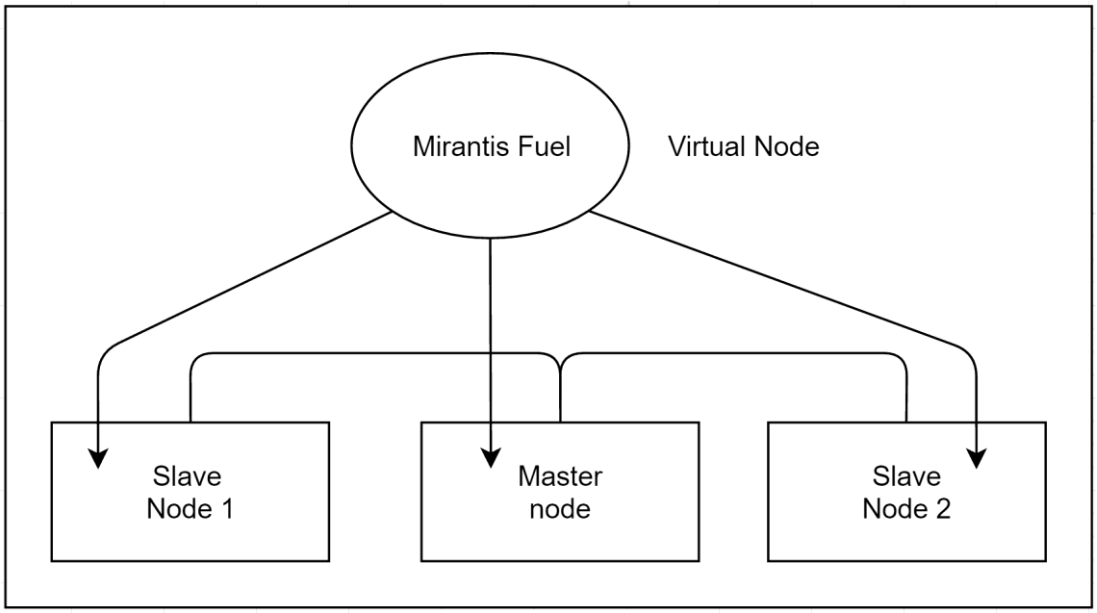

Figure 4: Test-bed representation

4.2 Energy Evaluation Technique

To evaluate the energy consumed by the database on each node, the tool Bitwatts, which is an extension of PowerAPI is used. PowerAPI provides an application programming interface that monitors, in real time, energy consumed by each process and sub-process running in the system. Bitwatts is used to build power meters in virtualised environment where these power meters are software defined. Bitwatts scales both in number of monitored processes and virtualisation levels. Section 2 gives a brief overview and motivation of Bitwatts, and PowerAPI.
Each node in the production server cluster is installed with the Bitwatts tool so that, the energy consumed by each process in each node is measured.

4.3 Experiment scenario for different databases

This section explains about three different scenarios for each database. The experiment scenarios are formed based on the goals of this work and they assist in answering the research questions.

4.3.1 InfluxDB

This section consists of three different scenarios which answers RQ1, RQ2 and RQ3.

4.3.1.1 Scenario 1

In this scenario the test-bed is considered with the installation of Bitwatts on all the three nodes. The main objective of this scenario is to measure the average energy consumed by each node after the installation of InfluxDB. The tool Bitwatts provides the estimated power consumed by each process for every second in the given time interval. This estimated power is then converted to energy.
The total energy consumed on each node by InfluxDB can be obtained by the formula:

$$\text{Energy}=\sum_{i=0}^{t} power(i)$$

Where,
t is the total time in seconds in this case 1800 seconds
power(i) is power reading at i^{th} second

Once the Energy is obtained Average energy consumed can be calculated using

$$\text{Average energy} = \frac{1}{n}\sum_{m=1}^{n} Energy(m)$$

Where,
n is total number of iterations in the experiment in this case it is 20

After the installation of InfluxDB the Bitwatts command mentioned above is executed for a duration of 30 minutes on each node in the cluster by changing the {required process name} (this is a Bitwatts module which helps in specifying a particular process whose power is measured) to the process id (pid) assigned to InfluxDB on the respective node. The average energy consumed by the database on each node after installation is calculated by the above mentioned formulae.

4.3.1.2 Scenario 2

The main objective of this scenario is to measure energy consumed by each node, when synchronisation among multiple nodes is considered. Synchronisation of a database depends upon the retention policy. Retention policy describes how long the data is stored in the database and how many copies of that data are stored in the cluster.

The number of copies to be stored in the cluster can be determined by the replication factor, which is an attribute of retention policy. The cluster contains 3 nodes i.e., master node, slave node 1 and slave node 2. Synchronisation can be considered in 3 ways: a) when the replication factor is 3, b) when the replication factor is 2, and c) when the replication factor is 1. Bitwatts command is executed for each node in the cluster by changing the {required process name} to the process id (pid) assigned to InfluxDB on the respective node for a duration of 30 minutes.

a. A database is created with a retention policy as follows:
    - Database: NOAA_water_database
    - Retention policy:
        - Retention policy name: Only_30_minutes
        - Duration: 30 minutes
        - Default: true
        - Replication factor: 3

Energy consumed on each node is measured for 30 minutes and the power consumed during synchronisation is calculated.

b. For the created database above mentioned the retention policy is altered as follows:
   - Database: NOAA_water_database
   - Retention policy:
     - Retention policy name: Only_30_minutes
     - Duration: 30 minutes
     - Default: true
     - Replication factor: 2

Energy consumed on each node is measured and querying operations are performed on each node, to differentiate between the nodes that contains data.

c. For the created database above mentioned the retention policy is altered as follows:
   - Database: NOAA_water_database
   - Retention policy:
     - Retention policy name: Only_30_minutes
     - Duration: 30 minutes
     - Default: true
     - Replication factor: 1

Energy consumed on each node is measured and querying operations are performed on each node, to differentiate between the nodes that contains data.

4.3.1.3 Scenario 3

The main objective of this scenario is to measure energy consumed by each node, when read and write operations are executed by the database. Bitwatts command is executed for each node in the cluster by changing the {required process name} to the process id (pid) assigned to InfluxDB on the respective node for a duration of 30 minutes. A database is created with default retention policy with a replication factor of 3. The data set used for performing read and write operations, was obtained from the website of InfluxDB, which has a set of 20,000 data points for read operations, and for write operations in multiple queries a dataset with 20,000 data points is considered and a dataset with 76290 data points is considered for writing data from a file.

As there are two operations read and write, the scenario has two objectives:

a. Read:

The HTTP API is the primary means for querying data in InfluxDB. Querying data can be done in two forms, Multiple queries and Continues Queries. Using a single API call multiple queries can be sent from the node where the query is executed to InfluxDB. Energy consumed during read operations can be measured in two ways, i.e., 1) Multiple queries are requested through http API to fetch data for a duration of 30 minutes, 2) Continuous queries for a duration of 30 minutes.

1. Multiple Queries:
   A group of queries which contain 20,000 data points are requested in a single API call from each node in the cluster one at a time. Through HTTP API multiple queries are performed for fetching data from master node, slave node 1 and slave node 2, in a sequential order.

   Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Querying from master node, ii) Querying from slave node 1, and iii) Querying from slave node 2.
2. Continuous queries (CQ):

A continuous query is an InfluxQL query that the system runs automatically and periodically within a database. When large amounts of raw data are present, repeatedly running the same queries by hand is be tedious so Continuous queries are created such that down sampling (i.e., the process of reducing the data rate) of that data is simplified. The results of CQ are stored in a specified measurement. A CQ is created with a duration of 30 minutes. Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Querying from master node, ii) Querying from slave node 1, and iii) Querying from slave node 2.

b. Write:

1. There are many ways to write data into InfluxDB including command interface, client libraries and plugins for common data formats such as graphite. But all of these have their own dependencies which increase energy consumption so we have opted the built in HTTP API for inserting data. Having HTTP API as the primary means of writing data to InfluxDB we ca insert data in two different ways. Energy consumed during write operations is measured in these two ways,1) Multiple queries are performed through as single HTTP API and 2) Writing multiple points from a file. The continuous queries mentioned above in read operations cannot be created for write operations. Multiple Queries through HTTP API:
   A group of queries with 20,000 data points are sent in a single API call from each node in a cluster in a sequential order.
   Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Inserting data from master node, ii) Inserting data from slave node 1, and iii) Inserting data from slave node 2.
2. Writing multiple points from a file:
   Energy is measured during the process of importing a file with 76290 data points into the database through HTTP API. Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Importing file from master node, ii) Importing file from slave node 1, and iii) Importing file from slave node 2.

4.3.2 OpenTSDB

This section consists of three different scenarios which answers RQ1, RQ2 and RQ3.

4.3.2.1 Scenario 1

In this scenario the test-bed is considered with the installation of Bitwatts on all the three nodes. The main objective of this scenario is to measure the average energy consumed by each node after the installation of OpenTSDB. The energy consumed by OpenTSDB on each node is obtained by the formula

$$\text{Energy}=\sum_{i=0}^{t} power(i)$$

Where,
t is the total time in seconds in this case 1800 seconds
power(i) is power reading at i^{th} second

Once the Energy is obtained Average energy consumed can be calculated using

$$\text{Average energy} = \frac{1}{n}\sum_{m=1}^{n} Energy(m)$$

Where,
n is total number of iterations in the experiment in this case it is 20

4.3.2.2 Scenario 2

The main objective of this scenario is to measure energy consumed by each node, when synchronisation among multiple nodes is considered. Zookeeper is the one task that coordinates distributed components and provides mechanisms to keep them in sync. Zookeeper is used to detect the failure of the NameNode and elect a new NameNode. It's also used with HBase to manage the states of the HMaster and the RegionServers.But the functionality of zookeeper mainly depends on the HDFS replication factor. By changing the replication factor in the cluster the variations of energy consumed by the database during synchronization is calculated. In the HDFS configuration files the replication factor is set to "N", where N is the number of nodes in the cluster and the HBase is restarted and OpenTSDB is up and running. Now the energy consumed by the database is measured for about 30 minutes. Similarly, the experiment is executed for replication factor is "N-1" and "N-2", respected energy is measured on each node.

4.3.2.3 Scenario 3

The main objective of this scenario is to measure energy consumed by each node, when read and write operations are executed by the database. Bitwatts command is executed for each node in the cluster by changing the {required process name} to the process id (pid) assigned to OpenTSDB on the respective node for a duration of 30 minutes. A database is created with default retention policy with a replication factor of N.

As there are two operations read and write, the scenario has two objectives:

a. Read

OpenTSDB offers a number of means to extract data such as CLI tools, an HTTP API and as a GnuPlot graph. OpenTSDB provides an HTTP based application programming interface to enable integration with external systems. Almost all OpenTSDB features are accessible via the API such as querying time series data, managing metadata and storing data points. Here for experimentation, querying data is done in two forms, Multiple queries and querying via Grouping. Using a single API call, multiple queries can be sent to OpenTSDB. Energy consumed during read operations can be measured in two ways, i.e., 1) Multiple queries are requested through http API to fetch data for a duration of 30 minutes, 2) Grouping for a duration of 30 minutes.

1. Multiple Queries:

A group of queries which contain 18,000 data points are requested in a single API call from each node in the cluster one at a time. Through HTTP API multiple queries are performed for fetching data from master node, slave node 1 and slave node 2, in a sequential order.
Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Querying from master node, ii) Querying from slave node 1, and iii) Querying from slave node 2.

2. Grouping:

A query aggregates time series data with multiple tags into groups based on a tag value. Two special characters can be passed to the right of the equals symbol in a query a) "*****" - The asterisk will return a separate result for each unique tag value and b) "**|**" - The pipe will return a separate result *only* for the exact tag values specified.

Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Querying from master node, ii) Querying from slave node 1, and iii) Querying from slave node 2.

b. Write:

There are currently three main methods to get data into OpenTSDB: Telnet API, HTTP API and batch import from a file. Here for experimentation, writing data is done in two ways, HTTP API and Importing from a file. Energy consumed during write operations is measured in these two ways,1) Multiple queries are performed through as single HTTP API and 2) Writing multiple points from a file.

1.Multiple queries:

A group of queries with 18,000 data points are sent in a single API call from each node in a cluster in a sequential order.

Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Inserting data from master node, ii) Inserting data from slave node 1, and iii) Inserting data from slave node 2.

2.Writing multiple points from a file:

Energy is measured during the process of importing a file with 76290 data points into the database through HTTP API. Energy is measured for a duration of 30 minutes during each process, i.e., as follows: i) Importing file from master node, ii) Importing file from slave node 1, and iii) Importing file from slave node 2.

5 RESULTS

This chapter presents the results obtained from the experiments conducted. The results are divided into section corresponding to the scenarios discussed in section 4.3. Thus obtained results were tabulated in this section. A detailed analysis of the results is presented in Section 6.

5.1 InfluxDB

5.1.1 Scenario 1:

This section presents the results of 3 nodes in the cluster, i.e., master node, slave node 1, and slave node 2, where the database is installed and the energy consumed by it is measured. Bitwatts provides an estimated value of power for every second. The total experiment is executed for 30 minutes of time interval, that is a series of 1800 values of estimated power are reported. The same experiment is conducted in twenty iterations. So the average value of power for 20 iterations is tabulated below. For further analysis, the measured values are tabulated below. The energy consumed for the installation of InfluxDB, as mentioned in the scenario 4.3.1.1 in the cluster is measured and tabulated as follows:

Name of the node	Power (watts)	Energy (Kilo Joules)
Mater node	21.97	39.55
Slave node 1	1.01	1.81
Slave node 2	8.99	16.17

Table 1: Power and energy consumptions of InfluxDB in Scenario 1

5.1.2 Scenario 2:

This section presents the results of 3 nodes in the cluster, i.e., master node, slave node 1, and slave node 2, when synchronization among these nodes is considered for the respective database and the energy consumed by it is measured. For further analysis, the measured values are tabulated below:

a. The energy consumed by the database on each node, when the replication factor of the database is set to 3, is measured and tabulated as follows:

Name of the node	Power (watts)	Energy (Kilo Joules)
Mater node	284.94	512.89
Slave node 1	197.84	356.11
Slave node 2	199.70	359.46

Table 2: Power and energy consumptions of InfluxDB in Scenario 2.a

b. The energy consumed by the database on each node, when the replication factor of the database is set to 2, is measured and tabulated as follows:

Name of the node	Power (30 minutes)	Energy (30 minutes)
Mater node	225.70	406.26
Slave node 1	122.06	219.71
Slave node 2	108.58	195.45

Table 3: Power and energy consumptions of InfluxDB in Scenario 2.b

c. The energy consumed by the database on each node, when the replication factor of the database is set to 1, is measured and tabulated as follows:

Name of the node	Power (30 minutes)	Energy (30 minutes)
Mater node	181.36	326.45
Slave node 1	67.29	121.12
Slave node 2	71.37	128.47

Table 4: Power and energy consumptions of InfluxDB in Scenario 2.c

5.1.3 Scenario 3:

This section presents the results of 3 nodes in the cluster, i.e., master node, slave node 1 and slave node 2, when read and write operations are performed in the database and the energy consumed is measured. This section contains 2 objectives; read and write.

a. **Read operations**:

i. Energy consumed by the database on each node is measured and tabulated, when multiple queries are requested from the database.

- Energy is measured when queries are requested from the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	58.15	104.67	5.23
Slave node 1	18.17	32.71	1.63
Slave node 2	13.11	23.61	1.18

Table 5: Power and energy consumptions of InfluxDB in Scenario 3.a.i (Master Node)

- Energy is measured when queries are requested from the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	45.90	82.63	4.13
Slave node 1	16.59	29.87	1.49
Slave node 2	16.11	29.00	1.45

Table 6: Power and energy consumptions of InfluxDB in Scenario 3.a.i (Slave Node 1)

- Energy is measured when queries are requested from the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	53.87	96.98	4.84
Slave node 1	14.68	26.42	1.32
Slave node 2	11.81	21.27	1.06

Table 7: Power and energy consumptions of InfluxDB in Scenario 3.a.i (Slave Node 2)

ii. Energy consumed by the database on each node is measured and tabulated, when continuous query is created in the database.

- Energy is measured when continuous query is created for the database on the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	43.89	79.00	3.95
Slave node 1	17.52	31.53	1.57
Slave node 2	10.48	18.95	0.94

Table 8: Power and energy consumptions of InfluxDB in Scenario 3.a.ii (Master Node)

- Energy is measured when continuous query is created for the database on the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules	Energy (joules) per data point
Mater node	55.34	99.62	4.98
Slave node 1	20.98	37.76	1.88
Slave node 2	12.84	23.11	1.15

Table 9: Power and energy consumptions of InfluxDB in Scenario 3.a.ii (Slave Node 1)

- Energy is measured when continuous query is created for the database on the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	51.84	93.30	4.66
Slave node 1	21.45	38.61	1.93
Slave node 2	17.93	32.27	1.61

Table 10: Power and energy consumptions of InfluxDB in Scenario 3.a.ii (Slave Node 2)

b. Write operations:

i. Energy consumed by the database on each node is measured and tabulated, when multiple queries for inserting data into the database are performed.

- Energy is measured when queries for inserting the data are sent from the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	5.70	10.26	0.51
Slave node 1	1.64	2.95	0.14
Slave node 2	1.73	3.12	0.15

Table 11: Power and energy consumptions of InfluxDB in Scenario 3.b.i (Master Node)

- Energy is measured when queries for inserting the data are sent from the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules	Energy (joules) per data point
Mater node	12.52	22.54	1.12
Slave node 1	3.45	6.21	0.31
Slave node 2	1.56	2.81	0.14

Table 12: Power and energy consumptions of InfluxDB in Scenario 3.b.i (Slave Node 1)

- Energy is measured when queries for inserting the data are sent from the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	17.67	31.81	1.59
Slave node 1	4.38	7.88	0.39
Slave node 2	4.33	7.80	0.39

Table 13: Power and energy consumptions of InfluxDB in Scenario 3.b.i (Slave Node 2)

ii. Energy consumed by the database on each node is measured and tabulated, when a file with multiple data points is inserted into the database.

- Energy is measured when file is imported from the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	23.46	42.22	0.55
Slave node 1	7.49	13.48	0.17
Slave node 2	5.47	9.85	0.12

Table 14: Power and energy consumptions of InfluxDB in Scenario 3.b.ii (Master Node)

- Energy is measured when file is imported from the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	13.81	24.85	0.32
Slave node 1	11.21	20.17	0.26
Slave node 2	6.86	12.34	0.16

Table 15: Power and energy consumptions of InfluxDB in Scenario 3.b.ii (Slave Node 1)

- Energy is measured when file is imported from the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	24.47	44.05	0.57
Slave node 1	6.59	11.87	0.15
Slave node 2	13.86	24.95	0.32

Table 16: Power and energy consumptions of InfluxDB in Scenario 3.b.ii (Slave Node 2)

5.2 OpenTSDB

5.2.1 Scenario 1

This section presents the results of 3 nodes in the cluster, i.e., master node, slave node 1, and slave node 2, where the database is installed and the energy consumed by it is measured. For further analysis, the measured values are tabulated below. The energy consumed for the installation of OpenTSDB, as mentioned in the scenario 4.3.2.1 in the cluster is measured and tabulated as follow:

Name of the node	Power (watts)	Energy (Kilo Joules)
Mater node	47.40	85.33
Slave node 1	38.03	68.45
Slave node 2	39.34	70.81

Table 17: Power and energy consumptions of OpenTSDB in Scenario 1

5.2.2 Scenario 2

This section presents the results of 3 nodes in the cluster, i.e., master node, slave node 1, and slave node 2, when synchronization among these nodes is considered for the respective database and the energy consumed by it is measured. For further analysis, the measured values are tabulated below:

a. The energy consumed by the database on each node, when the replication factor of the database is set to 3, is measured and tabulated as follows:

Name of the node	Power (watts)	Energy (Kilo Joules)
Mater node	339.03	610.27
Slave node 1	206.46	371.63
Slave node 2	65.32	310.11

Table 18: Power and energy consumptions of OpenTSDB in Scenario 2.a

b. The energy consumed by the database on each node, when the replication factor of the database is set to 2, is measured and tabulated as follows:

Name of the node	Power (30 minutes)	Energy (30 minutes)
Mater node	390.80	703.45
Slave node 1	229.23	412.62
Slave node 2	229.59	413.26

Table 19:Power and energy consumptions of OpenTSDB in Scenario 2.b

c. The energy consumed by the database on each node, when the replication factor of the database is set to 1, is measured and tabulated as follows:

Name of the node	Power (30 minutes)	Energy (30 minutes)
Mater node	486.75	876.15
Slave node 1	218.29	392.92
Slave node 2	237.07	426.73

Table 20: Power and energy consumptions of OpenTSDB in Scenario 2.c

5.2.3 Scenario 3

This section presents the results of 3 nodes in the cluster, i.e., master node, slave node 1 and slave node 2, when read and write operations are performed in the database and the energy consumed is measured. This section contains 2 objectives; read and write.

a. **Read operations**:

i. Energy consumed by the database on each node is measured and tabulated, when multiple queries are requested from the database.

- Energy is measured when queries are requested from the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	117.03	210.67	11.70
Slave node 1	70.25	126.45	7.02
Slave node 2	61.81	111.26	6.18

Table 21: Power and energy consumptions of OpenTSDB in Scenario 3.a.i (Master Node)

- Energy is measured when queries are requested from the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	217.32	391.17	21.73
Slave node 1	109	196.21	10.90
Slave node 2	87.02	156.63	8.70

Table 22: Power and energy consumptions of OpenTSDB in Scenario 3.a.i (Slave Node 1)

- Energy is measured when queries are requested from the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	224.43	403.98	22.44
Slave node 1	121.90	219.42	12.19
Slave node 2	117.93	212.27	11.79

Table 23: Power and energy consumptions of OpenTSDB in Scenario 3.a.i (Slave Node 2)

ii. Energy consumed by the database on each node is measured and tabulated, when continuous query is created in the database.

- Energy is measured when continuous query is created for the database on the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	268.13	482.63	26.81
Slave node 1	164.88	296.78	16.49
Slave node 2	161.11	290.03	16.11

Table 24: Power and energy consumptions of OpenTSDB in Scenario 3.a.ii (Master Node)

- Energy is measured when continuous query is created for the database on the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules	Energy (joules) per data point
Mater node	165.27	297.48	16.53
Slave node 1	61.20	110.17	6.12
Slave node 2	105.42	189.76	10.54

Table 25: Power and energy consumptions of OpenTSDB in Scenario 3.a.ii (Slave Node 1)

- Energy is measured when continuous query is created for the database on the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	142.45	256.41	14.24
Slave node 1	98.32	176.98	9.83
Slave node 2	95.06	171.10	9.51

Table 26: Power and energy consumptions of OpenTSDB in Scenario 3.a.ii (Slave Node 2)

b. Write operations:

i. Energy consumed by the database on each node is measured and tabulated, when multiple queries for inserting data into the database are performed.

- Energy is measured when queries for inserting the data are sent from the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	151.88	273.38	15.18
Slave node 1	117.71	211.87	11.77
Slave node 2	86.87	156.36	8.68

Table 27: Power and energy consumptions of OpenTSDB in Scenario 3.b.i (Master Node)

- Energy is measured when queries for inserting the data are sent from the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules	Energy (joules) per data point
Mater node	170.82	307.47	17.08
Slave node 1	141.35	254.44	14.13
Slave node 2	149.03	268.27	14.90

Table 28: Power and energy consumptions of OpenTSDB in Scenario 3.b.i (Slave Node 1)

- Energy is measured when queries for inserting the data are sent from the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	220.72	397.31	22.07
Slave node 1	165.84	298.51	16.58
Slave node 2	159.53	287.15	15.95

Table 29: Power and energy consumptions of OpenTSDB in Scenario 3.b.i (Slave Node 2)

ii. Energy consumed by the database on each node is measured and tabulated, when a file with multiple data points is inserted into the database.

- Energy is measured when file is imported from the master node.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	311.44	560.59	7.35
Slave node 1	216.88	390.39	5.12
Slave node 2	219.41	394.94	5.11

Table 30: Power and energy consumptions of OpenTSDB in Scenario 3.b.ii (Master Node)

- Energy is measured when file is imported from the slave node 1.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	233.56	420.41	5.51
Slave node 1	172.39	310.31	4.07
Slave node 2	154.83	278.70	3.65

Table 31: Power and energy consumptions of OpenTSDB in Scenario 3.b.ii (Slave Node 1)

- Energy is measured when file is imported from the slave node 2.

Name of the node	Power (watts)	Energy (kilo joules)	Energy (joules) per data point
Mater node	227.40	409.33	5.36
Slave node 1	128.73	231.72	3.04
Slave node 2	175.02	315.03	4.13

Table 32: Power and energy consumptions of OpenTSDB in Scenario 3.b.ii (Slave Node 2)

6 ANALYSIS AND COMPARISON

6.1 Analysis of InfluxDB

6.1.1 RQ1

According to the method specified in chapter 4, the total average energy consumed by InfluxDB database on each node in the cluster is shown in the following graph.

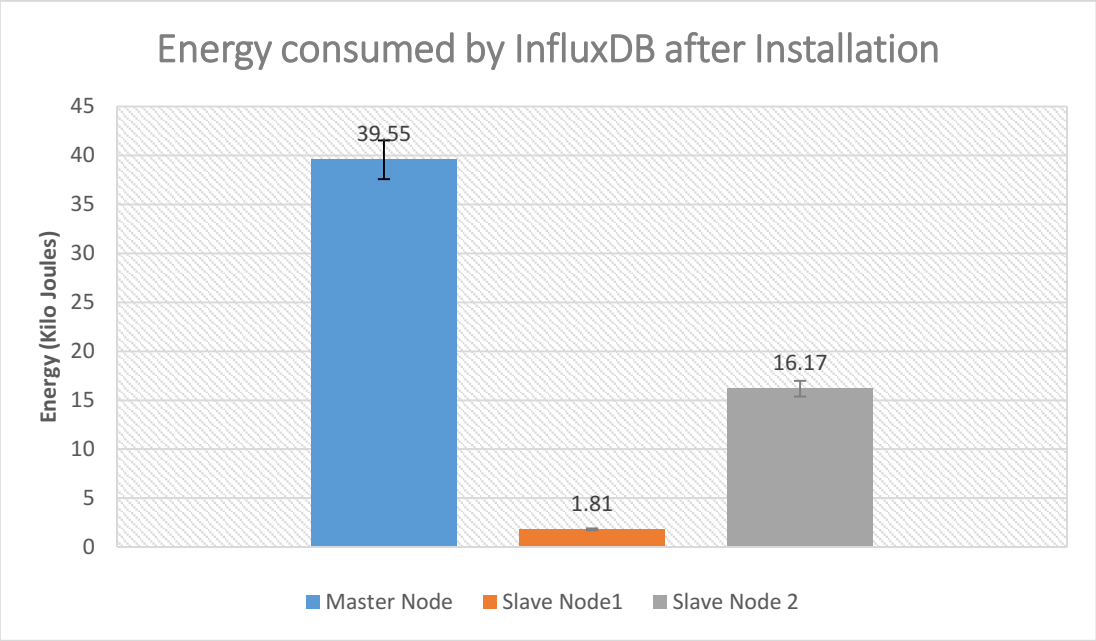

Figure 5: Energy consumed by InfluxDB after Installation

From the figure 5, the energy consumed by InfluxDB on the master node is quite high when compared to the other two slave nodes in the cluster. This can be explained by considering that the master node hosts various storage and processing management services for the entire cluster whereas the slave nodes act as the data nodes which performs only the data services. A cluster should always contain at least one data node. The master node manages the storage functionalities of the entire cluster and other different functions such as administration, databases, retention policies, continuous queries. The data nodes perform data service which is an InfluxDB services that sends time-series data to the node. As there is no load on the distributed time series database, each node just runs in its idle mode which specifies only the basic functionalities of its character. So the energy consumed by master node is quite high when compared to the other slave nodes in the cluster. The energy consumed by slave node 2 is comparatively more that energy consumed by slave node 1 as slave node 2 is installed prior to the slave node 1. From the results it can be concluded that the average energy consumed by the time series database on each node in the cluster is higher for master node than the slave nodes.

6.1.2 RQ2

According to the method explained in section 4, the three different scenarios where synchronization is considered is by changing the replication factor of the database from “N”, “N-1”, “N-2” … to 1, where N is the number of nodes in the cluster. Figure 6 shows the energy measured by each node in the cluster in all the three scenarios. Considering each scenario, when the replication factor is equal to 1, the energy consumed by master node is quite high

which is approximately 201 kilo joules more when compared to the average of the other two slave nodes. When replication factor is 2 the master node consumed approximately 198 kilo joules more when compared to the average of the two slave nodes. Similarly, when the replication factor is 3 the energy consumed by master node is approximately 235 kilo joules more when compared to the average of the two slave nodes. In all the three scenarios it can be seen that master node has the large amount of energy consumption when compared to the slave nodes as master node acts as both data node and Consensus node that is it handles different administrative functionalities as well as stores data in it.

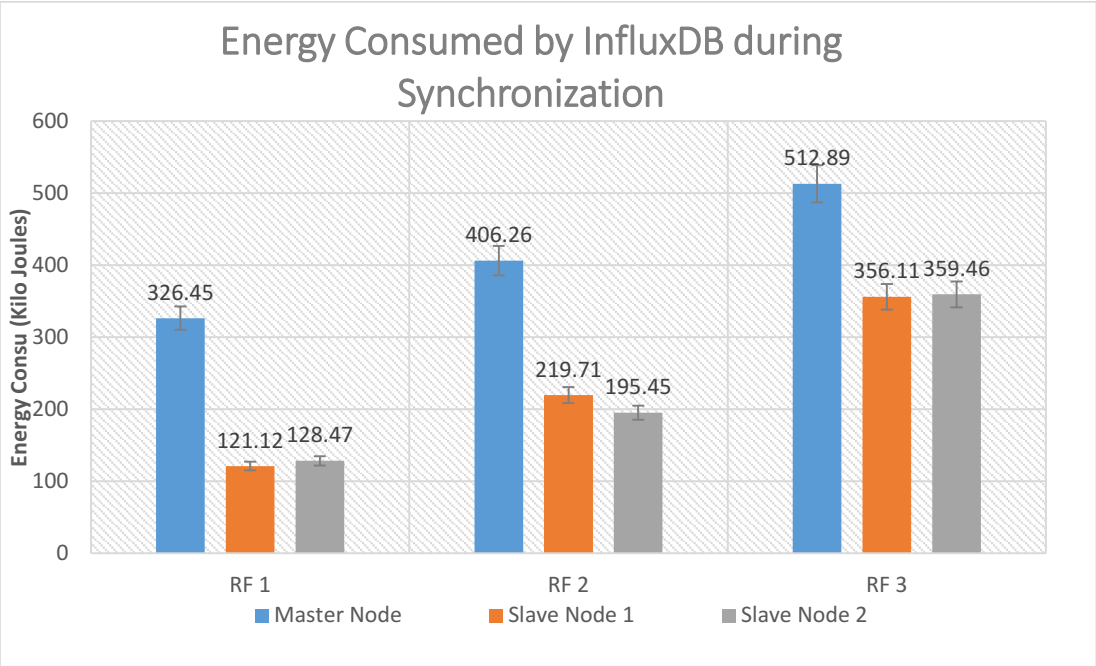

Figure 6: Energy consumed by InfluxDB during Synchronization

Coming to the comparison of scenarios, the energy consumed by master node when the RF is 3 is more when compared to the other two scenarios. This is because in this scenario the master node experiences a larger number of processes for replicating data and synchronizing it in all the 3 nodes in the cluster. The energy difference between RF 3 to RF 2 for master node is approximately 106 kilo joules and between RF 2 to RF 1 is approximately 80 kilo joules. Similarly, the energy consumed by Slave node 1 and Slave node 2 is more when the RF is 3, because on all the three nodes the data is replicated which increases energy consumption on all the nodes. It can be seen that replicating data on all the N nodes in the cluster consumes more energy when compared to replicating data to N-1 and N-2 nodes in the cluster.

6.1.3 RQ3

a. Read

To measure the energy consumed by database on each node in the cluster two scenarios were considered i. Multiple queries and ii. Continuous queries. The below graphs shows the energy consumed by each node when queries are requested from Master node, Slavenode1 and Slavenode2. Let us consider the querying from master node as “case 1” and querying from slave node 1 as “case 2” and from slave node 2 as “case 3”. Considering both scenarios, when queries are requested from any node that is in any “cases” it can be seen that the energy consumption is always high for master node. This can be explained as the master node performs all the processing functionalities of the cluster.

i. Multiple Queries

Figure 7 shows the energy consumed by each node when multiple queries are requested from Master node, Slave node1 and Slave node2. When each node is compared in all the three scenarios there is a lot of variation. So while comparing the amount of energy consumed by master node in “case 1”, “case 2” and “case3”, it can be observed that the energy consumption is 22.2 kilo joules more in “case 2” than that of “case 3” and the energy consumption is 8 kilo joules more in “case 1” than that of “case 3”. This can be explained as whenever a query for reading data is executed from the master node lot of energy is consumed than the slave nodes. This is because the master node processes functions such as synchronization of the cluster and then responses to every query that are requested in the cluster. Coming to the slave node 1, while comparing the energy consumed by slave node 1 in “case 1”, “case 2” and case3 it can be seen that the energy consumption of slave node 1 is 2.82 kilo joules more than in “case 2” the energy consumption of slave node 1 is 6.89 kilo joules more than “case 1”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 2” is 5.39 kilo joules more than in “case 1” and 7.73 kilo joules more than “case 3”.

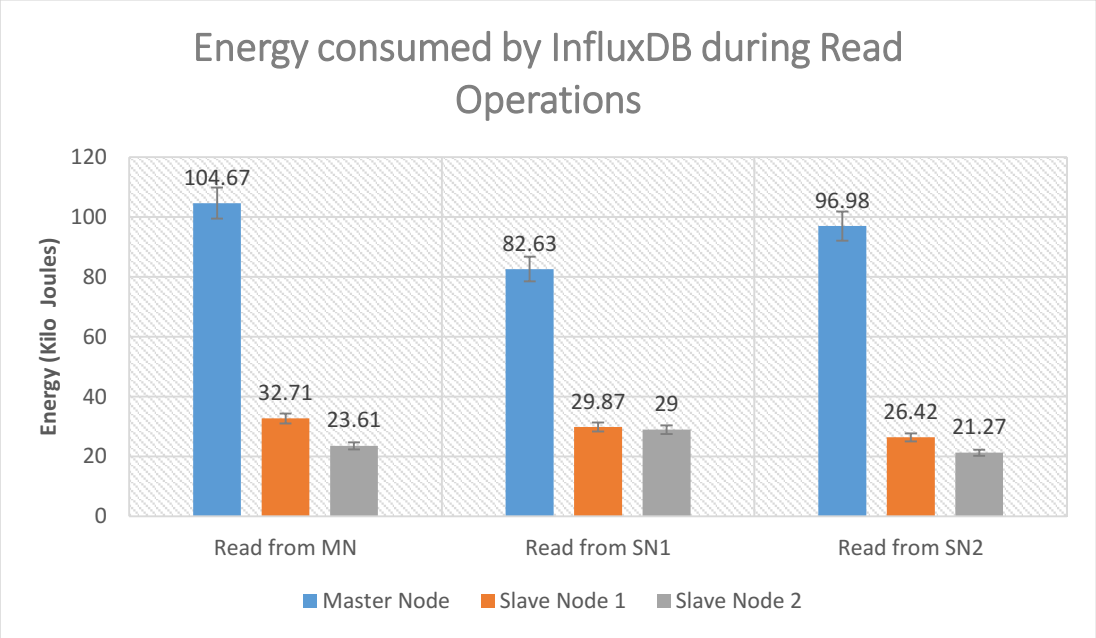

Figure 7: Energy consumed by InfluxDB during Multiple Queries (READ)

On average, each node in the InfluxDB cluster InfluxDB consumes 54 kilo joules when querying is done from master node. Similarly, an average of 47 kilo joules and 48 kilo joules is consumed by InfluxDB on each node when querying is done from slave node 1 and slave node 2. It can be said that while creating a single API call for multiple queries it is always better to create it from any of the slave nodes instead of the master node for InfluxDB as the energy consumed by slave nodes is less than that consumed by the master node.

ii. Continuous Queries

The figure 8, shows the energy consumed by each node when Continuous queries are requested from Master node, Slave node1 and Slave node2.When a continuous query is

created in “case 1” and requested from Master node it can be seen that the energy consumed is low when compared to other cases. While comparing the energy consumed by master node in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 2” is 20.62 kilo joules more than “case 1” and 6.32 kilo joules more than “case 3”. While comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 7.08 kilo joules more than “case 1” and 0.85 kilo joules more than “case 2”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in case 3”is 13.32 kilo joules more than “case 1” and 9.16 kilo joules more than “case 2”. On an average in the cluster InfluxDB consumes an average of 43 kilo joules on each node when querying is done from master node. Similarly, an average of 53 kilo joules and 55 kilo joules is consumed by InfluxDB on each node when querying is done from slave node 1 and slave node 2.

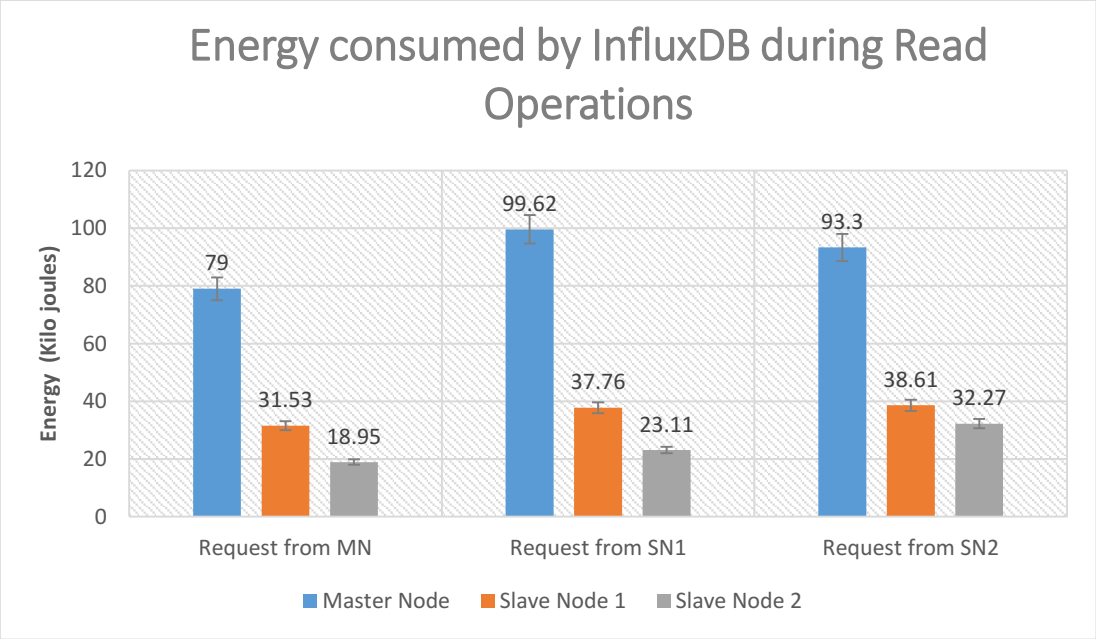

Figure 8: Energy consumed by InfluxDB during Continuous Queries (READ)

It can be said that while creating a continuous query it is always better to create it from the master node instead of the slave nodes for InfluxDB as the energy consumed by master node is less than that consumed by the slave nodes.

b. Write

To measure the energy consumed by database while inserting data into InfluxDB two scenarios were considered i. Multiple insert queries and ii. Importing from file. The below graphs show the energy consumed by each node when insert queries are requested from Master node, Slave node1 and Slave node2. Let us consider the querying from master node as “case 1” and querying from slave node 1 as “case 2” and from slave node 2 as “case 3”. Considering both scenarios, when queries are requested from any node that is in all the cases it can be seen that the energy consumption is always high for master node. This can be analyzed as the master node performs all the processing functionalities of the cluster.

i. Multiple Queries

Figure 9 shows the energy consumed by each node when Multiple insert queries are requested from Master node, Slavenode1 and Slavenode2.When a single API is created for multiple insert queries in “case 1” the energy consumed is low when compared to other cases. While comparing the energy consumed by master node in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 21.55 kilo joules more than “case 1” and 9.27 kilo joules more than “case 2”. While comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 4.93 kilo joules more than “case 1” and 1.67 kilo joules more than “case 2”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 4.68 kilo joules more than “case 1” and 4.99 kilo joules more than “case 2”.

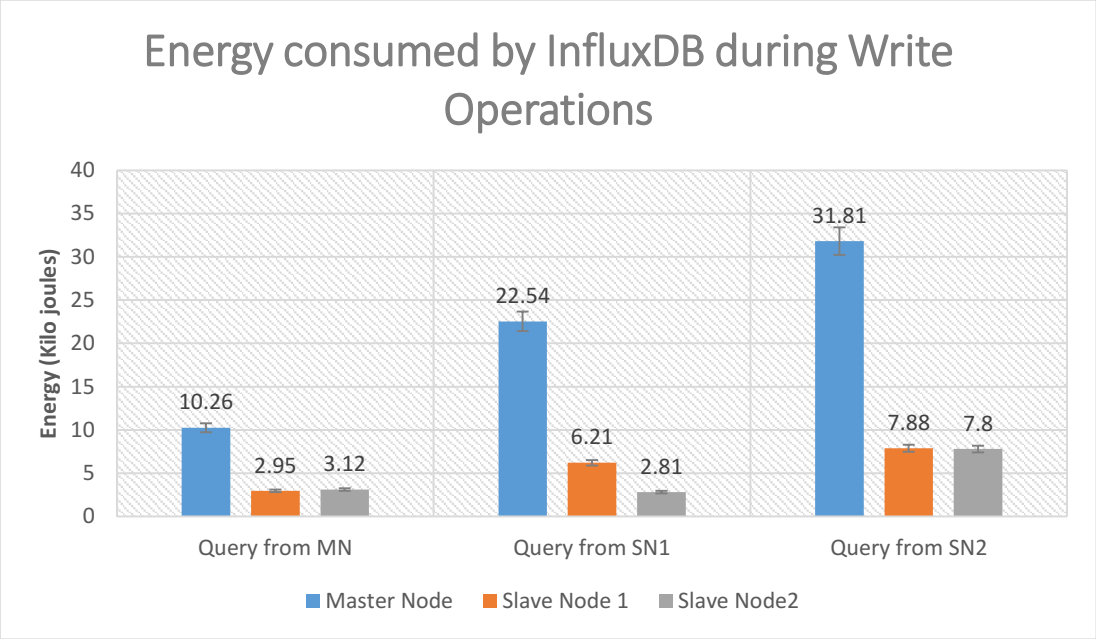

Figure 9: Energy consumed by InfluxDB during Multiple Queries (WRITE)

On an average in the cluster InfluxDB consumes an average of 5 kilo joules on each node when querying is done from master node. Similarly, an average of 10 kilo joules and 16 kilo joules is consumed by InfluxDB on each node when inserting is done from slave node 1 and slave node 2. It can be said that while Inserting multiple data points it is always better to insert it from the master node instead of the slave nodes for InfluxDB as the energy consumed during master node is less than the slave nodes.

ii. Import from File

Figure 10 shows the energy consumed by each node when a Single data file is inserted from Master node, Slavenode1 and Slavenode2.When a Single data file with multiple data points is inserted in “case 1” it can be seen that the energy consumed is high when compared to “case 2” but low when compared to “case 3”. While comparing the energy consumed by master node in “case 1”, “case 2” and “case 3” it can be seen that the energy consumption in “case 3”is 1.83 kilo joules more than “case 1” and 19.2 kilo joules more than “case 2”. While comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 2” is 11.37 kilo joules more than “case 1” and 12.98 kilo joules more than “case 3” Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 15.1 kilo joules more than “case 1” and 12.61 kilo joules more than “case 2”. On an average in the cluster InfluxDB

consumes an average of 22 kilo joules on each node when querying is done from master node. Similarly, an average of 19 kilo joules and 27 kilo joules is consumed by InfluxDB on each node when inserting is done from slave node 1 and slave node 2.

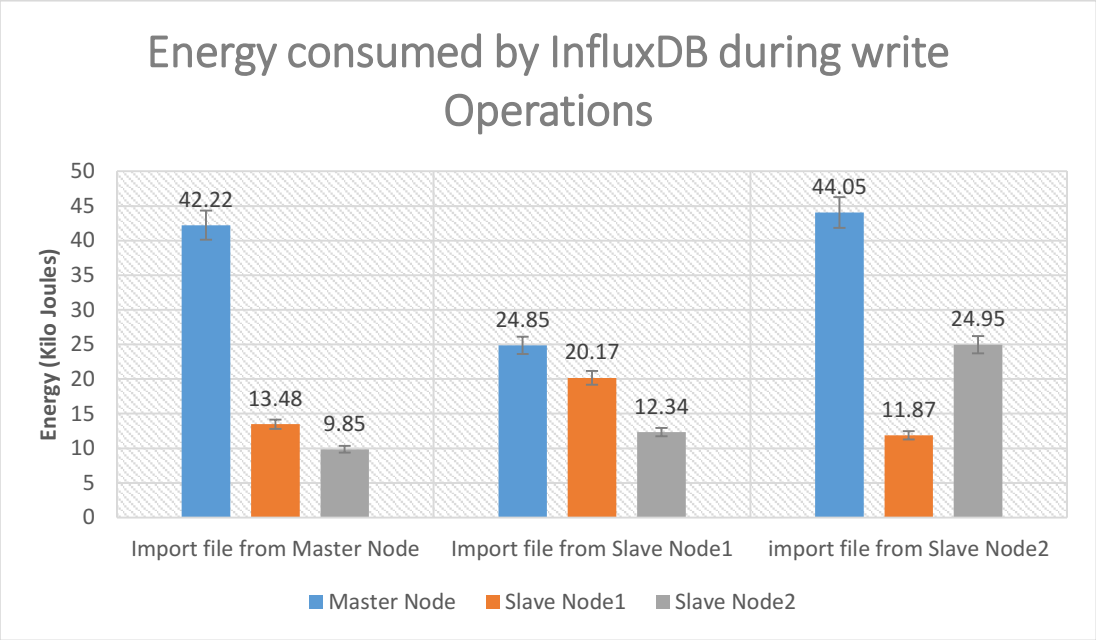

Figure 10: Energy consumed by InfluxDB during data importing from file (WRITE)

It can be said that while creating a continuous query it is always better to import a file from the master node instead of the slave nodes for InfluxDB as the energy consumed during master node is less than the average of slave nodes.

6.2 Analysis of OpenTSDB

6.2.1 RQ1

According to the method specified in chapter 4, the total average energy consumed by OpenTSDB database on each node in the cluster is shown in the following graph. From the figure 11, the energy consumed by OpenTSDB on the master node is high when compared to the other two slave nodes in the cluster. A hypothetical explanation is that the master node hosts various storage and processing management services for the entire cluster whereas the slave nodes act as the data nodes which performs only the data services. A cluster should always contain at least one data node. The master node manages the storage functionalities of the entire cluster and other different functions such as administration, databases, retention policies, continuous queries. Master node works as Active NameNode, Resource Manager, Journal Node, Job Tracker, HMaster, Zookeeper. The data nodes perform as HRegion Servers, Node Manager, Application master, Containers. As there is no load on the distributed time series database, each node just runs in its idle mode where only the basic functionalities are active. So the energy consumed by master node is high when compared to the other slave nodes in the cluster. The energy consumed by the two slave nodes have a difference of approximately 2.36 kilo joules.

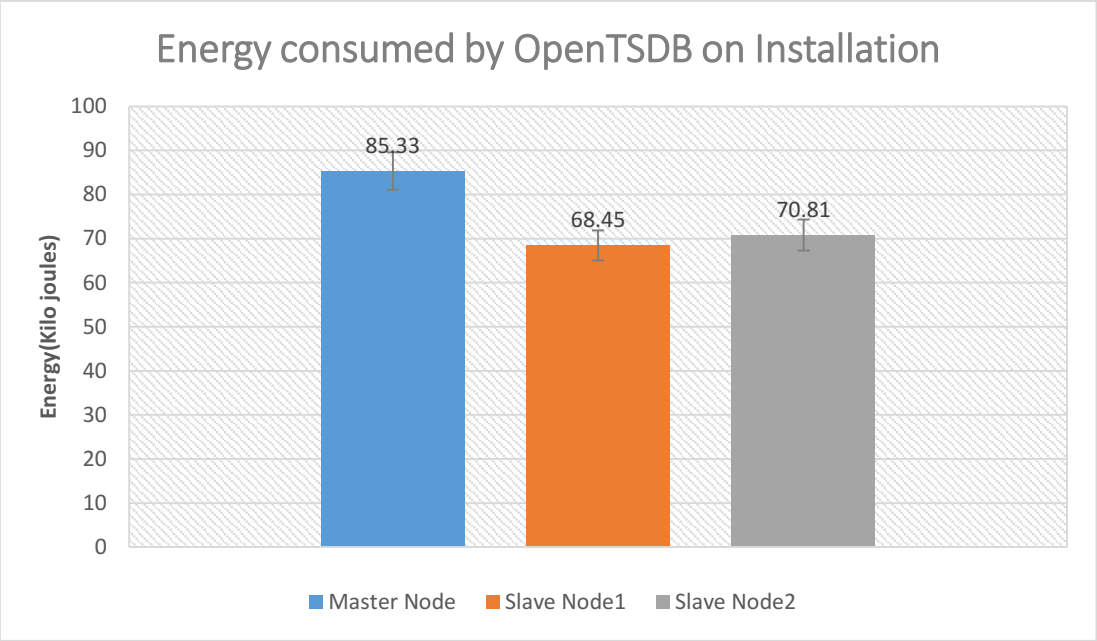

Figure 11: Energy consumed by OpenTSDB upon Installation

From the results, it can be concluded that the average energy consumed by OpenTSDB on each node in the cluster is more for master node than the slave nodes.

6.2.2 RQ2

According to the method explained in section 4, the three different scenarios where synchronization is considered by changing the replication factor from "N" to "N-1" and "N-2", where N is the number of nodes in the cluster. The figure 12, shows the energy measured by each node in the cluster in all the three scenarios. Considering each scenario, when the replication factor is equal to 1, the energy consumed by master node is quite high which is approximately 466 kilo joules more when compared to the average of the other two slave nodes. When replication factor is 2 the master node consumed approximately 290 kilo joules more when compared to the average of the two slave nodes. Similarly, when the replication factor is 3 the energy consumed by master node is approximately 269 kilo joules more when compared to the average of the two slave nodes. In all the three scenarios it can be seen that master node has the large amount of energy consumption when compared to the slave nodes as master node acts as both Name Node and Node Manager that is it handles different administrative functionalities as well as stores data in it.

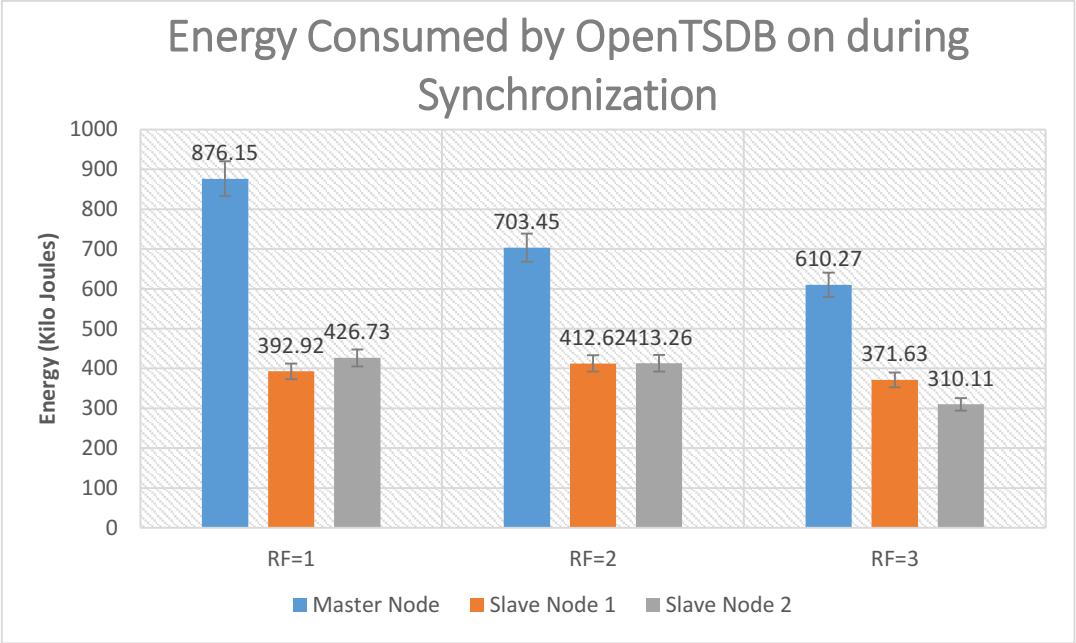

Figure 12: Energy consumed by OpenTSDB during Synchronization

Coming to the comparison of scenarios, the energy consumed by master node when the RF is 1 is more when compared to the other two scenarios. This is because in this scenario the master node experiences a higher number of processes for checking if any data is present in any of the slave nodes in the cluster. The energy difference between RF 1 to RF 2 for master node is approximately 173 kilo joules and between RF 2 to RF 3 is approximately 93 kilo joules. Similarly, the energy consumed by Slave node 1 is more when the RF is 2 and by slave node 2 is more when the RF is 1. The average energy consumed by database in the cluster when RF=1 is 565 kilo joules, RF= 2 is 510 kilo joules and RF= 3 is 431 kilo joules. It can be seen that while replicating data on all the N nodes in the cluster, RF= "N-2" consumes more energy when compared to RF= "N-1" and "N".

6.2.3 RQ3

a. Read

To measure the energy consumed by database on each node in the cluster two scenarios were considered: i. Multiple queries and ii. Continuous queries. The below graphs shows the energy consumed by each node when queries are requested from Master node, Slave node1 and Slave node2. Let us consider the querying from master node as "case 1" and querying from slave node 1 as "case 2" and from slave node 2 as "case3". Considering both scenarios, when queries are requested from any node that is in all the cases it can be seen that the energy consumption is always high for master node. This can be explained as the master node performs all the processing functionalities of the cluster.

i. Multiple Queries

The figure 13 shows the energy consumed by each node when multiple queries are requested from Master node, Slave node1 and Slave node2. When each node is compared in all the three scenarios the results vary a lot. So while comparing the energy consumed by master node in "case 1", "case 2" and case3 the energy consumption in "case 3"is 12.9 kilo joules more than in "case 2" and 193.3 kilo joules more than "case 1". This can be explained as whenever you query data from master node a lot of more energy is consumed than you query from the slave nodes. This is because the master

node processes functions such as synchronization of the cluster and then responds to the query requested. Coming to the slave node 1, while comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 23.2 kilo joules more than in “case 2”. Similarly, the energy consumption is 92.9 kilo joules more than “case 1”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 55.64 kilo joules more than in “case 2” and 101.1 kilo joules more than “case 1”. On an average in the cluster OpenTSDB consumes an average of 149 kilo joules on each node when querying is done from master node. Similarly, an average of 248 kilo joules and 278 kilo joules is consumed by OpenTSDB on each node when querying is done from slave node 1 and slave node 2. It can be said that while creating a single API call for multiple queries it is always better to create it from the master node for OpenTSDB instead of the slave nodes as the energy consumed during slave nodes is more than the master node.

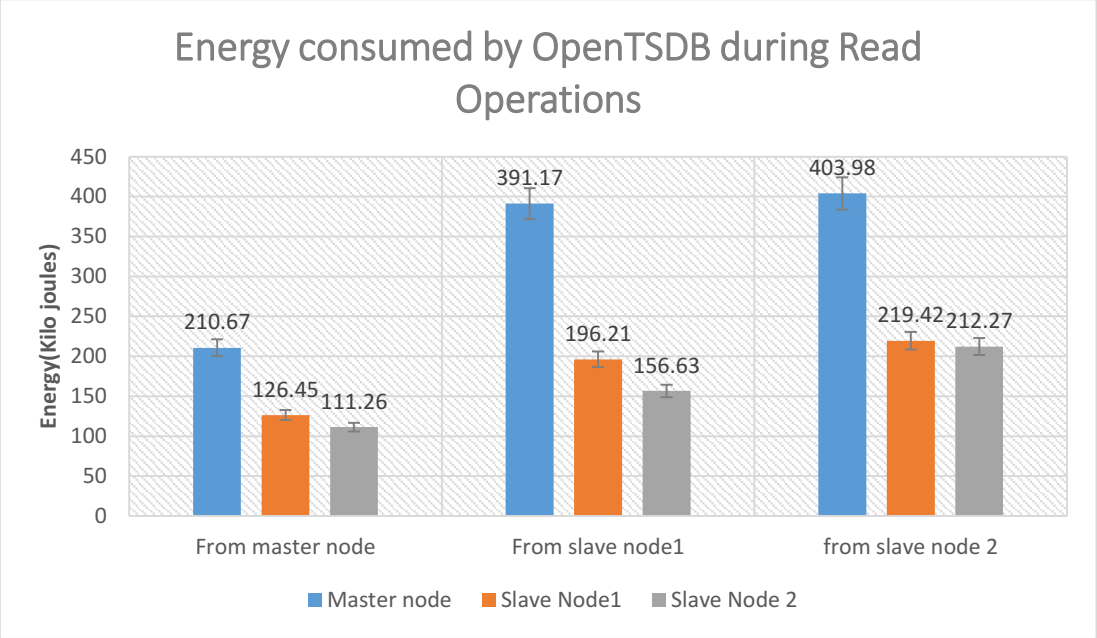

Figure 13:Energy consumed by OpenTSDB Multiple Queries (READ)

ii. Grouping

The figure 14, shows the energy consumed by each node when grouping of measurements is done and queries are requested from Master node, Slave node1 and Slave node2.When a query for grouping is created in “case 1” and requested from Master node, the energy consumed is more when compared to other cases. While comparing the energy consumed by master node in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 1” is 185.15 kilo joules more than “case 2” and 226.22 kilo joules more than “case 3”. While comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 1” is 186.6 kilo joules more than “case 2” and 119.8 kilo joules more than “case 2”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 1” is 100.27 kilo joules more than “case 2” and 118.9 kilo joules more than “case 3”.

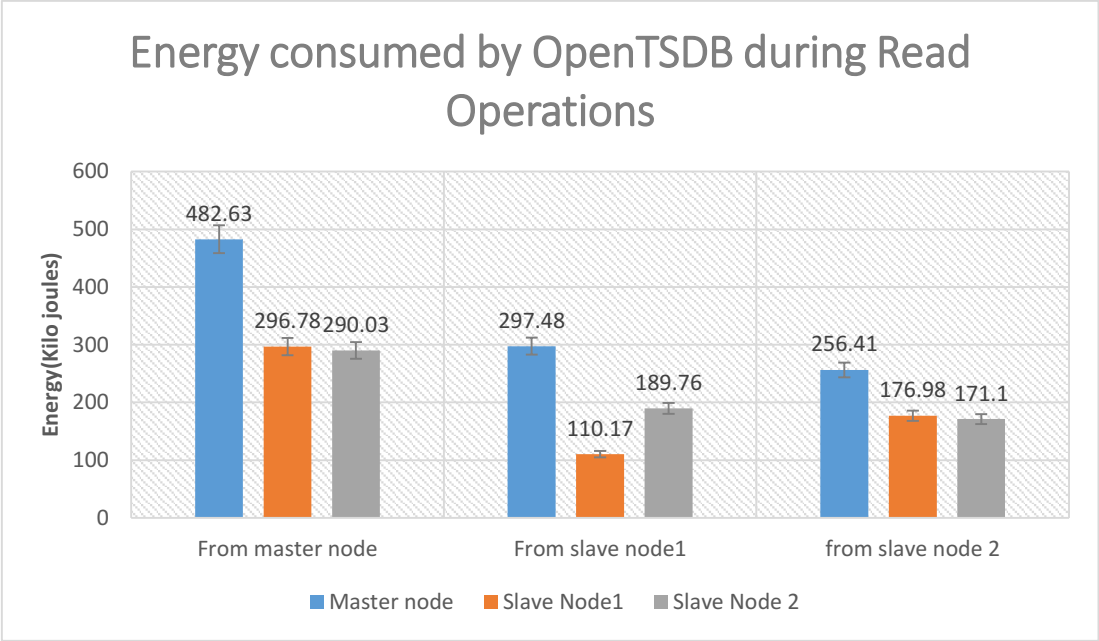

Figure 14: Energy consumed by OpenTSDB during Continuous Queries(READ)

On an average in the cluster OpenTSDB consumes an average of 356.48 kilo joules on each node when querying is done from master node. Similarly, an average of 199.1 kilo joules and 201.49 kilo joules is consumed by OpenTSDB on each node when querying is done from slave node 1 and slave node 2. It can be said that while creating a query for grouping it is always better to create it from the slave nodes instead of the master node for OpenTSDB as the energy consumed during master node is more than the slave nodes.

b. Write

To measure the energy consumed by database while inserting data into OpenTSDB two scenarios were considered i. Multiple insert queries and ii. Importing from file. The below graphs shows the energy consumed by each node when insert queries are requested from Master node, Slave node1 and Slave node2. Let us consider the querying from master node as “case 1” and querying from slave node 1 as “case 2” and from slave node 2 as “case 3”. Considering both scenarios, when queries are requested from any node that is in all the cases it can be seen that the energy consumption is always high for master node. This can be analyzed as the master node performs all the processing functionalities of the cluster.

i. Multiple Queries

The figure 15, shows the energy consumed by each node when Multiple insert queries are requested from Master node, Slave node1 and Slave node2.When a single API is created for multiple insert queries in “case 1”, the energy consumed is low when compared to other cases. While comparing the energy consumed by master node in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 123.93 kilo joules more than “case 1” and 89.84 kilo joules more than “case 2”. While comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 86.64 kilo joules more than “case 1” and 44.07 kilo joules more than “case 2”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 3”is 18.88 kilo joules more than “case 2” and 130.79 kilo joules more than “case 1”.

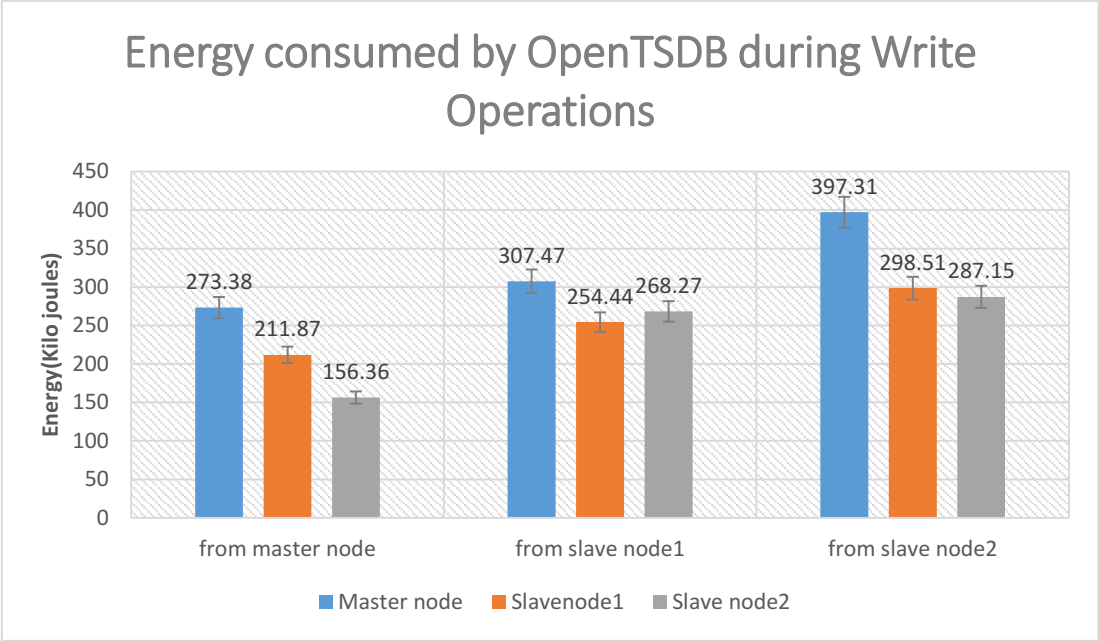

Figure 15: Energy consumed by OpenTSDB during Multiple Queries (WRITE)

On an average in the cluster OpenTSDB consumes an average of 214 kilo joules on each node when querying is done from master node. Similarly, an average of 277 kilo joules and 328 kilo joules is consumed by OpenTSDB on each node when inserting is done from slave node 1 and slave node 2. It can be said that while Inserting multiple data points it is always better to insert it from the master node instead of the slave nodes for OpenTSDB as the energy consumed during master node is less than the slave nodes.

ii. Import from file

Figure 16, shows the energy consumed by each node when a Single data file is inserted from Master node, Slave node1 and Slave node2.When a Single data file with multiple data points is inserted in “case 1”, the energy consumed is high when compared to “case 2” and “case 3”. While comparing the energy consumed by master node in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 1” is 140.2 kilo joules more than “case 2” and 151.26 kilo joules more than “case 2”. While comparing the energy consumed by slave node 1 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 1” is 80.08 kilo joules more than “case 2” and 158.7 kilo joules more than “case 3”. Similarly, while comparing the energy consumed by slave node 2 in “case 1”, “case 2” and “case 3”it can be seen that the energy consumption in “case 1” is 116.24 kilo joules more than “case 2” and 163.2 kilo joules more than “case 3”. On an average in the cluster OpenTSDB consumes an average of 448.7 kilo joules on each node when querying is done from master node. Similarly, an average of 336.4 kilo joules and 318.7 kilo joules is consumed by OpenTSDB on each node when inserting is done from slave node 1 and slave node 2.

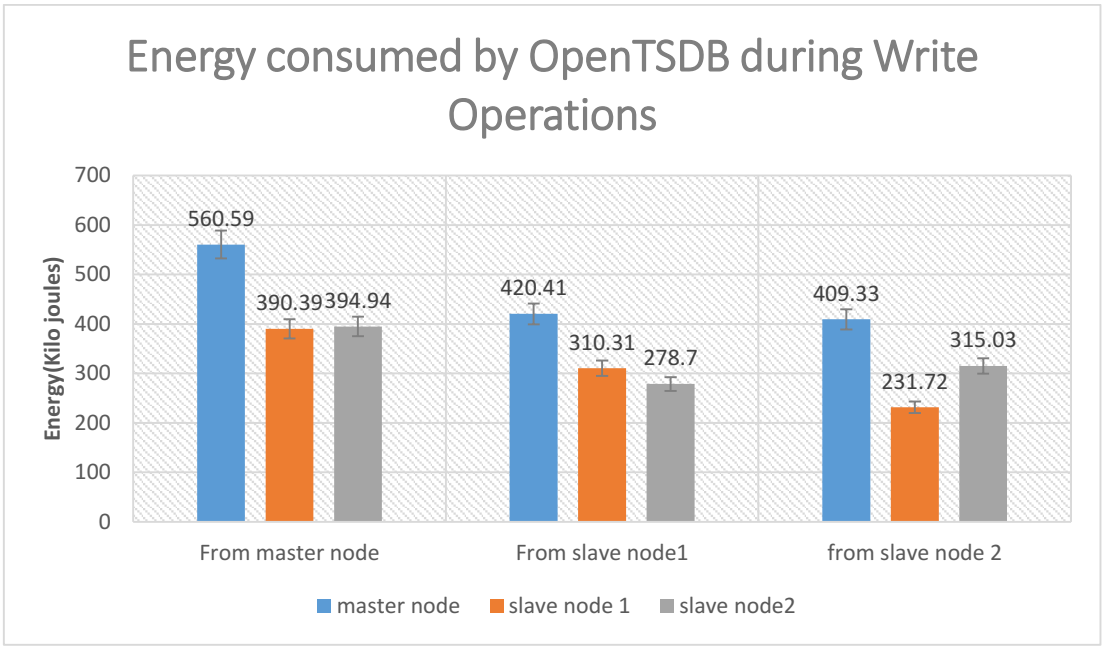

Figure 16: Energy consumed by OpenTSDB during Inserting File

It can be said that while importing a file it is always better to import a file from the slave nodes instead of the master nodes for OpenTSDB as the energy consumed during master node is more than the average of slave nodes.

6.3 Comparison between InfluxDB and OpenTSDB

6.3.1 RQ1

The figure 14, shows the energy consumed by each database on each node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node the difference for both the databases is approximately 45.78 kilo joules. Similarly, for slave node 1 and slave node 2 are 66.64 kilo joules and 86.98 kilo joules. On an average in the cluster, OpenTSDB consumes an average of 75 kilo joules on each node in ideal conditions. Similarly, an average of 19.26 kilo joules of energy is consumed by InfluxDB on each node after installations.

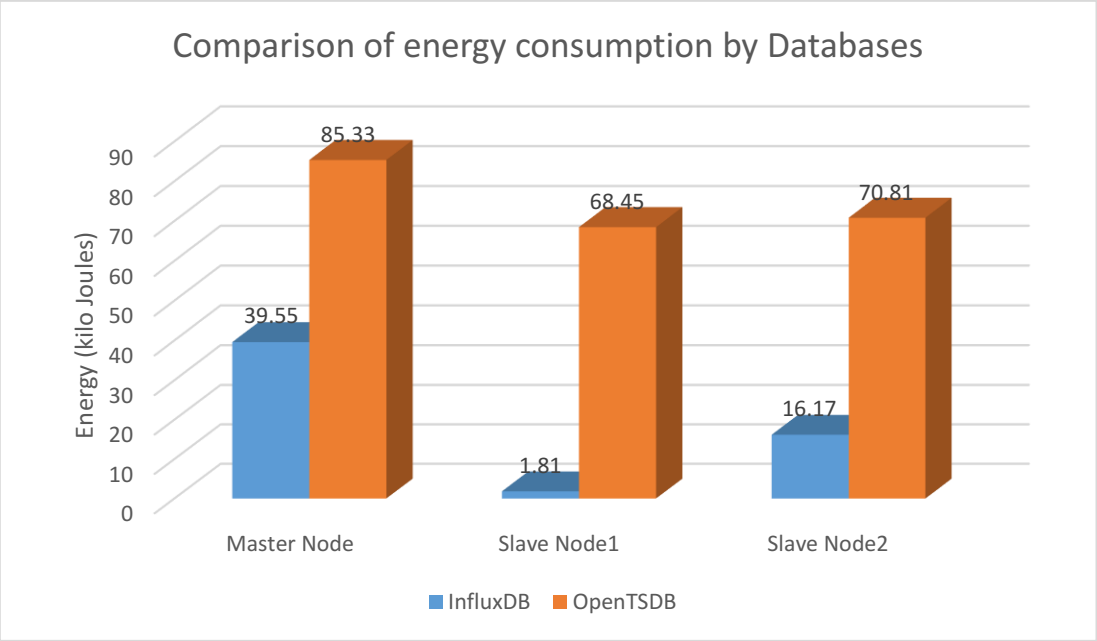

Figure 17: Comparison of Energy consumption by Databases in Scenario 1

From the figure 17, it can be analyzed that energy consumption by InfluxDB is quite less on each node in the cluster when compared to OpenTSDB. As both are effectively running Time series databases, regarding their performance analysis based on energy consumption it can be seen that InfluxDB consumes less energy than OpenTSDB when installed in a cluster of 3 nodes.

6.3.2 RQ2

6.3.2.1 Replication Factor is 1

The figure 18, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node the difference for both the databases is approximately 288 kilo joules. Similarly, for slave node 1 and slave node 2 are 182 kilo joules and 194 kilo joules. On an average in the cluster OpenTSDB consumes an average of 565 kilo joules on each node when the replication factor is 1. Similarly, an average of 343 kilo joules of energy is consumed by InfluxDB.

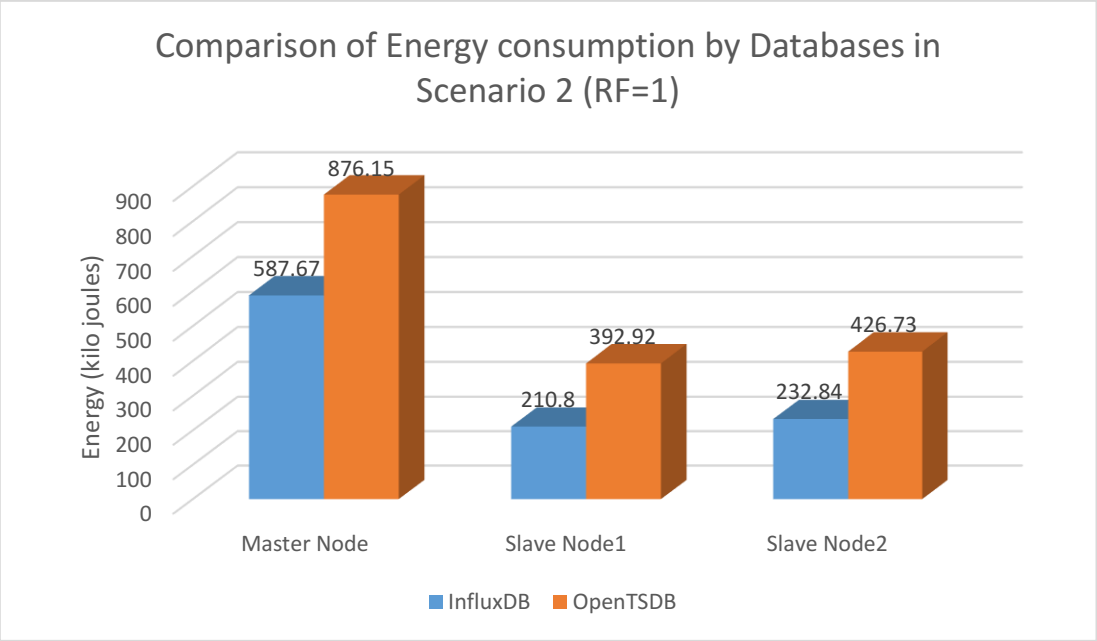

Figure 18: Comparison of Energy consumption by Databases in Scenario 2 (RF=1)

6.3.2.2 Replication Factor is 2

The figure 19, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node the difference for both the databases is approximately 429 kilo joules. Similarly, for slave node 1 and slave node 2 are 288 kilo joules and 291 kilo joules. On an average in the cluster OpenTSDB consumes an average of 509 kilo joules on each node when the replication factor is 2. Similarly, an average of 173 kilo joules of energy is consumed by InfluxDB.

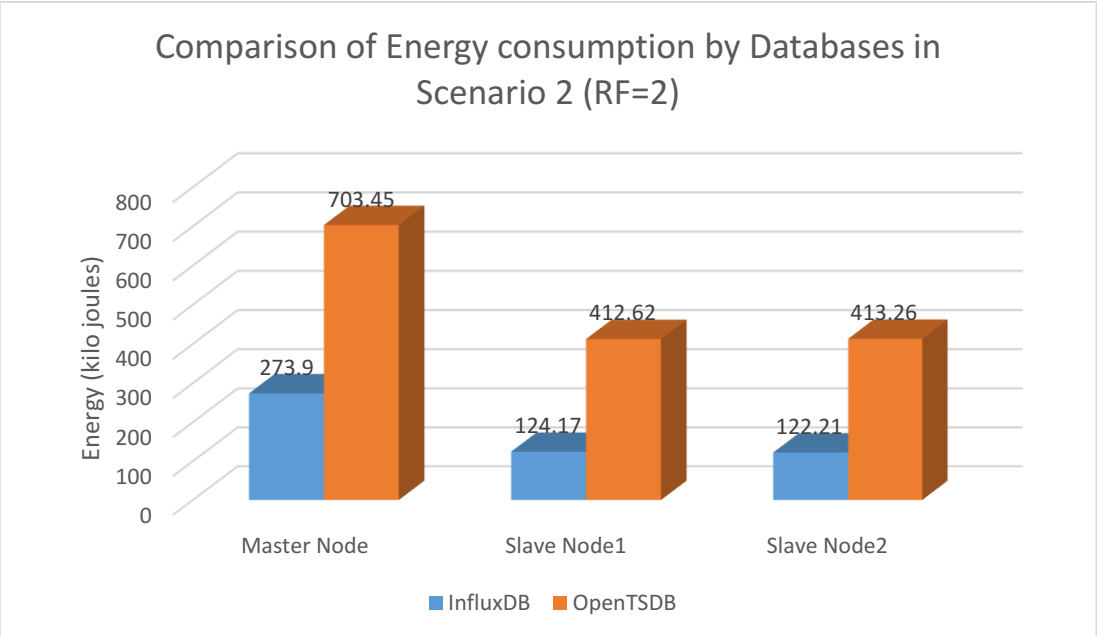

Figure 19: Comparison of Energy consumption by Databases in Scenario 2 (RF=2)

6.3.2.3 Replication Factor is 3

The figure 20, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB compared to InfluxDB. Coming to the comparison of Master node the difference for both the databases is approximately 340 kilo joules. Similarly, for slave node 1 and slave node 2 are 247 kilo joules and 192 kilo joules. On an average in the cluster, OpenTSDB consumes an average of 430 kilo joules on each node when the replication factor is 3. Similarly, an average of 170 kilo joules of energy is consumed by InfluxDB.

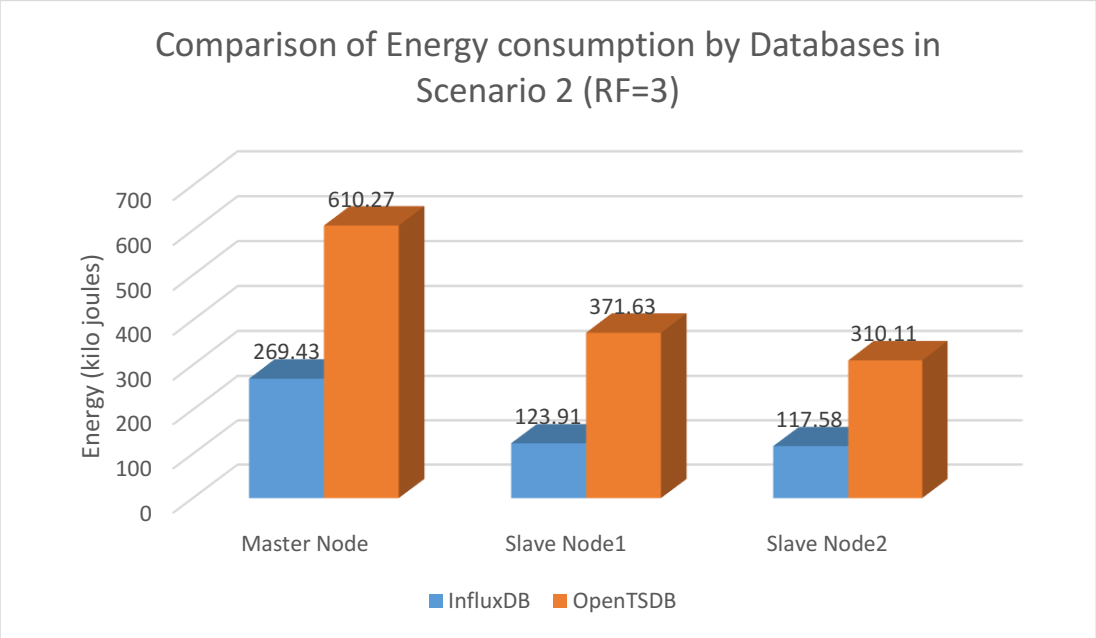

Figure 20: Comparison of Energy consumption by Databases in Scenario 2 (RF=3)

From the above graphs in all the cases it can be analyzed that energy consumption by InfluxDB is quite less on each node in the cluster when compared to OpenTSDB. As both are effectively running Time series databases, regarding their performance analysis based on energy consumption it can be seen that InfluxDB consumes less energy than OpenTSDB when synchronization among multiple hosts is considered.

6.3.3 RQ3

a. Read

To measure the energy consumed by database on each node in the cluster two scenarios were considered i. Multiple queries and ii. Continuous queries. The below graphs shows the energy consumed by each node when queries are requested from Master node, Slavenode1 and Slavenode2.

i. Multiple queries

Let us consider the querying from master node as “case 1” and querying from slave node 1 as “case 2” and from slave node 2 as “case 3”.

1. From Master Node (case1)

The figure 21, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when read operations are carried out from Master node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB compared to InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 6.5 kilo joules. Similarly, for slave node 1 and slave node 2 are 5.39 kilo joules and 5 kilo joules.

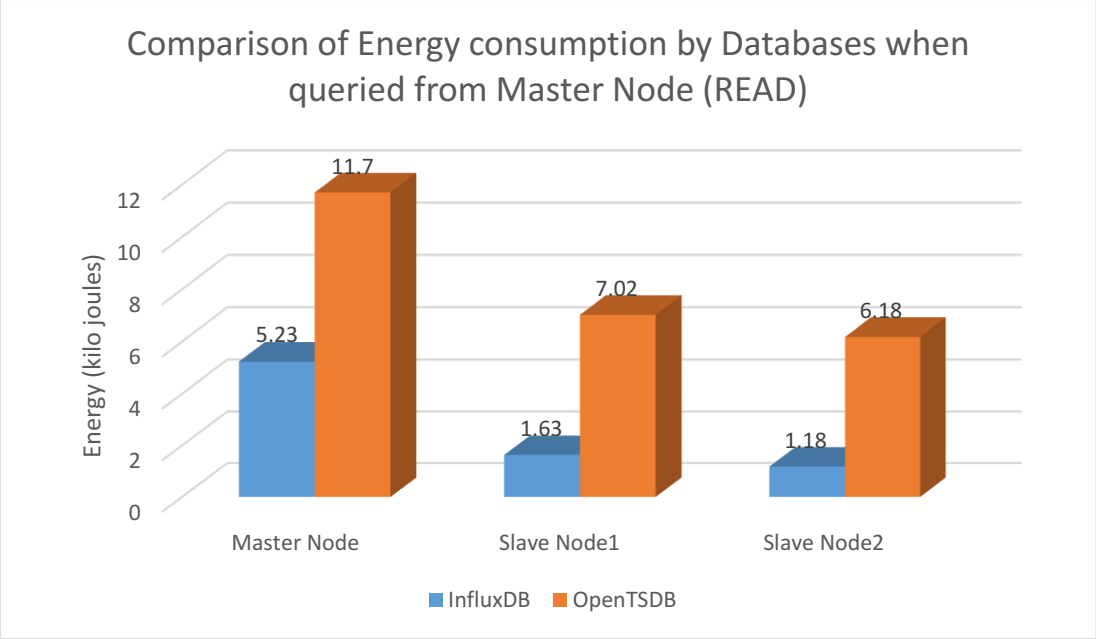

Figure 21: Comparison of Energy consumption by Databases when queried from Master Node (READ)

On an average in the cluster OpenTSDB consumes an average of 8 kilo joules on each node when a single data point is queried from the Master Node. Similarly, an average of 3 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

2. From Slave Node 1 (“case 2”)

The figure 22, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when read operations are carried out from Master node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Mater node, the difference for both the databases is approximately 17.6 kilo joules. Similarly, for slave node 1 and slave node 2 are 9.4 kilo joules and 7.25 kilo joules. On an average in the cluster OpenTSDB consumes an average of 14 kilo joules on each node when a single data point is queried from the Slave Node 1. Similarly, an average of 2 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

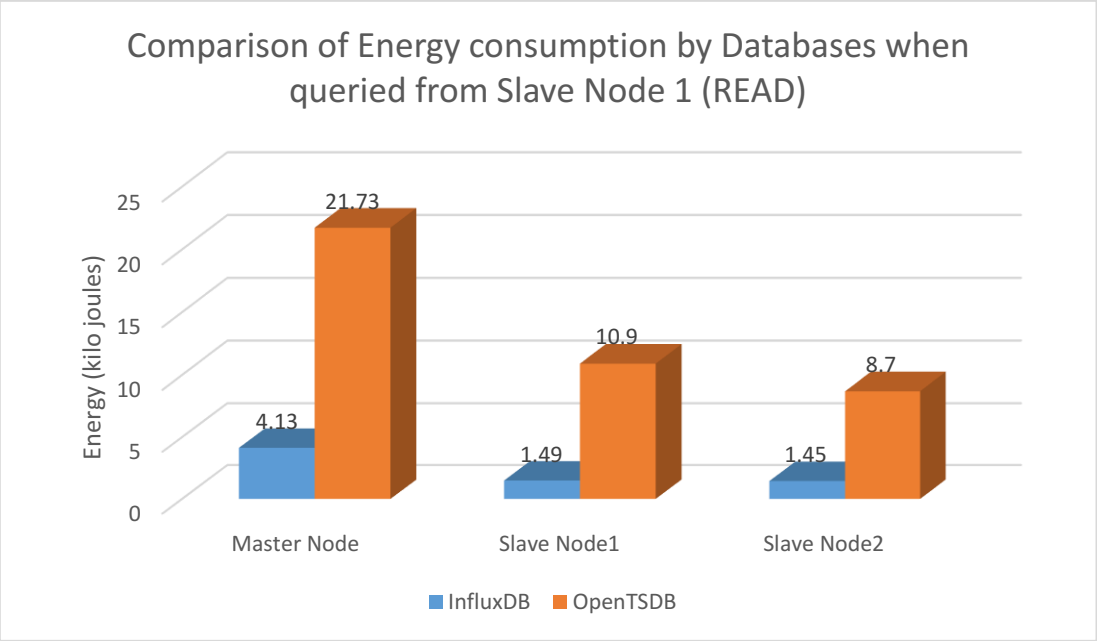

Figure 22: Comparison of Energy consumption by Databases when queried from Slave Node 1 (READ)

3. From Slave Node 2 (“case 3”)

The figure 23, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when read operations are carried out from Slave Node 2 in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Mater node, the difference for both the databases is approximately 17.6 kilo joules. Similarly, for slave node 1 and slave node 2 are 10.87 kilo joules and 10.73 kilo joules.

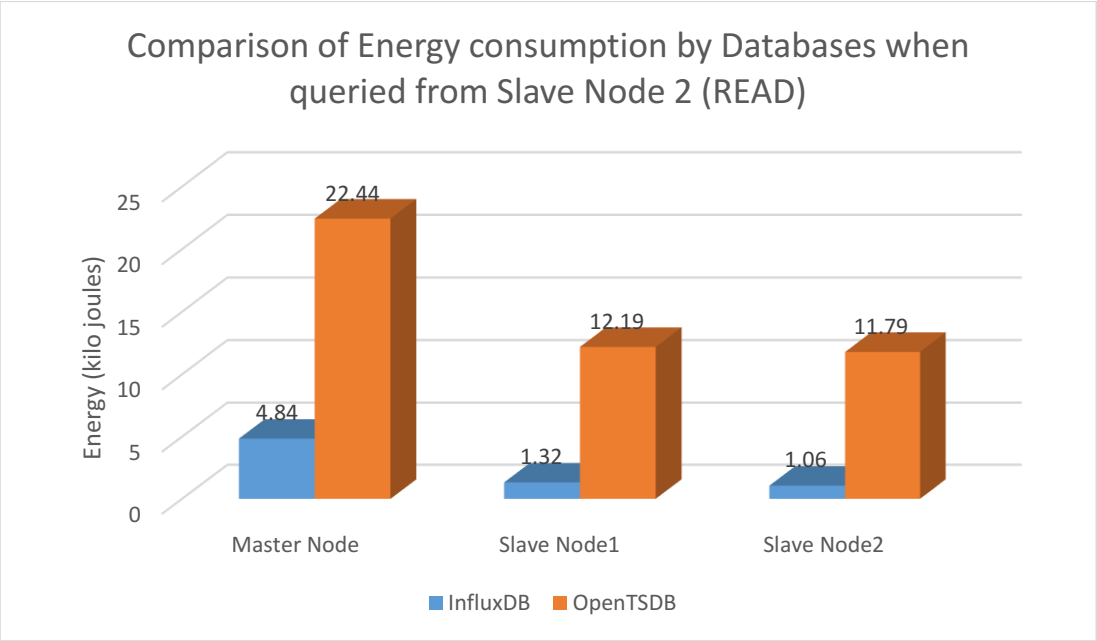

Figure 23: Comparison of Energy consumption by Databases when queried from Slave Node 2 (READ)

On an average in the cluster OpenTSDB consumes an average of 15 kilo joules on each node when a single data point is queried from the Slave Node 2. Similarly, an average

of 2 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

From the above graphs in all the cases it is clearly observed that OpenTSDB consumes more energy when read operations are carried out. Considering "case 2" and "case 3" we can say that it is always better to query from the master node instead of salve nodes as the energy consumed while querying from slave nodes is quite high in any database.

ii. Continuous queries or Grouping

Let us consider the querying from master node as "case 1" and querying from slave node 1 as "case 2" and from slave node 2 as "case 3".

1. From Master Node

The figure 24, shows the energy consumed by both the databases, InfluxDB and OpenTSDB on each node when Continuous queries or Grouping of measurements are requested from Master Node. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Mater node, the difference for both the databases is approximately 22.8 kilo joules. Similarly, for slave node 1 and slave node 2 are 14.92 kilo joules and 15.17 kilo joules.

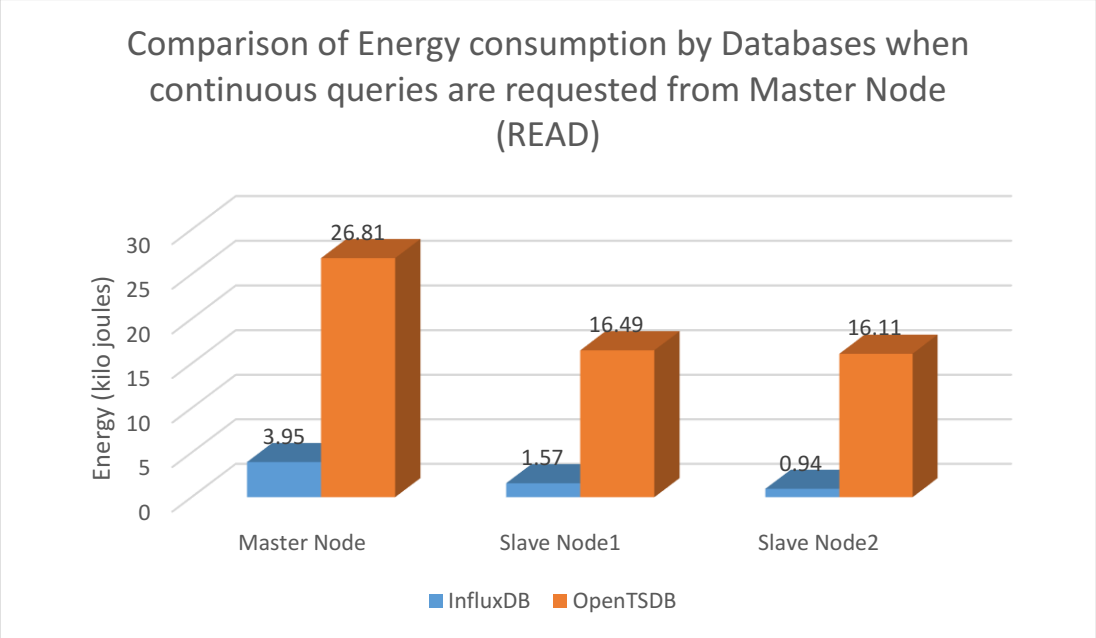

Figure 24: Comparison of Energy consumption by Databases when continuous queries are requested from Master Node (READ)

On an average in the cluster OpenTSDB consumes an average of 19 kilo joules on each node when a single data point is queried from the Master Node. Similarly, an average of 2 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

2. From Slave Node 1

The figure 25, shows the energy consumed by both the databases, InfluxDB and OpenTSDB on each node when Continuous queries or Grouping of measurements are requested from Slave Node 1. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Mater node, the

difference for both the databases is approximately 11.55 kilo joules. Similarly, for slave node 1 and slave node 2 are 4.24 kilo joules and 9.39 kilo joules.

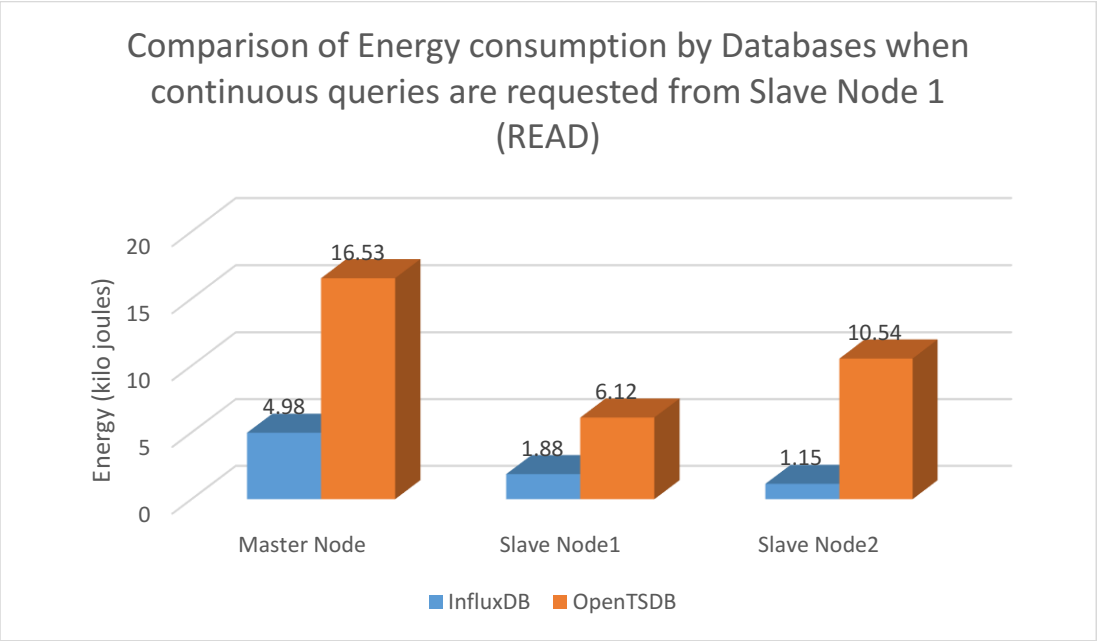

Figure 25: Comparison of Energy consumption by Databases when continuous queries are requested from Slave Node 1 (READ)

On an average in the cluster OpenTSDB consumes an average of 11 kilo joules on each node when a single data point is queried from the Slave Node 1. Similarly, an average of 3 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

3. From Slave Node 2

The figure 26, shows the energy consumed by both the databases, InfluxDB and OpenTSDB on each node when Continuous queries or Grouping of measurements are requested from Slave Node 2. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Mater node, the difference for both the databases is approximately 9.58 kilo joules. Similarly, for slave node 1 and slave node 2 are 7.9 kilo joules and 7.9 kilo joules. On an average in the cluster OpenTSDB consumes 11 kilo joules on each node when a single data point is queried from the Slave Node 1. Similarly, an average of 3 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

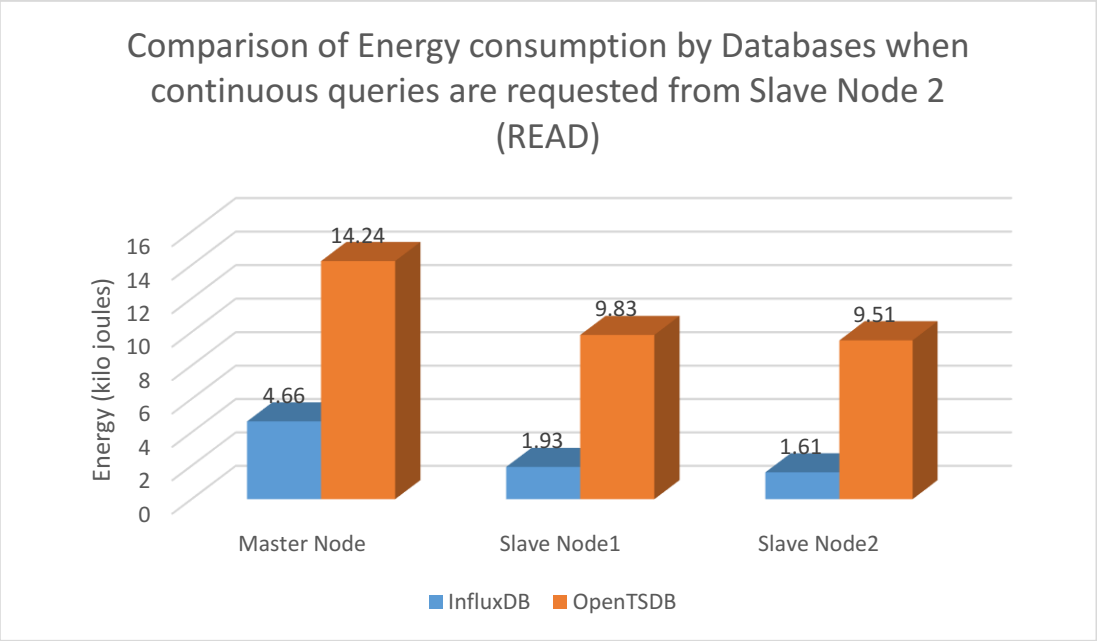

Figure 26: Comparison of Energy consumption by Databases when continuous queries are requested from Slave Node 2 (READ)

From the above graphs in all the cases it is clearly observed that OpenTSDB consumes more energy when Continuous queries or grouping is done. Considering "case 2" and "case 3" we can say that it is always better to query from the slave nodes instead of master node as the energy consumed while querying from slave nodes is low in any database.

b. Write

To measure the energy consumed by database on each node in the cluster two scenarios were considered i. Multiple queries and ii. Import from File. The below graphs shows the energy consumed by each node when queries are requested from Master node, Slavenode1 and Slavenode2.

i. Multiple queries

Let us consider the querying from master node as "case 1" and querying from slave node 1 as "case 2" and from slave node 2 as "case 3".

1. From Master Node

The figure 27, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when multiple write operations are carried out from Master node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 14.67 kilo joules. Similarly, for slave node 1 and slave node 2 are 11.63 kilo joules and 8.53 kilo joules. On an average in the cluster OpenTSDB consumes an average of 12 kilo joules on each node when a single data point is queried from the Master Node. Similarly, an average of 0.3 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be inserted from the master node in both the databases.

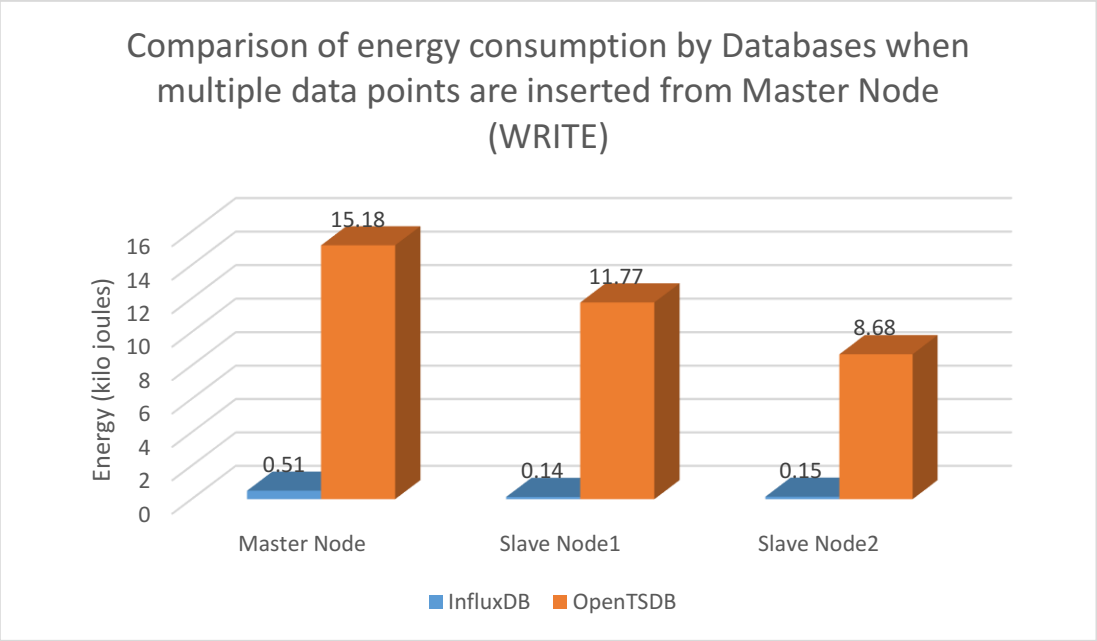

Figure 27: Comparison of energy consumption by Databases when multiple data points are inserted from Master Node (WRITE)

2. From Slave Node 1

The figure 28, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when multiple write operations are carried out from Slave Node 1 in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 15.96 kilo joules. Similarly, for slave node 1 and slave node 2 are 13.82 kilo joules and 14.76 kilo joules.

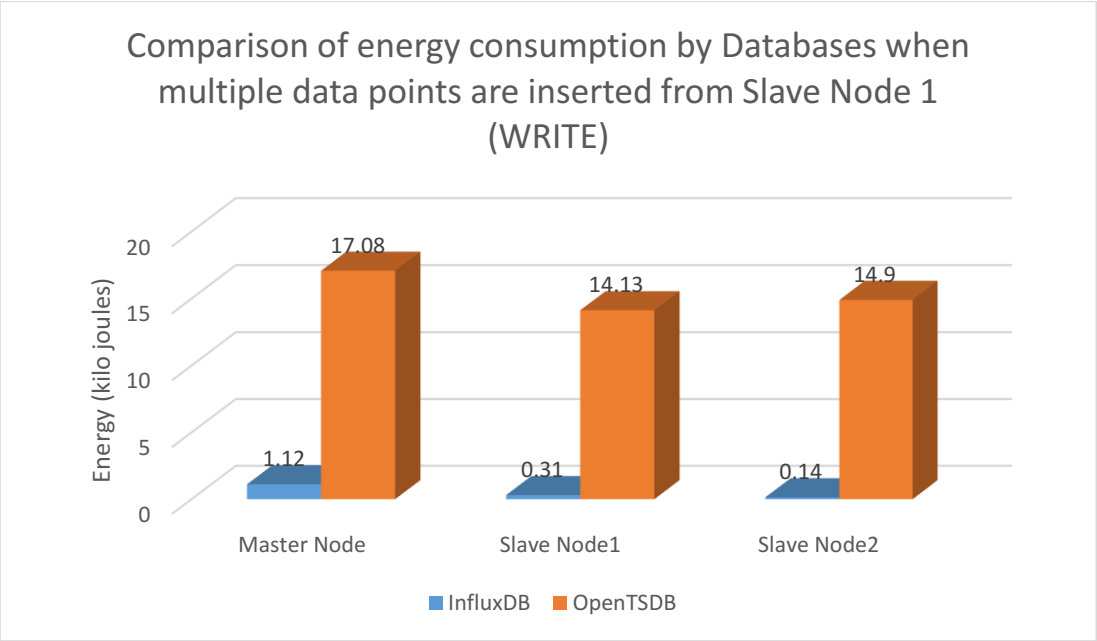

Figure 28: Comparison of energy consumption by Databases when multiple data points are inserted from Slave Node 1 (WRITE)

On an average in the cluster OpenTSDB consumes an average of 15 kilo joules on each node when a single data point is queried from the Slave Node 1. Similarly, an average

of 0.52 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be inserted from the slave node 1 in both the databases.

3. From Slave Node 2

The figure 29, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when multiple write operations are carried out from Slave node 2 in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 20.48 kilo joules. Similarly, for slave node 1 and slave node 2 are 16.19 kilo joules and 15.56 kilo joules.

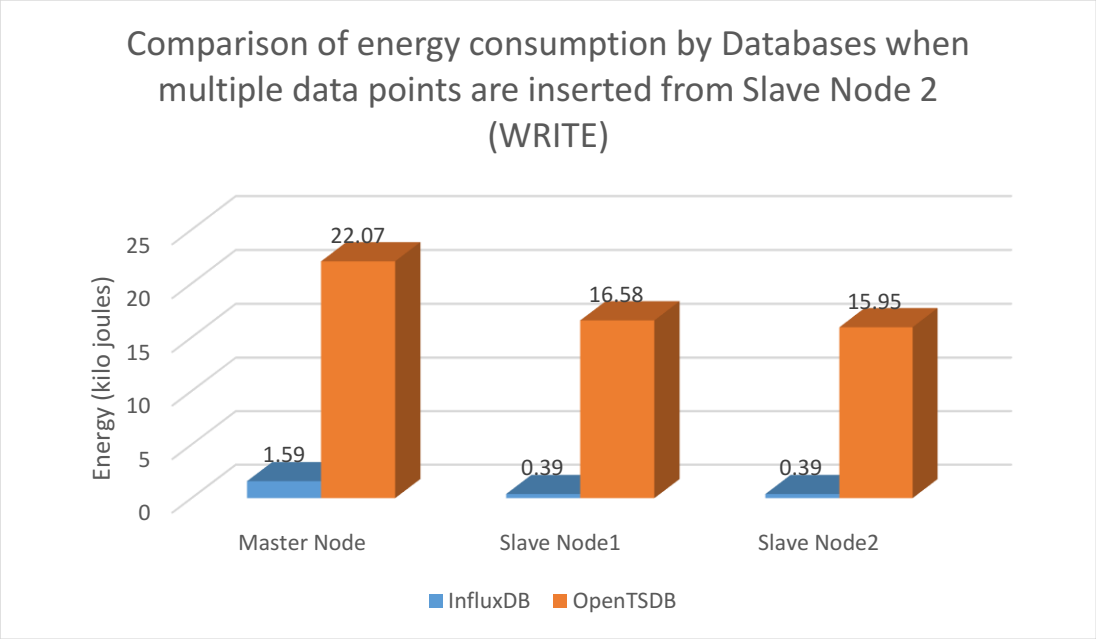

Figure 29: Comparison of energy consumption by Databases when multiple data points are inserted from Slave Node 2 (WRITE)

On an average in the cluster OpenTSDB consumes an average of 18 kilo joules on each node when a single data point is queried from the Master Node. Similarly, an average of 0.8 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be queried from the master node in both the databases.

From the above graphs in all the cases it is clearly observed that OpenTSDB consumes more energy when Continuous queries or grouping is done. Considering “case 2” and “case 3” we can say that it is always better to insert data from the Master node instead of slave nodes as the energy consumed while querying from slave nodes is low in any database.

ii. Import from File

Let us consider the querying from master node as “case 1” and querying from slave node 1 as “case 2” and from slave node 2 as “case 3”.

1. From Master Node

The figure 30, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when multiple data points in a file are imported from Master

node in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 6.5 kilo joules. Similarly, for slave node 1 and slave node 2 are 4.95 kilo joules and 5 kilo joules.

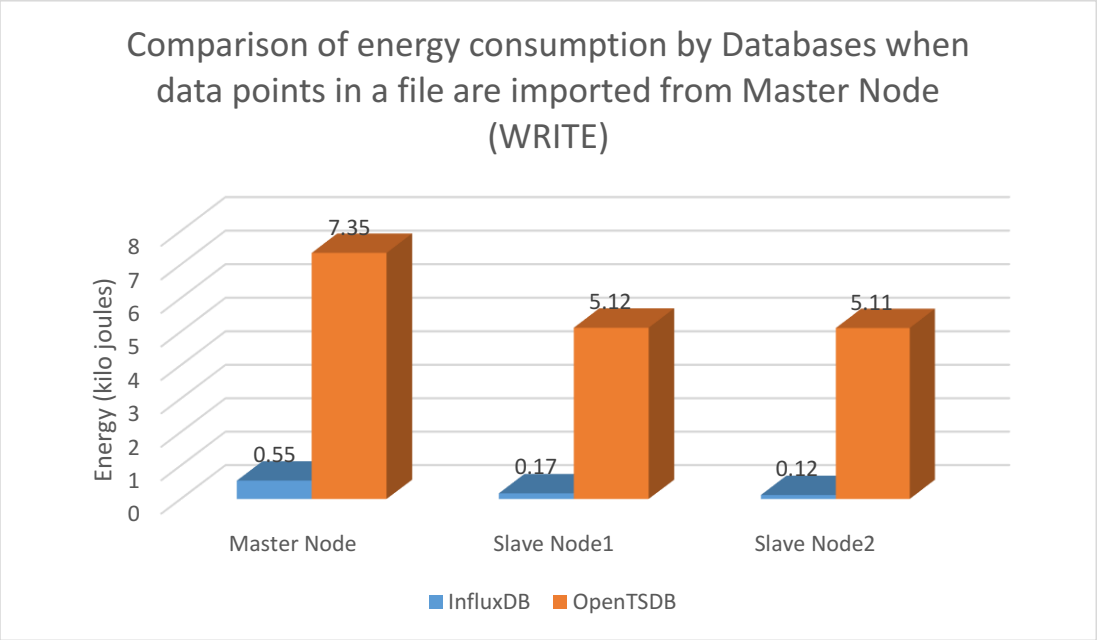

Figure 30: Comparison of energy consumption by Databases when data points in a file are imported from Master Node (WRITE)

On an average in the cluster OpenTSDB consumes an average of 6 kilo joules on each node when a single data point is inserted from the Master Node. Similarly, an average of 0.84 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be inserted from the master node in both the databases.

2. From Slave Node 1

The figure 31, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when multiple data points in a file are imported from Slave node 1 in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 5.19 kilo joules. Similarly, for slave node 1 and slave node 2 are 3.81 kilo joules and 3.49 kilo joules. On an average in the cluster OpenTSDB consumes an average of 4 kilo joules on each node when a single data point is inserted from the Slave Node 1. Similarly, an average of 0.24 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be inserted from the slave node 1 in both the databases.

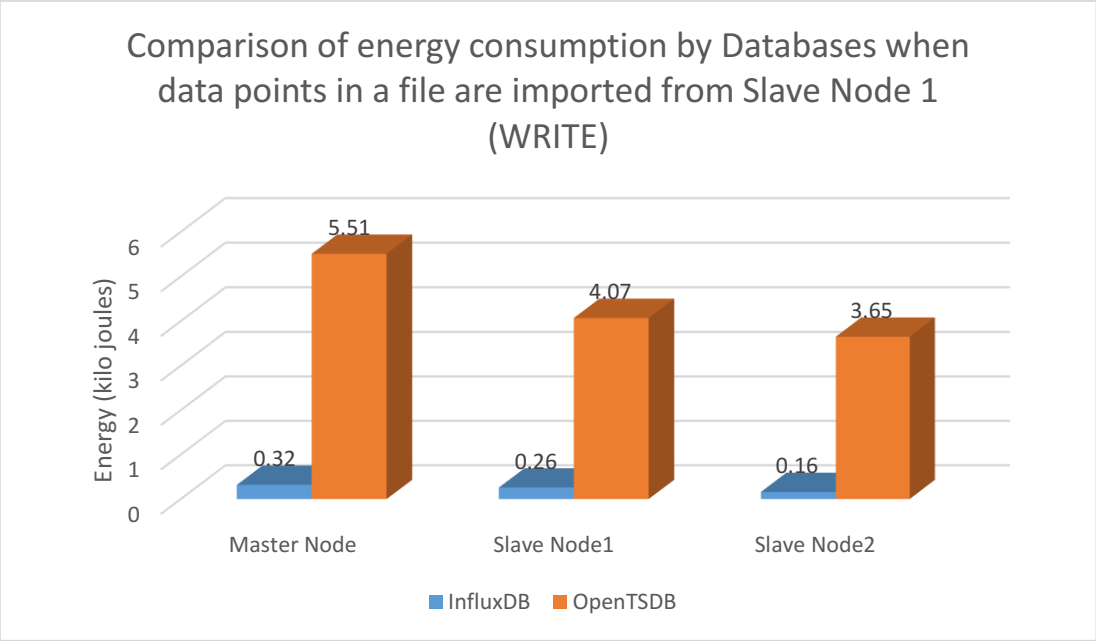

Figure 31: Comparison of energy consumption by Databases when data points in a file are imported from Slave Node 1 (WRITE)

3. From Slave Node 2

The figure 32, shows the average energy consumed by both the databases, InfluxDB and OpenTSDB on each node when multiple data points in a file are imported from Slave node 2 in the cluster. On all the nodes the energy consumption is quite high for OpenTSDB instead of InfluxDB. Coming to the comparison of Master node, the difference for both the databases is approximately 4.79 kilo joules. Similarly, for slave node 1 and slave node 2 are 2.89 kilo joules and 3.81 kilo joules.

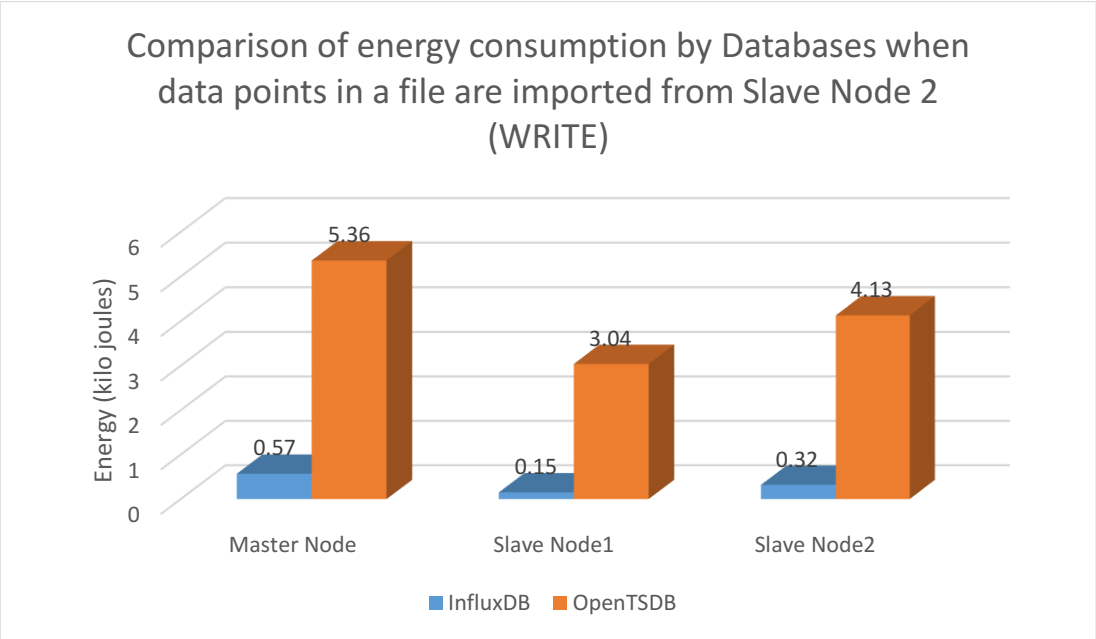

Figure 32: Comparison of energy consumption by Databases when data points in a file are imported from Slave Node 2 (WRITE)

On an average in the cluster OpenTSDB consumes an average of 4 kilo joules on each node when a single data point is inserted from the Slave Node 2. Similarly, an average

of 0.35 kilo joules of energy is consumed by InfluxDB. All the above measurements are considered for single data point to be inserted from the slave node 2 in both the databases.

From the above graphs in all the cases it is clearly observed that OpenTSDB consumes more energy when data points are inserted from a file. Considering “case 2” and “case 3” we can say that it is always better to import file from the slave nodes instead of master node as the energy consumed while querying from slave nodes is low in any database.

7 CONCLUSION AND FUTURE WORK

This section summarizes the Performance of two Time series databases i.e. OpenTSDB and InfluxDB based on the Energy Consumed by them in different scenarios. Further, objectives for future research work are also discussed.

7.1 Conclusion

The purpose of this research work is to evaluate the performance of InfluxDB and OpenTSDB based on the energy consumed by them during load and no load conditions. InfluxDB and OpenTSDB which are evolving distributed Time series databases. The evaluation of these databases is done in three different scenarios i.e. i) The average energy consumed by the database on each node in the cluster under no load conditions, ii) The average energy consumed by the database on multiple hosts under synchronization process, and iii) The average energy consumed by the database per read and write operation. Both InfluxDB and OpenTSDB are tested and experimented under same conditions in all the scenarios. An individual analysis is conducted for each database and a comparative analysis is done between the databases to evaluate which database consumes low energy. Individual analysis of InfluxDB and OpenTSDB are explained clearly in section 6.1.1 and 6.1.2 where the average energy consumed by them on each node in the cluster is analyzed in all the three scenarios. During the comparative analysis between the databases it is concluded that InfluxDB consumes less energy when compared to OpenTSDB in all the three scenarios. Based on the results obtained and analysis done, the following conclusions can be deduced:

- Research Question 1: How much energy is consumed on average by a node of the distributed time series database?
  - InfluxDB consumes less energy on each node when compared to OpenTSDB.
  - The master node consumes more energy when compared to the slave nodes for both InfluxDB and OpenTSDB.
- Research Question 2: How much energy is consumed by the database when synchronization among multiple nodes is considered?
  - InfluxDB consumes less energy on each node when compared to OpenTSDB when Synchronization among multiple nodes is considered.
  - The master node consumes more energy when compared to slave nodes for both InfluxDB and OpenTSDB in all the three replication cases mentioned in section 6.1.2InfluxDB consumes more energy during a replication factor of 3, which tells that energy is consumed more when data is synchronized with a replication factor 3.
  - OpenTSDB consumes more energy during a replication factor of 1, which tells that energy is consumed more when data is synchronized with a replication factor 1.
- Research Question 3: What is the estimated energy of a read and write operation, respectively, executed by the database?
  - In InfluxDB during read operations using multiple queries master node consumes more energy than the slave nodes, and read operations using continuous queries slave nodes consumes more energy than the master node.
  - In InfluxDB during write operations using multiple queries slave nodes consume more energy than the master node, and write operations while importing from a file, slave nodes consumes more energy than the master node.
  - In OpenTSDB during read and write operations using multiple queries slave nodes consume more energy than the master node.
  - In OpenTSDB during read operations using continuous queries master node consume more energy than the slave nodes.

- In OpenTSDB during write operations while importing from a file, master node consumes more energy than the slave nodes.

7.2 Future work

This thesis mainly focused on the energy consumed by the database in three conditions such as no load, synchronization and during read write operations. It would be interesting to find the energy consumed by the systems when different error conditions occur such as during the failure of a single node in the cluster during synchronization and etc. Identifying energy consumption during error conditions helps in obtaining a more complete view of the performance of the database. Investigating the energy consumed by the GUI tools, which execute graphs on the host, are used by the databases to provide graphs would also be an interesting concept which can help the real time users to differentiate between the energy consumed by the database itself and the GUI tools individually.

REFERENCES

[1] B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V. Sharma, "Cloud computing for Internet of Things amp; sensing based applications," in *2012 Sixth International Conference on Sensing Technology (ICST)*, 2012, pp. 374–380.

[2] T. A. M. Phan, "Cloud Databases for Internet-of-Things Data," Master's Thesis, Aalto University, Denmark, 2013.

[3] Z. Jian-Hua and Z. Nan, "Cloud computing-based data storage and disaster recovery," in *Future Computer Science and Education (ICFCSE), 2011 International Conference on*, 2011, pp. 629–632.

[4] M. Kaveh, "ETL and Analysis of IoT data using OpenTSDB, Kafka, and Spark," 2015.

[5] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, "Process-level power estimation in VM-based systems," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, p. 14.

[6] F. F. Moghaddam, M. B. Rohani, M. Ahmadi, T. Khodadadi, and K. Madadipouya, "Cloud computing: Vision, architecture and Characteristics," in *2015 IEEE 6th Control and System Graduate Research Colloquium (ICSGRC)*, 2015, pp. 1–6.

[7] P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.

[8] J. Horrigan, *Use of cloud computing applications and services*. Pew Internet & American Life Project, 2008.

[9] B. E. S. J. D. Owusu, "A Survey on Software as a Service (Saas) Cloud for Programming Language Computing."

[10] B. Dorsemaine, J.-P. Gaulier, J.-P. Wary, N. Kheir, and P. Urien, "Internet of Things: A Definition & Taxonomy," in *ResearchGate*, 2015.

[11] "InfluxData Documentation." [Online]. Available: https://docs.influxdata.com/. [Accessed: 20-Jul-2016].

[12] E. A. Brewer, "Towards robust distributed systems," in *PODC*, 2000, vol. 7.

[13] S. Fogel and P. Lane, *Oracle Database Administrator's Guide*. Oracle, 2005.

[14] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[15] R. Want, "An introduction to RFID technology," *IEEE Pervasive Comput.*, vol. 5, no. 1, pp. 25–33, 2006.

[16] K. Kaur and R. Rani, "Modeling and querying data in NoSQL databases," in *Big Data, 2013 IEEE International Conference on*, 2013, pp. 1–7.

[17] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, 2011, pp. 363–366.

[18] C. Strozzi, *Nosql: A non-sql relational database management system*. 2013.

[19] "NOSQL Databases." [Online]. Available: http://nosql-database.org/. [Accessed: 19-Jul-2016].

[20] E. Lai, "No to SQL? Anti-database movement gains steam," *Computerworld Softw. July*, vol. 1, 2009.

[21] C. Strauch, "Nosql databases. Lecture selected topics on software-technology ultra-large scale sites," *Manuscr. Stuttg. Media Univ.*, 2011.

[22] N. Leavitt, "Will NoSQL databases live up to their promise?," *Computer*, vol. 43, no. 2, pp. 12–14, 2010.

[23] M. Andreolini, M. Colajanni, and M. Pietri, "A scalable architecture for real-time monitoring of large information systems," in *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, 2012, pp. 143–150.

[24] "Apache HBase – Apache HBase™ Home." [Online]. Available: http://hbase.apache.org/. [Accessed: 19-Jul-2016].

[25] B. Agrawal, "Analysis of large time-series data in OpenTSDB," 2013.

[26] "Installation — OpenTSDB 2.2 documentation." [Online]. Available: http://opentsdb.net/docs/build/html/installation.html. [Accessed: 19-Jul-2016].

[27] A. Carpen-Amarie, A.-C. Orgerie, and C. Morin, “Experimental Study on the Energy Consumption in IaaS Cloud Environments,” in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, 2013, pp. 42–49.
[28] A. Beloglazov, “Energy-efficient management of virtual machines in data centers for cloud computing,” 2013.
[29] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” 2007.
[30] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, “Analyzing the energy efficiency of a database server,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 231–242.
[31] A. Botta, W. de Donato, V. Persico, and A. Pescapé, “Integration of cloud computing and internet of things: a survey,” *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, 2016.
[32] O. Bernard, J. Sainte-Marie, B. Sialve, and J.-P. Steyer, “Hydrodynamics-Biology Coupling for Algae Culture and Biofuel Production,” *ERCIM News*, vol. 92, p. 47, 2013.
[33] T. Goldschmidt, A. Jansen, H. Koziolek, J. Doppelhamer, and H. P. Breivold, “Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes,” in *2014 IEEE 7th International Conference on Cloud Computing*, 2014, pp. 602–609.
[34] A. TaheriMonfared, T. W. Wlodarczyk, and C. Rong, “Real-time handling of network monitoring data using a data-intensive framework,” in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, 2013, vol. 1, pp. 258–265.
[35] T. Sun, J. Liu, and L. Feng, “Improving the index structure with hierarchical techniques in time-series databases,” in *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, 2010, vol. 5, pp. 2089–2094.
[36] S. Harizopoulos, M. Shah, J. Meza, and P. Ranganathan, “Energy efficiency: The new holy grail of data management systems research,” *ArXiv Prepr. ArXiv09091784*, 2009.

8 APPENDIX

Networking of InfluxDB:

By default, InfluxDB uses the following network ports:

- TCP port 8083 is used for InfluxDB's Admin panel
- TCP port 8086 is used for client-server communication over InfluxDB's HTTP API

Installation of InfluxDB:

The latest and stable version of InfluxDB can be acquired from the official website [23], for the respective Operating Systems. In this thesis Ubuntu 14.04.3 LTS is used and the commands used to install InfluxDB are as follows:

"wget https://dl.influxdata.com/influxdb/releases/influxdb_0.13.0_amd64.deb"

"sudo dpkg -i influxdb_0.13.0_amd64.deb"

To start the InfluxDB service the following command needs to be executed in the terminal:

"sudo service influxdb start"

InfluxDB provides us an influx command line interface (CLI), which is included in all InfluxDB packages and is a lightweight and simple way to interact with the database. The CLI communicates with InfluxDB directly by making requests to the InfluxDB HTTP API over port 8086 by default. Using InfluxDB we can create a database, write and explore data into it etc. To access the CLI, launch the command "influx" in your terminal. Once you are successfully entered the influx shell, there are several arguments that we can pass into influx when starting. They can be listed with the following command "influx –help" [23].

Bitwatts installation

The following steps are performed for the installation of Bitwatts:

1. Download and extract the latest release of Bitwatts tool from GitHub.
2. Install the latest version Java runtime environment
3. Open 'bitwatts.conf' file of the conf directory- 'bitwatts-cli.1.0'
4. Add text "powerapi.cpu.tdp=x", where x is the respective TDP power of the hardware used.
5. To get the TDP power, check the configuration of the hardware (node).

To measure the energy consumed by a particular process the following command needs to be executed in the 'bin' directory of 'bitwatts-cli.1.0'.

"*. /bitwatts modules cpu-simple monitor –frequency {required frequency} –targets {respective process name} –agg {required aggregate type} –file {file name} duration {required time}*"

READ Operations:

The syntax of the read query is as follows:

"curl -G 'http://IPADDRESS:8086/query?pretty=true' --data-urlencode "db= DATABASE_NAME" --data-urlencode "q=SELECT value FROM MEASURMENT_NAME

WHERE TAGKEY_NAME='TAG_VALUES'; SELECT value FROM MEASURMENT_NAME WHERE TAGKEY_NAME='TAG_VALUES'"”

The syntax of the read query is as follows:

“curl -i -XPOST 'http:// IPADDRESS:8086/write?db= DATABASE_NAME ' --data-binary ‘MEASURMENT_NAME =TAGKEY_NAME value=TAG_VALUE’”

Installations of OpenTSDB:

The procedure or the prerequisites for the installation of OpenTSDB in a cluster mode are:
A Hbase 3 node cluster is installed and configured with Hadoop distributed file system as the storage unit. The configuration of the cluster needs to be done in fully distributed mode. Zookeeper needs to be accessible and Hbase needs to be up and running. Bitwatts command mentioned in section 4.2 is executed on each node accordingly such that power consumed by all the processes in that node is measured for a duration of 30 minutes. As the database OpenTSDB depends mainly on Hbase and HDFS, the energy consumed by it on a node is the sum of all the three processes Hbase, HDFS, and OpenTSDB. Basic steps to keep OpenTSDB running are like creating tables with the Hbase instance and starting the TSD. The average energy consumed by the database on each node after installation is calculated by the above mentioned procedure.

The syntax of the read query is as follows:

“GET http://IPADDRESS:4242/api/query?start=TIME_STAMP&m=sum:METRIC_NAME{TAG_NNAME}”

The syntax of the read query is as follows:

“GET http://IPADDRESS:4242/api/query?start=TIME_STAMP&m=sum:METRIC_NAME{TAG_NAMNA=*}”

The syntax of the read query is as follows:

“PUT http://IPADDRESS:4242/api/put?TYPE=METRIC_NAME&q=TAG_NAMES=TAG_VALUES”