ENERGY EFFICIENCY ANALYSIS AND OPTIMIZATION OF RELATIONAL AND NOSQL DATABASES

by

Divya Mahajan

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
December 2016

Committee Members:

Ziliang Zong, Chair

Yijuan Lu

Guowei Yang

COPYRIGHT

By

Divya Mahajan

2016

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

DEDICATION

Dedicated to my family, whose support and encouragement during my graduate studies motivated me to complete this thesis.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Ziliang Zong for his guidance, support and feedback. Also, I would like to extend my appreciation for my thesis committee Dr. Yijuan Lu, and Dr. Guowei Yang.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATIONS

Abbreviation	Description

CPU - Central Processing Unit

GPU - Graphic Processing Unit

YCSB - Yahoo Cloud Server Benchmark

DVFS - Dynamic Voltage and Frequency Scaling

DRAM - Dynamic Random Access Memory

ABSTRACT

As big data becomes the norm of various industrial applications, the complexity of database workloads and database system design has increased significantly. To address these challenges, conventional relational databases have been constantly improved and NoSQL databases such as MongoDB and Cassandra have been proposed and implemented to compete with SQL databases. In addition to traditional metrics such as response time, throughput, and capacity, modern database systems are posing higher requirements on energy efficiency due to the large volume of data that need to be stored, queried, updated, and analyzed. While decades of research in the database and data processing communities has produced a wealth of literature that optimize for performance, research on optimizations for energy efficiency has been historically overlooked and only very few studies have investigated the energy efficiency of database systems. To the best our knowledge, currently no comprehensive studies analyze the impact of query optimizations on performance and energy efficiency across both SQL and NoSQL databases. In fact, the energy behavior of many basic database operations (e.g. insertion, deletion, searching, update, indexing, etc) remains largely unknown due to the lack of accurate power measurement methodologies for various databases and

queries. In this thesis, we developed a tool that can accurately measure the real-time power consumption of queries running on both SQL and NoSQL databases and investigated a series of query optimization techniques for improving the energy-efficiency of both Relational Databases and NoSQL Parallel databases. We used both widely acceptable benchmarks (e.g. Yahoo! Cloud Server Benchmark) and customized datasets (converted from 100GB of Twitter data) in our experiments to evaluate the effectiveness of optimization techniques. We performed cross database analysis on SQL based database (MySQL) and NoSQL based databases (MongoDB and Cassandra) to compare their performance and energy efficiency. Additionally, we studied a variety of optimization techniques that can improve energy efficiency without compromising performance on the databases derived from the Twitter data. Using these techniques, we were able to achieve significant energy savings without performance degradation.

1. INTRODUCTION

Energy efficiency is an emerging critical design and operational criteria for computing environments that includes data centers, small clusters, and even stand-alone servers. Database Management Systems (DBMSs) running in server environments have largely ignored energy efficiency, but we can no longer afford such oversight. For example, Google currently processes over 2.5 million queries per minute which means that the rapid accumulation of power required for these queries not only cost money but also resources [1]. Today, People express their opinions and views on Twitter and emerging events or news are often followed almost instantly by a burst in Twitter volume, which provides a unique opportunity to gauge the relation between expressed public sentiment. Therefore, Twitter has become another exemplary big dataset where many social media analytics tools (e.g. sentiment analysis [2]) are being used to determine attitude of people towards a product, idea, and so on. However, analyzing such humungous volume of data with accuracy and efficiency is very costly thus requires the databases to be highly efficient in terms both performance and energy efficiency.

The goal of this research is to study optimization techniques that can harness high performance in an energy efficient way. Retrieving information promptly and cost-effectively from massive amount of data stored in a large-scale database opens a wide range of research issues that percolate through nearly all aspect of a DBMS, including

query evaluation strategies, query optimization, query scheduling, physical database design, and dynamic workload management.

The current research and practices on databases emphasizes more on performance than energy efficiency. Fallacies and misconceptions abound due to the lack of research on database energy efficiency. For example, many database researchers believe that energy optimization is merely a byproduct of performance optimization while other researchers argue that performance optimization and energy optimization are conflicting goals (i.e. performance needs to be sacrificed to save energy or vice versa) [29]. The research questions we would like to answer include 1) Is performance efficiency equivalent to energy efficiency? Will there be a win-win situation for both performance and energy consumption? 2) What are the correlations of performance, power and energy when optimizing databases? How to identify these correlations? The research goal of this thesis is to investigate the performance-energy tradeoff by finding answers to these questions and revealing the correlations of performance, power and energy on optimizing databases. We also explore optimization techniques used in relational as well as parallel databases [3] to make them more energy efficient without degrading performance.

To evaluate a diverse set of query optimization techniques on the aforementioned databases, we develop a tool on the NSF funded Marcher system that can accurately measure the real-time power consumption of various queries running on MySQL,

MongoDB and Cassandra databases. The queries generated by the above databases are submitted to the Marcher system as an executable file. The Marcher system and power measurement methodology will be discussed in detail in Chapter 4.

For each experiment, we evaluate the performance and energy efficiency of two queries running over the same dataset where one query utilizes optimization techniques and other without any optimizations. We execute each query numerous times to make sure outliers are eliminated. Finally, we use the Greenup, Powerup and Speedup metrics (Please refer to Chapter 4 Section 4.2.1 for definitions) to analyze the results. We use both widely acceptable benchmarks (e.g. Yahoo! Cloud Server Benchmark) and customized datasets (converted from 100GB of Twitter data) in our experiments to evaluate the effectiveness of optimization techniques.

The major contributions of this thesis are summarized below:

1) We conduct a comprehensive study (first of its kind to the best of our knowledge) on various databases namely MySQL, MongoDB and Cassandra to study the optimization techniques to improve performance as well as energy efficiency.

2) We develop a tool that can accurately measure the real-time power consumption of various queries running on MySQL, MongoDB and Cassandra databases.

3) We present a methodology using Greenup, Powerup and Speedup to reveal the correlations between performance, power and energy efficiency of the databases.

4) We perform cross database comparison using both the Yahoo! Cloud Server Benchmark (YCSB) [4] and the customized Twitter datasets to evaluate the performance and energy efficient of MySQL, MongoDB and Cassandra at different scenarios.

5) We study the impact of DVFS [5,6] on the energy consumption of databases.

The rest of the thesis is organized as follows. In Chapter 2, we present the literary review of the research related to energy efficiency of databases. Chapter 3 provides an overview of relational and NoSQL databases. In Chapter 4, we analyze the system, metrics and benchmarks used for power measurement and database efficiency evaluation. In Chapter 5, we study optimization techniques for performance and energy efficiency of MySQL. In Chapter 6, we analyze and study techniques to optimize queries running on MongoDB using the Twitter data. In Chapter 7, we analyze and study techniques to optimize queries running on Cassandra. In Chapter 8, we extend our analysis towards relational database MySQL and provide various techniques for achieving high performance and power efficiency. In Chapter 9, we conduct a comparison analysis of all three databases. Finally, chapter 10 summarizes this work, draws conclusions and discusses future research directions.

2. RELATED WORK

The power consumption in databases or green databases has just started drawing attention from the research community. The 2008 Claremont Report [7] suggested energy-aware databases as a promising research topic. Various other topics such as energy quantification of database servers, benchmarking, cost-based query plan evaluation are also reported. Tsirogiannis et al. identified factors in databases that had an important impact on power consumption [8]. Lang et al. investigated the design of energy efficient DBMS clusters in [11]. Zu et al. provided insights on redesigning the DBMS kernel for power-saving purposes [12]. Based on these results, they provided suggestions on how to make the database system more power efficient. Subramaniam and Feng et al. studied the energy proportionality of servers in the context of a distributed NoSQL data store and measured the power consumption and performance of a Cassandra cluster using power and resource provisioning techniques [13].

Previous studies primarily focused on how to improve database energy efficiency by modifying the hardware. For example, Schall et al. proposed to use solid state disks (SSDs) instead of magnetic disks (HDDs) [14] to store data. Graefe et al. suggested using memory devices instead of rotating disks, reducing RAM by using hash memory, and enabling or disabling memory banks to save energy. Despite the opportunities to reduce energy consumption through hardware, the work claimed that the best way is through

database server software. Graefe et al. discussed further ways to reduce energy consumption, such as data compression, I/O scheduling and placement, and parallelism [15]. A few studies have explored using software approaches (e.g. optimizing queries, modifying resource patterns, and/or managing storage) to improve energy efficiency [9,16,17,18,19] of databases. Goncalves et al. implemented two techniques, Processor Voltage and Frequency Control (PVC) and Improved Query Energy-efficiency by Introducing Explicit Delays (QED), on MySQL, which improved energy efficiency but also notably increased the response time [16]. These techniques and studies have shown that the mechanisms used to conserve energy often compromise performance [19, 21]. Other studies that do recognize the tradeoffs between performance and energy have only addressed the issue by setting a static response time goal and finding the optimal energy efficient query plan to balance it [19,20]. For example, Xu et al. designed a basic power aware query optimizer that picks the best query plan based on performance and power consumption [21]. Xu et al. also developed the power-aware throughput control (PAT) mechanism that reduced energy usage by 51.3% [17]. However, Tsirogiannis et al. concluded that optimizing performance and optimizing energy efficiency are similar goals that can be done without having to fix any variables. The work shows that the tradeoff between energy efficiency and performance exists because studies erroneously do not take into consideration peripheral components' power and idle CPU power [10].

The previous literature has provided numerous suggestions on how to optimize databases for better performance and energy efficiency. However, the majority of them focused on upgrading to the newest hardware, redesigning the database management system (DBMS), or optimizing traditional SQL based relational databases.

This research is orthogonal to previous literature because we aim to investigate optimizations and techniques at the software level that can improve the energy efficiency of databases. In addition, we evaluate both traditional relational database systems as well as NoSQL parallel database systems. To the best of our knowledge not much investigation has been done to study techniques for improving energy efficiency of NoSQL parallel databases. This thesis provides a number of innovative insights from this new perspective.

3. INTRODUCTION TO RELATIONAL AND NOSQL DATABASES

In today's world, it is almost impossible to think of any application that does not make use of databases. From simple games to business-related tools, including web sites, certain type(s) of data is processed, recorded, and retrieved with each operation. Database Management Systems (DBMS) are the higher-level software, working with lower-level application programming interfaces (APIs), that take care of these operations. To help with solving different kind of problems, new kinds of DBMSs have been developed for decades. It includes relational and non-relational or NoSql databases along with applications implementing them (e.g. MySQL, PostgreSQL, MongoDB, Redis, etc). In this chapter, we will discuss about architecture and various features of relational and NoSql databases [34].

3.1 Overview of Relational databases

In this section, we briefly introduce relational databases and their features.

Relational databases have dominated the software industry for a long time providing mechanisms to store data persistently, concurrency control, transactions, mostly standard interfaces and mechanisms to integrate application data, reporting. A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many ways without having to reorganize the database tables.

In other words, it is a set of tables containing data fitted into predefined categories. Each

table (which is sometimes called a relation) contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns. When a database is described as relational, it has been designed to conform to a set of practices called the rules of normalization. A normalized database is one that follows the rules of normalization. For example, in an organization, there are employees who work in specific departments. Each employee and department has a number and a name. This information can be organized in a table as shown in Table 3-1.

Table 3-1: Sample Employee Information

EmpNo	Ename	DeptNo	DeptName
101	Bob	10	Marketing
102	David	20	Purchase
103	Evelyn	10	Marketing
104	Doug	30	Sales

If we structure data this way and make certain changes to it, there will be few problems. For example, deleting all the employees in the Purchasing department will eliminate the department itself. Using the principles of normalized relational databases, we can eliminate these problems by restructuring Employee and Department data in Table 3-1 into two separate tables (DEPT and EMP), as shown in Tables 3-2 and 3-3.

Table 3-2: A Sample Relational DEPT Table

DeptNo	DeptName
10	Marketing
20	Purchase
30	Sales

Table 3-3: A Sample Relational EMP Table

EmpNo	EmpName
101	Bob
102	David
103	Elevyn
104	Doug

3.1.1 MySQL

MySQL is a database query language designed for the retrieval and management of data in a relational database. SQL stands for Structured Query Language. The scope of SQL includes data insert, query, update, delete, schema creation and data access control. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.

3.2 Overview of NoSql Databases

Over the last few years we have seen the rise of a new type of databases, known as NoSQL databases. NoSql databases are challenging the dominance of relational databases. NoSQL means Not Only SQL, implying that when designing a software solution or product, there are more than one storage mechanism that could be used based on the needs. NoSQL does not have a prescriptive definition but there are a set of common observations which can help define it.

- Not using the relational model
- Running well on clusters

- Mostly open-source
- Schema-less

The rise of the web as a platform also created a vital factor change in data storage as the need to support large volumes of data by running on clusters. Also, relational databases were not designed to run efficiently on clusters. NoSQL databases can broadly be categorized in four types.

1. Key-Value databases:

Key-value stores are the simplest NoSQL data stores to use from an API perspective. The client can either get the value for the key, put a value for a key, or delete a key from the data store. The value is a blob that the data store just stores, without caring or knowing what's inside; it's the responsibility of the application to understand what was stored. Structure of key-value database is shown in Figure 3-1. Since key-value stores always use primary-key access, they generally have great performance and can be easily scaled. Some of the popular key-value databases are Riak, Redis and CouchBase.

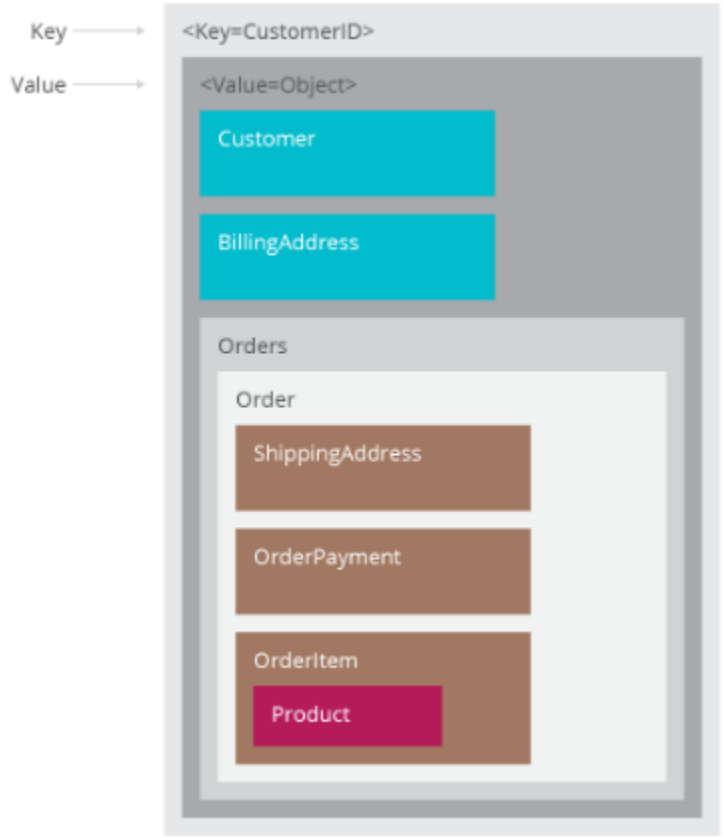

Figure 3-1: Key-Value Databases

2. Document databases:

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. Some of the popular document databases we have seen are MongoDB, CouchDB and Terrastore.

3. Column family stores:

Column-family databases store data in column families as rows that have many columns

associated with a row key as shown in Figure 3-2. Column families are groups of related data that is often accessed together. Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows. Some of the popular column family stores are Cassandra and HBase.

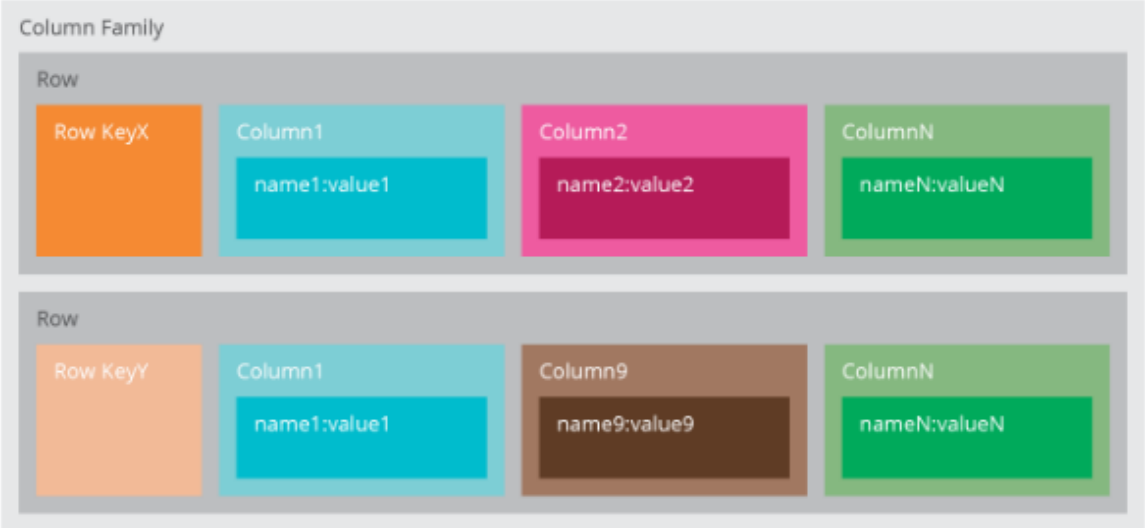

Figure 3-2: Column Family Stores

4. Graph Databases:

Graph databases allow you to store entities and relationships between these entities. Entities are also known as nodes, which have properties. In graph databases, traversing the joins or relationships is very fast. The relationship between nodes is not calculated at query time but is persisted as a relationship. Traversing persisted relationships is faster than calculating them for every query. Some of the popular graph databases are Infinite

Graph, FlockDB and Neo4j.

In this thesis, we will limit our experiments to two of the most popular NoSQL databases MongoDB and Cassandra to investigate various optimization techniques which helps in gaining energy efficiency.

3.2.1 MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. scaling. MongoDB organizes its data in the following hierarchy: database, collection and document. A database is a set of collections and a collection is a set of documents. Collections are analogous to tables in relational databases. Unlike a table, however, a collection does not require its documents to have the same schema. The organization of data in MongoDB is shown in Figure3-3.

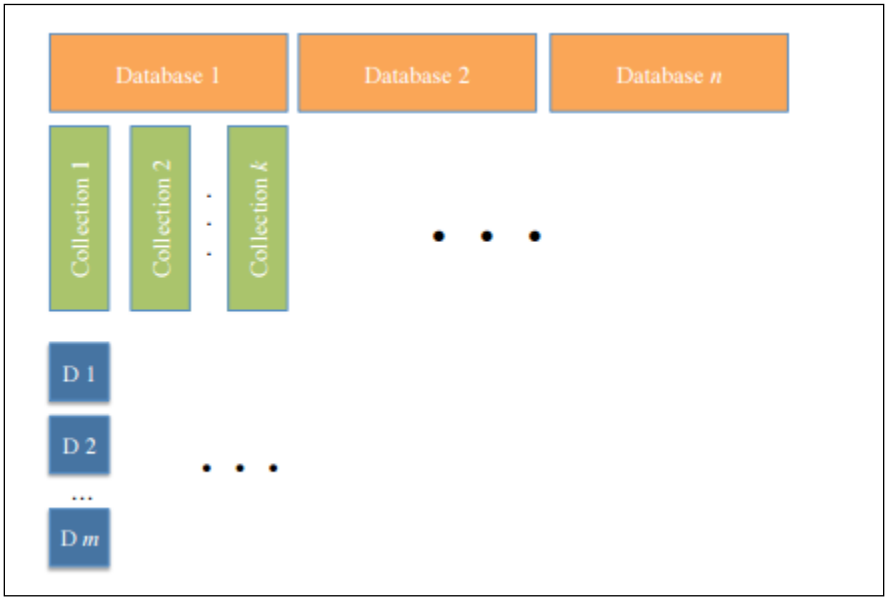

Figure 3-3: Organization of data in MongoDB

A record in MongoDB is a document, which is a data structure composed of field and value pairs. The values of fields may include other documents, arrays, and arrays of documents. MongoDB documents are like JSON objects. Sample JSON object is depicted in Figure 3-4.

Figure 3-4: JSON Object

3.2.2 Cassandra

Apache Cassandra is a free and open-source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It is a column family database that stores data in column families as rows that have many columns. In Cassandra, all nodes play an identical role; there is no concept of a master node, with all nodes communicating with each other via a distributed, scalable protocol called "gossip." To improve availability, each data item can be replicated at N different hosts, where N is the replication factor. Cassandra's built-for-scale architecture means that it is capable of handling large amounts of data and thousands of concurrent users. To add more capacity, new nodes can be added to an existing cluster.

Data is written to Cassandra in a way that provides both full data durability and high performance. Data written to a Cassandra node is first recorded in an on-disk commit log and then written to a memory-based structure called a memtable. When a memtable's size exceeds a configurable threshold, the data is written to an immutable file on disk called an SSTable. Figure 3-5 shows how data is written in Cassandra.

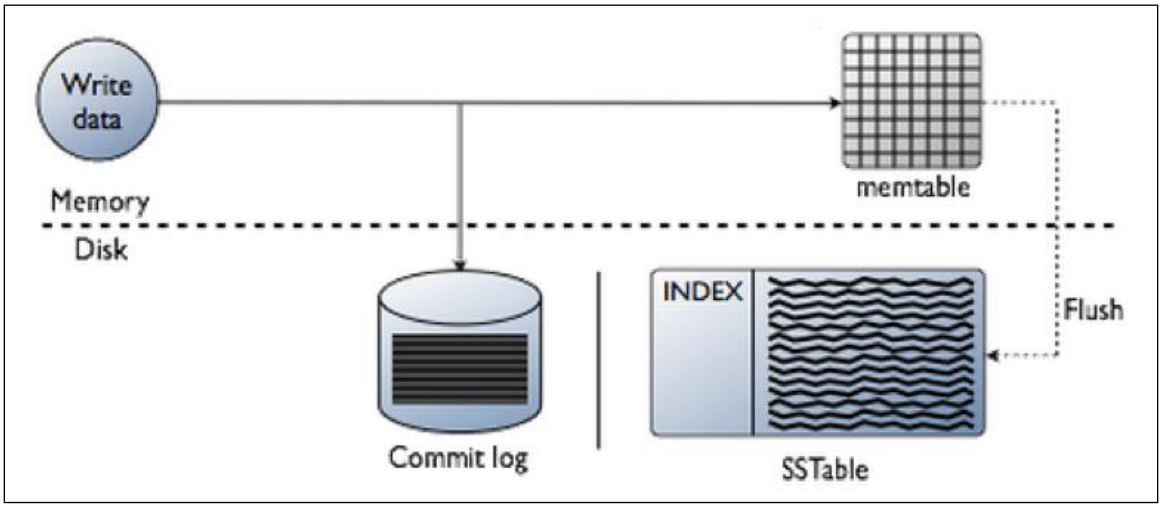

Figure 3-5: The Cassandra Write Path source ([23])

For a read request, Cassandra consults an in-memory data structure called a Bloom filter that checks the probability of an SSTable having the needed data. Figure 3-6 shows how a read request is handled in Cassandra.

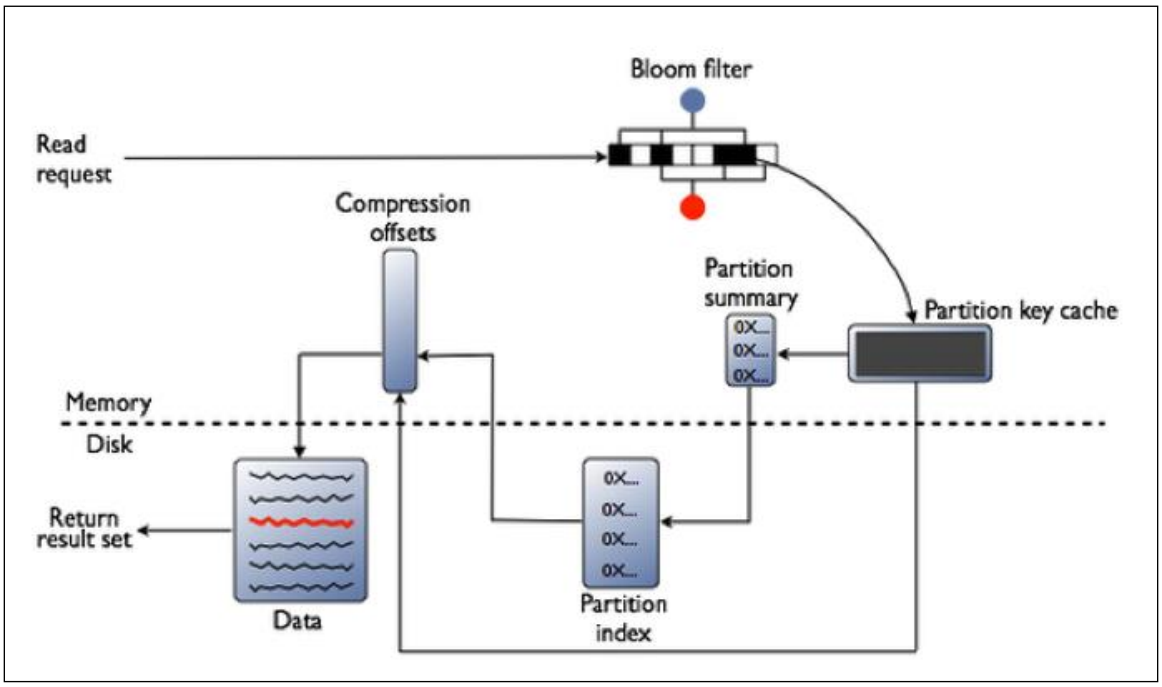

Figure 3-6: The Cassandra Read Path (source [23])

4. SYSTEMS, METRICS AND BENCHMARKS

In Chapter 3, we introduced various database systems that we will be investigating for energy efficiency and performance. In this chapter, we will study systems, metrics and benchmarks used in our experiments to measure the energy efficiency of databases.

4.1 Marcher System Configurations

All experiments presented in this thesis are executed on nodes of the Marcher system, which is provided as part of the NSF funded Marcher project. Marcher is a power-measurable heterogeneous cluster system containing general-purpose multicores, GPU K20 accelerators and Intel Xeon Phi (MIC) coprocessors, as well as DDR3 main memory and hybrid storage with hard drives and SSDs. Marcher is equipped with complementary, easy to deploy component-level power measurement tools for collecting accurate power consumption data of all major components (e.g CPU, DRAM, Disk, GPU, and Xeon Phi). To reduce the cost and time of designing and manufacturing external power sensors, we leveraged the built-in power sensors provided by some of the computing components. These power sensors are available for CPUs, GPUs and Xeon Phis, which can be accessed via the Intel RAPL interface, the NVIDIA Management Library (NVML) interface, and the Intel MICAccess API respectively.

All power results presented in this thesis are generated by using "Log_power_to_file" API which takes a script file containing query to be executed as a parameter.

"Log_power_to_file" uses the mentioned power sensors to measure power consumption of various components of a system. We are primarily interested in CPU and DRAM power. Although, the Marcher System can also provide disk power, we do not include it because database queries are not I/O intensive therefore disk power remains identical most of the time.

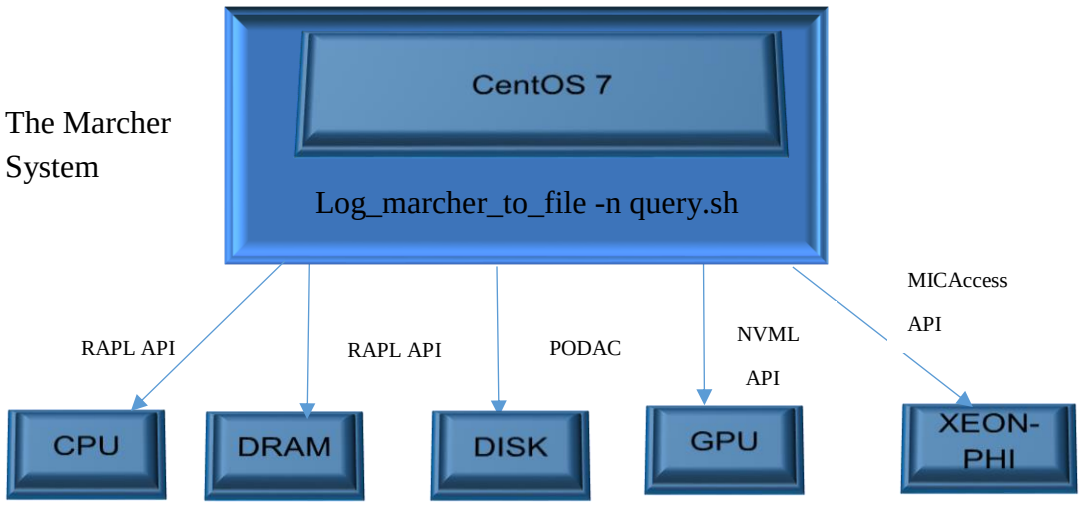

Figure 4-1: Architecture of the Marcher System

We used a cluster of two nodes for our experiments. System specification of each node has been provided in Table 4-1.

Table 4-1: System Specifications

OS	CentOS 7
Processor	Intel Xeon processor E5-2600 and E5-2600 v2 family
CPU Cores	16
Threads/Core	2
Chips Enabled	2
Cores Per Chip	8
Power Governor	Performance
CPU Memory Size	32 GB
File Sharing System	NFSv3
Average Idle Power	89.69 Watt

4.2 Power Measurement Metrics

In this chapter, we study the Greenup, Powerup and Speedup (GPS-UP in short) metrics, which allow software developers intuitively understand the correlations of performance, power, and energy for software optimizations [25]. The GPS-UP metrics can categorize almost all software optimizations. GPS-UP metrics are three numbers calculated for each software run to evaluate the relationship between energy, power and runtime.

4.2.1 Speedup, Greenup and Powerup Metrics

The Speedup concept covers any comparison between two implementations of the same query whether it is a parallel or serial code. Assume we have two implementations of an query. One of them is an un-optimized query and the other is an optimized query for better performance or energy consumption. Speedup of the optimized version is defined as

$$\text{Speedup} = \frac{T\phi}{To}, \quad (1)$$

where Tϕ is the total execution time of non-optimized query, and To is the total execution time of the optimized query. Similarly, Greenup is the ratio of the total energy consumption of the non-optimized query (Eϕ) over the total energy consumption of the optimized query (Eo). Greenup is analogous to Speedup as it reflects how green the optimized code is in term of energy consumption.

$$\text{Greenup} = \frac{E\phi}{Eo}, \quad (2)$$

Assuming, Pϕ is the average power consumed by the non-optimized query and Po is the average power consumed by the optimized query, we can define Eϕ and Eo as

$$E\phi = T\phi * P\phi \qquad Eo = To * Po \quad (3)$$

By substituting Eq.3 in Eq.2, we get

$$\text{Greenup} = \frac{T\phi P\phi}{Eo} = \frac{\text{Speedup} * P\phi}{Po} \quad (4)$$

Greenup and Speedup defines the measure of the energy and performance respectively. Eq.4 introduces a new ratio to define the average power consumption ratio, namely Powerup.

$$\text{Powerup} = \frac{P_0}{P_\phi} = \frac{\text{Speedup}}{\text{Greenup}} \quad (5)$$

Powerup implies the power effects of an optimization. A less than 1 Powerup implies power savings while a greater than 1 Powerup indicates that the optimized code consumes more power in average.

4.2.2 GPS-UP Software Categories

We can compare any two queries to find out which one is better in terms of performance and energy efficiency using Greenup, Powerup and Speedup metrics. This method provides a unique way to evaluate the impact of the optimization on performance, power and energy efficiency. We can categorize impact of optimized query based on powerup and speedup as follow:

1. Powerup < 1, and Speedup > 1 indicates optimizations run faster and consumes less power, leading to more energy savings as both time and power have decreased.
2. Powerup = 1, and Speedup > 1 indicates optimizations have better performance but on average consume the same amount of power. We usually get this category in serial optimizations. This category justifies why some developers only focus on performance and neglect energy efficiency. It is usually found in CPU intensive applications where

energy and time scale linearly.

3. Powerup > 1, Speedup > 1, and Speedup > Powerup indicates better performance at the expense of consuming more power. Since the Speedup obtained is more than the power penalty spent, the optimized code still saves energy.

4.3 YCSB Benchmark

In this section, we will discuss about the Yahoo! Cloud Server Benchmark (YCSB) used for our experiments to compare performance and energy efficiency of databases. YCSB is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs. It is often used to compare relative performance of database management systems. However, we will be studying not only the performance of the databases but also the energy efficiency of databases using our Marcher System.

4.3.1 Why YCSB

YCSB has long been the de facto open standard for comparative performance evaluation of data stores. Many factors go into deciding which data stores to use including basic features, data model, and performance characteristics on a given type of workload. It's critical to compare multiple data stores intelligently and objectively so that sound architectural decisions can be made.

From the perspective of a generic, database-neutral, performance evaluation utility, YCSB is currently the de-facto comparative benchmark for SQL and NoSQL data stores.

It includes support for a wide range of database bindings and is commonly used to compare their performance for a set of desired workloads.

4.3.2 YCSB Setup

In this subsection, we will discuss in detail about how to set up YCSB.

YCSB consists of two parts:

1. The YCSB Client, an extensible workload generator.
2. The core workloads, a set of workload scenarios to be executed by the generator.

YCSB includes a set of core workloads that define a basic benchmark for cloud systems. The core workloads consist of six different workloads as shown in Table 4-2.

Table 4-2: YCSB Workloads

YCSB Workloads		
Workload	**Type**	**Operation**
A	Update Heavy	Read: 50%, Update: 50%
B	Read Heavy	Read: 95%, Update: 5%
C	Read Only	Read: 100%
D	Read Latest	Read: 95%, Insert: 5%
E	Short Ranges	Scan: 95%, Insert: 5%

All six workloads have a data set which is similar. Workloads D and E insert records during the test run. Thus, to keep the database size consistent, we ran the workloads in following sequence:

1. Load the database, using workload A's parameter file (workloads/workloada) and the "-load" switch to the client.
2. Run workload A (using workloads/workloada and "-t") for a variety of throughputs.
3. Run workload B (using workloads/workloadb and "-t") for a variety of throughputs.
4. Run workload C (using workloads/workloadc and "-t") for a variety of throughputs.
5. Run workload F (using workloads/workloadf and "-t") for a variety of throughputs.
6. Run workload D (using workloads/workloadd and "-t") for a variety of throughputs. This workload inserts records, increasing the size of the database.
7. Delete the data in the database.
8. Reload the database, using workload E's parameter file (workloads/workloade) and the "-load switch to the client.
9. Run workload E (using workloads/workloade and "-t") for a variety of throughputs. This workload inserts records, thus increasing the size of the database.

We ran the above workloads for MySQL, MongoDB and Cassandra in order to study performance and power usage of the various databases under similar workload conditions.

4.4 Database Analysis using Twitter Data

YCSB provides an effective and efficient platform to compare various databases with ease. However, datasets used by the benchmark were fairly simple with a single table or one field document which we feel is almost non-existent use case in real world scenarios. Also, YCSB is limited in terms of executing complex queries.

We wanted to analyze databases with more complicated data and index landscape with much more extensive querying. For achieving that we required a highly available large set of data which we could easily extract and input in our databases for our experiments. We used Twitter data for our experiments.

Twitter is a massive social networking site tuned towards fast communication. Users on Twitter generate over 400 million Tweets every day. Twitter's popularity as an information source has led to the development of applications and research in various domains. Researchers have used Twitter to predict the occurrence of earthquakes and identify relevant users to follow to obtain disaster related information. Some of these Tweets are available to researchers and practitioners through public APIs at no cost. A sampled view of Twitter can be easily obtained through the APIs.

For our analysis, we used Streaming APIs to collect Twitter data. Streaming APIs provides a continuous stream of public information from Twitter. These APIs use the push strategy for data retrieval. Once a request for information is made, the Streaming APIs provide a continuous stream of updates with no further input from the user. These

streams can be extracted in JSON which is a lightweight data interchange format which can later be consumed by databases using relevant transformations.

4.5 Dynamic Voltage and Frequency Scaling (DVFS)

Dynamic Voltage and Frequency Scaling (DVFS) is an advanced power-saving technology which aims to lower a component's power state while still meeting the performance requirement of the running workload. Some of the governors supported by Linux kernel are as follows:

1. Performance: This CPUfreq governor sets the CPU statically to the highest frequency within the borders of scaling_min_freq and scaling_max_freq.

2. Powersave: This governor sets the CPU statically to the lowest frequency within the borders of scaling_min_freq and scaling_max_freq.

3. Ondemand: Ondemand governor sets the CPU depending on the current usage. To do this the CPU must have the capability to switch the frequency very quickly. Some papers studied the effect of using DVFS to save energy [36]. To determine the effect of DVFS we conducted our experiments on various databases by executing queries using "performance" and "ondemand" governors.

5. PERFORMANCE AND ENERGY ANALYSIS OF MYSQL

In this chapter, we focus on analyzing relational database in depth using the Twitter data. SQL statements are used to retrieve data from the MYSQL relational database. We can get the same results by writing different SQL queries. But use of the best query is important when performance and energy efficiency is considered. Relational databases have been studied for decades to determine the tradeoffs between energy and performance [22,24,30,31,32]. Different methods for query processing and optimization are used as per the data size and the complexity of queries.

Database performance is one of the most challenging aspects of an organization's database operations. A well-designed application may still experience performance problems if the SQL query it uses is poorly constructed. It is much harder to write efficient SQL queries than to write functionally correct SQL queries. As such, SQL query optimization can help significantly improve a system's performance and energy efficiency. The key to tuning SQL queries is to minimize the search path that the database traverses to find the data.

As the amount of data increases, the performance decreases and the execution time and energy consumption increases. Therefore, optimization of these queries becomes

essential since the speed of user response and running performance of database system determine the vitality of information system.

5.1 Twitter data analysis using MySQL

To analyze MySQL database for performance and power efficiency using complicated data and extensive querying, we streamed about 100 GB of Twitter data using Streaming APIs. Thereafter, we designed a normalized database and imported Twitter data into the tables. Various optimization techniques studied to improve performance and energy efficiency are described in the following section.

5.2 Query Optimization Techniques

SQL tuning is a phenomenally complex subject. Many books have been written about the nuances of Oracle SQL tuning; however, there are some general guidelines that every database developer follows to improve the performance of their systems. The goals of SQL tuning focus on improving the execution optimization of database system, which plays an important role and runs through the entire life cycle of database applications. There are various optimization techniques, which can be implemented to make the SQL queries run faster and consume less energy. The goal of optimizing any SQL statement includes delivering quick response times using less CPU resources, and reducing I/O operations. The following content provides best practices for optimizing the performance of SQL queries.

5.2.1 Indexing

Unnecessary full-table scans cause a huge amount of unnecessary I/O and can drag-down an entire database. The tuning expert first evaluates the SQL based on the number of rows returned by the query. The most common tuning remedy for unnecessary full-table scans is adding indexes [26]. The Primary Key for a table acts as a default index. Additional indexes can be added to a table depending upon the data size it holds. Other type of indexes like Standard b-tree indexes, bitmapped and function-based indexes can also eliminate full-table scans. For analyzing the above technique, we created normalized tables and created index on the fields being used for data manipulation. For example, to get the count of users for a location, we ran the following query using indexes and without using indexes.

select count(*) as tweets from location_details where location='San Diego';

The query results are presented in Table 5-1 and Figure 5-1.

Table 5-1: Non-indexed vs Indexed Search Query in MySQL

Non-Indexed Vs Indexed Search						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Non-indexed	142.8777	3.721247	531.6832	66.3442	0.9492	69.8942
Indexed	135.62	0.05609	7.606926			

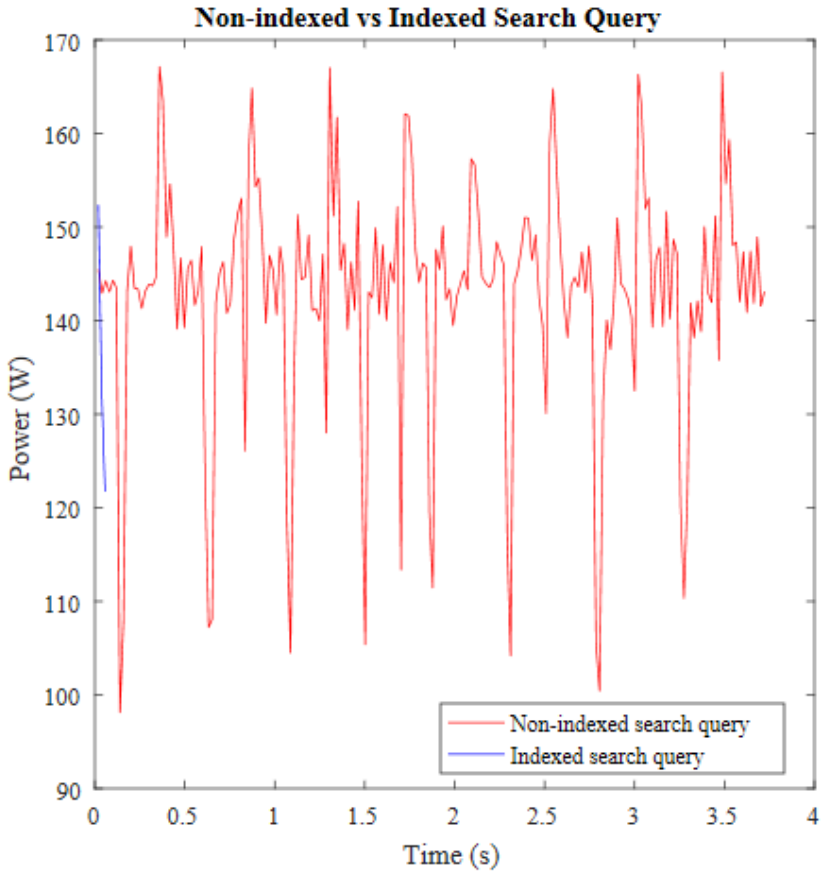

Figure 5-1: Non-indexed vs Indexed Search Query in MySQL

As depicted in Figure 5-1, indexing helped in gaining a speedup of about 66X and greenup of almost 70X. Using this optimization technique, we not only gained high performance but also saved energy.

We extended the similar experiment for delete operation to study the effect of indexing on other query operations. Results for the same are displayed in Table 5-2 and Figure 5-2.

Table 5-2: Non-indexed vs Indexed Delete Query in MySQL

Non-Indexed Vs Indexed Delete Query						
Query	Power (W)	Time (s)	Energy (J)	Speedup	Powerup	Greenup
Non-indexed	67.887	3.120975	211.8736	13.4708	0.7789	17.2927
Indexed	52.883	0.231684	12.25212			

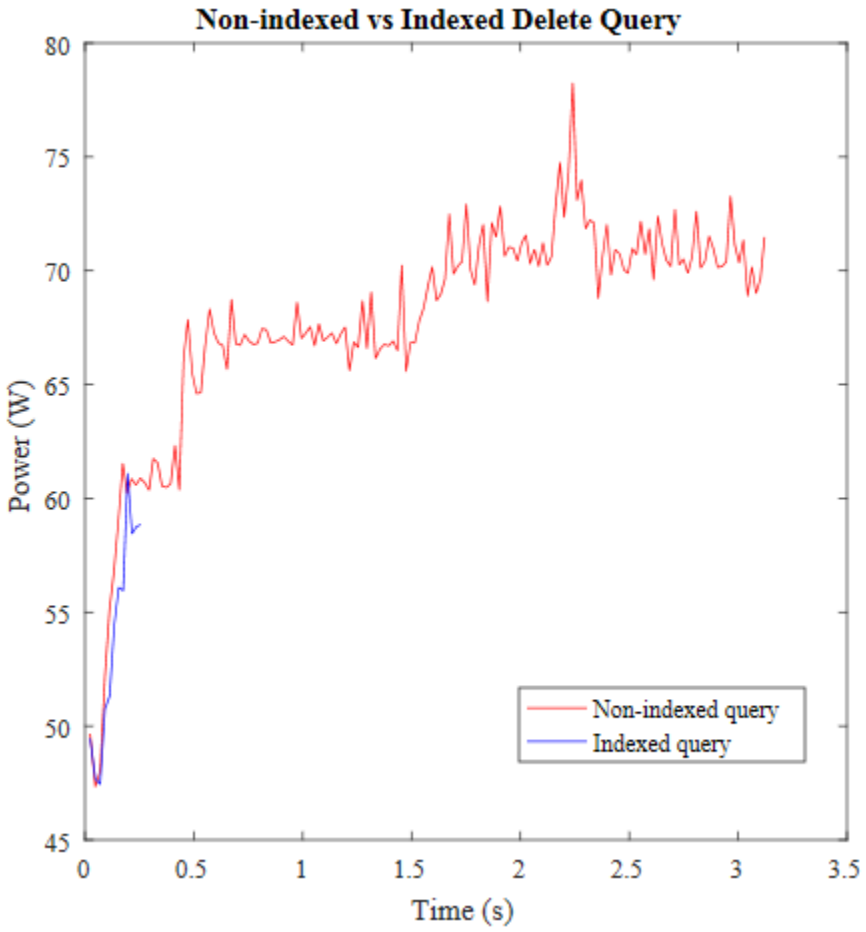

Figure 5-2: Non-indexed vs Indexed Delete Query in MySQL

Here, indexing helped in gaining performance by 13X and reducing energy consumption by 17X. Since each index keeps the indexed fields stored separately, it makes finding the

right entries particularly easy. The database finds the entries in the index then cross-references them to the entries in the tables. This cross-referencing takes time but is faster than scanning the entire table. This contributes to lower execution times and reduces power consumption.

There was another scenario where we found indexing helpful in attaining high performance and high energy efficiency. The following query was used to find the most tweeted user.

SELECT username, count(*) AS count FROM tweet_details GROUP BY username ORDER BY count DESC LIMIT 1;

The results of the above query is shown in Table 5-3.

Table 5-3: Non-indexed vs Indexed Query to find most tweeted user in MySQL

Non-Indexed Vs Indexed Query						
Query	Power W)	Time (s)	Energy (J)	Speedup	Powerup	Greenup
Non-indexed	75.8749	135.0396	10246.12	33.0759	0.8218	40.2436
Indexed	62.361	4.082716	254.6023			

As per Table 5-3 and Figure 4-3, we observed that indexed query has speedup 33 times more than that of non-indexed query and greenup of about 40X. Also, greenup is more than speedup in this query. This is mainly because in this case, indexed query not only runs faster but also consumes less power, leading to more energy savings as both time and power have decreased. This kind of performance boost typically occurs for queries

which rely more on the cache rather than CPU.

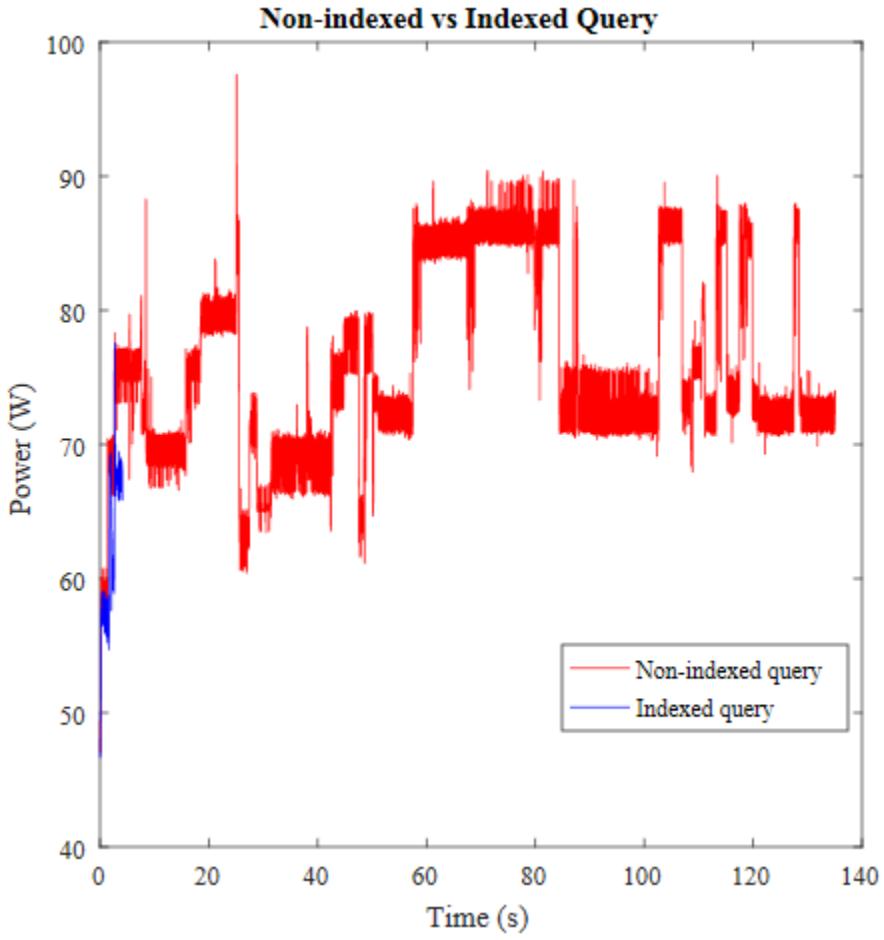

Figure 5-3: Non-indexed vs Indexed Query to find most tweeted user in MySQL

5.2.2 Avoid using Select * Clauses

The dynamic SQL column reference (*) gives you a way to refer to all of the columns of a table. Do not use the * feature because it is very inefficient -- the * has to be converted to each column in turn. The SQL parser handles all the field references by obtaining the names of valid columns from the data dictionary and substitutes them on the command line, which is time consuming. To verify the assumption, we ran the following queries.

SELECT * FROM location_details l,user_details u WHERE u.username=l.username AND l.location='Houston';

Query without using '*':

SELECT u.screen_name,l. tweet_id FROM location_details l, user_details u WHERE u.username=l.username AND l.location='Houston';

Here, as indicated from Table 5-4 and Figure 5-4, query "without using select *" has a speedup 1.0683 times more than query with "select *" clause and greenup of 1.2185 times. It means that the query "without using select *" not only runs faster but also consumes less power, leading to more energy savings as both time and power have decreased.

Table 5-4: Query execution using select * clauses in MySQL

Select * Clauses						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
With Select *	66.6615	3.84684	256.4361	1.0683	0.8767	1.2185
Without Select *	58.444	3.600677	210.438			

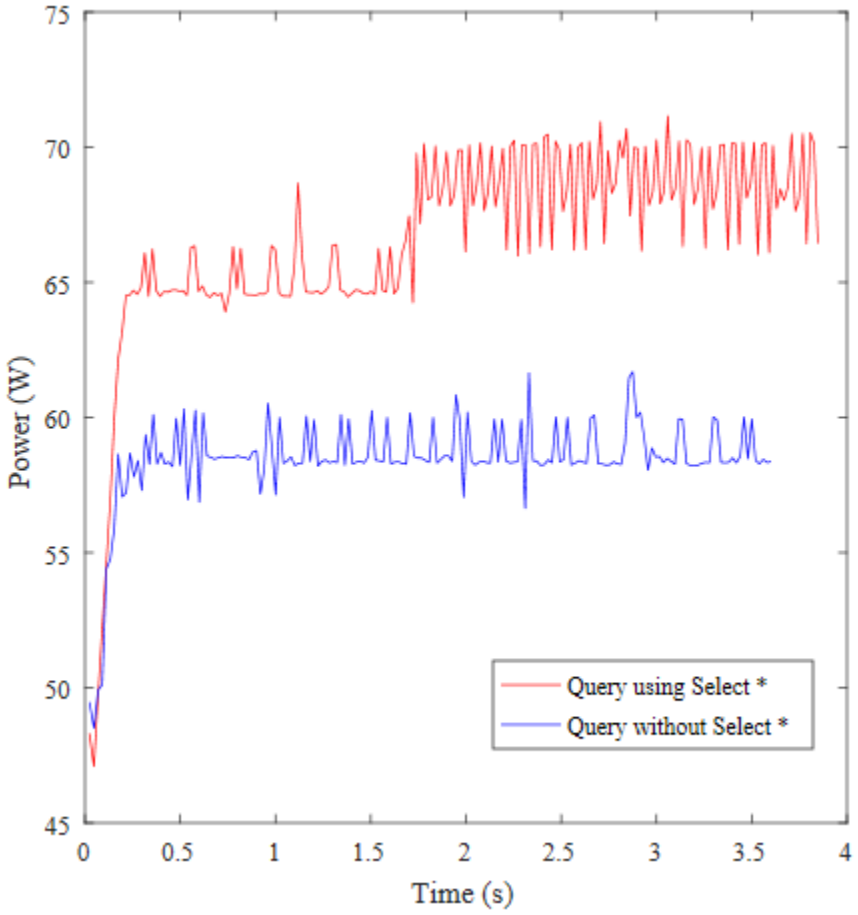

Figure 5-4: Query execution using select * clauses

5.2.3 IN vs EXISTS

The EXISTS function searches for the presence of a single row that meets the stated criteria, as opposed to the IN statement that looks for all occurrences. There was a wide gap in performance of the queries using the above-mentioned clauses. Queries for the same are presented in the following page.

Query using 'IN':

SELECT l.tweet_id FROM location_details l WHERE l.username IN(SELECT u.username FROM user_details u) AND l.location='Houston';

Query using 'EXISTS':

SELECT l.tweet_id FROM location_details l WHERE EXISTS(SELECT '1' FROM user_details u WHERE u.username=l.username) AND l.location='Houston';

For the mentioned technique, we observed that query using EXISTS clause is a clear winner(as indicated in Table 5-5 and Figure 5-5). It provides a speedup of 146 times than query using IN clause and greenup of 208 times. We note that greenup is a lot more than speedup of query using EXISTS clause. This particularly happens because of the way EXISTS clause works. In case of EXISTS clause, EXISTS do a partial scan of the table as it can stop after it finds the very first matching row. However, IN clause scans every row in the entire table to determine if they match the criteria. The ability to stop working after finding the first row that meets the criteria of the WHERE clause is what makes EXISTS so efficient.

Table 5-5: IN vs EXISTS

IN vs EXISTS						
Query	Power W)	Time (s)	Energy (J)	Speedup	Powerup	Greenup
Using 'IN'	68.9364	13.59254	937.0208	146.3404	0.7010	208.7463
Using 'EXISTS'	48.3275	0.092883	4.4888			

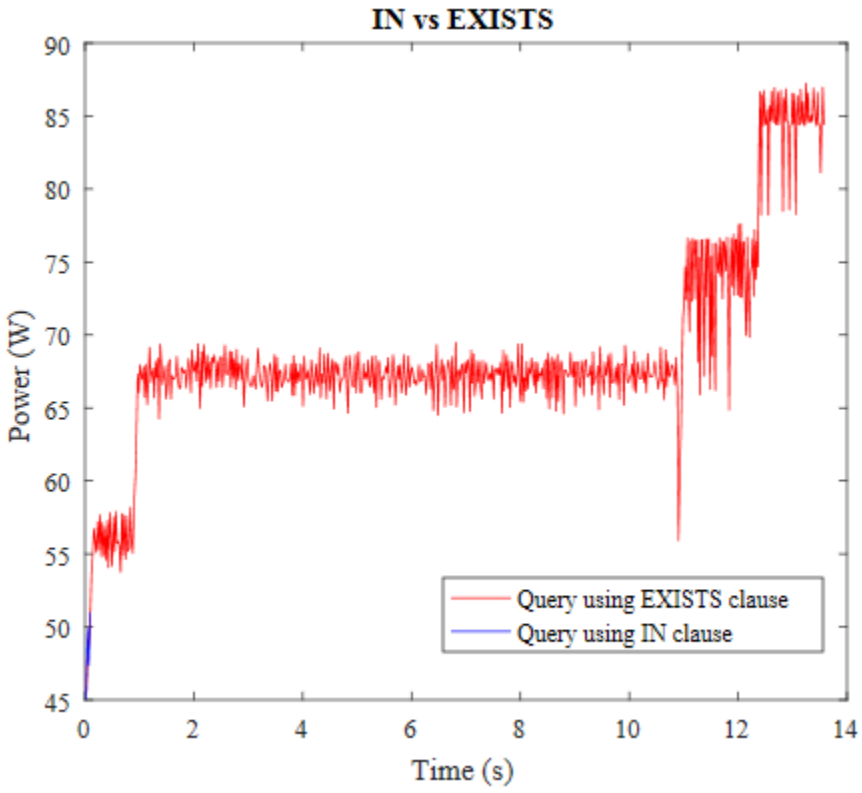

Figure 5-5: IN vs EXISTS

5.3 Impact Analysis of DVFS on optimized queries

To study the impact of DVFS on query execution, we ran several queries by setting CPU frequency to "performance" and "ondemand" governors. As explained in Chapter 4, "performance" and "ondemand" are two CPU frequency governors supported by linux keernels. In case of "performance" governor, CPU is set to the highest frequency whereas in case of "ondemand" governor CPU frequency varies depending on the current usage. To study the impact of DVFS on optimized queries, we conducted following experiments. Figure 5-6 shows the behavior of MySQL insert query.

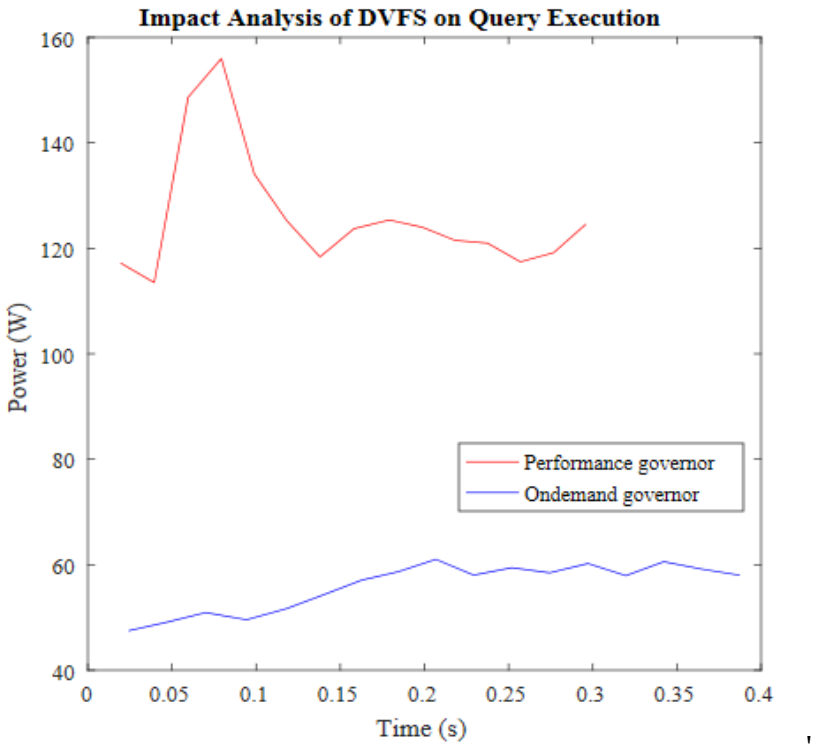

Figure 5-6: Impact analysis of DVFS on MySQL insert query

As observed in Figure 5-6, power consumption for "performance" governor is almost twice as compared to "ondemand" governor. Even though query execution takes a little longer when run using "ondemand" governor, nonetheless it is more energy efficient. We can conclude that lower execution time does not always mean that it will be more energy efficient. Power utilization and execution time are equally important in determining energy efficiency.

Another query where we compared "performance" and "ondemand" governor was MySQL update query. The trace for the same is shown in Figure 5-7.

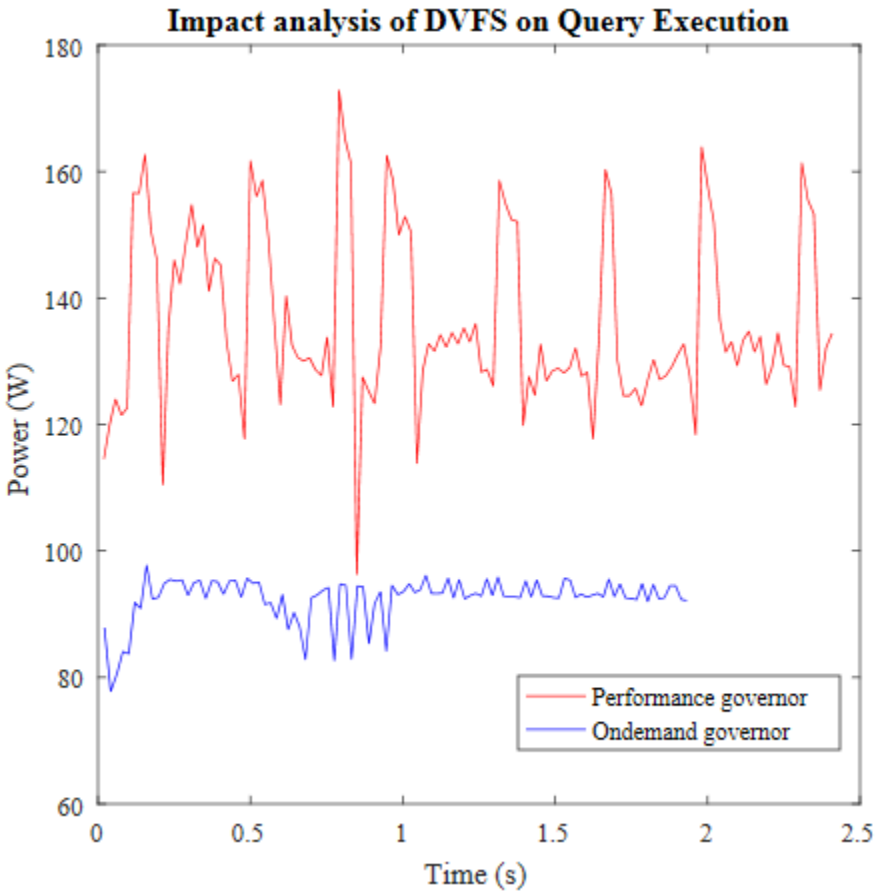

Figure 5-7: Impact analysis of DVFS on MySQL update query

We observed that in case of "performance" governor, both power consumption and query execution time were high as compared to "ondemand" governor. This can be explained as sometimes higher power and processing speeds can result in slowdown [35]. The slowdowns occur at higher frequencies when the early arrival of a single thread causes the atomic journal commit to lock with less batched threads than in the lower frequency case. In the lower frequency case, the difference between the lead thread and other threads is much smaller, therefore less time is spent in waiting. Slower processor

frequencies effectively increase the number of threads that access the shared resource while reduce the overall commits required at higher processor frequencies.

5.4 Conclusion

This chapter summarized various optimization techniques which not only helped in improving performance but also energy efficiency. We unfolded few interesting findings that contribute towards enhancing energy efficiency. Firstly, indexed query not only runs faster but also consumes less power, leading to more energy savings as both time and power have decreased. We gained a speedup of almost 30X in some cases using this technique. Secondly, we observed that how a query works internally is also a contributing factor in saving power. In case of IN vs EXISTS clause, the internal functioning of EXISTS clauses helped in attaining a speedup of 146 times than IN clause. Lastly, we studied the impact of DVFS on query execution. We found that lower execution time does not always lead to high energy efficiency. Power utilization and execution time are equally important in determining energy efficiency. Also, higher power and processing speeds can result in slowdown sometimes due to longer waiting times in synchronizing events.

6. PERFORMANCE AND ENERGY ANALYSIS OF MONGODB

In this chapter, we focus on analyzing MongoDB which is a NOSQL/non-relational database in depth using Twitter data. There are many factors that can affect database performance and responsiveness including index use, query structure, data models and application design, as well as operational factors such as architecture and system configuration. MongoDB provides the following capabilities which makes it a highly efficient database for large data stores.

1. Document-Oriented Storage - MongoDB stores its data in JSON-style objects. This makes it very easy to store raw documents from Twitter's APIs.
2. Index Support - MongoDB allows for indexes on any field, which makes it easy to create indexes optimized for your application.
3. Straightforward Queries - MongoDB's queries, while syntactically much different from SQL, are semantically very similar. In addition, MongoDB supports MapReduce, which allows for easy lookups in the data.

6.1 Twitter Data Analysis using MongoDB

To analyze MongoDB using Twitter data, we need to create a collection in MongoDB to store Twitter streams. Since, MongoDB uses JSON to store its documents, we can import the data from Twitter API using the following command:

mongoimport --db Twitter_db --collection "Twitter_data" --type json --file filename.json

Here, "mongoimport" is a utility that is packaged with MongoDB that allows to import JSON documents. “Twitter_db” refers to the database with “Twitter_data” as collection. To get the power reading using the Marcher system, we used PyMongo [38] API to run queries. PyMongo is a Python distribution containing tools for working with MongoDB.

6.2 Query Optimization Techniques

To make our documents quickly accessible and perform various operations on the large amount of data stored in our collection, it is important to optimize queries. There are many factors that can affect database performance and energy efficiency including index use, query structure, data models and application design, as well as operational factors such as architecture and system configuration. The following sub-sections describe techniques for optimizing application performance as well as energy efficiency of MongoDB.

6.2.1 Covered Queries

Indexes improve the efficiency of read operations by reducing the amount of data that query operations need to process. This simplifies the work associated with fulfilling queries within MongoDB. A covered query is a query that can be satisfied entirely using an index and does not have to examine any documents. An index covers a query with the

following conditions:

- all the fields in the query are part of an index, and
- all the fields returned in the results are in the same index.

For analyzing the above technique, we created index on “user.location” field using the following query:

db.Twitter_data.ensureIndex({'user.location':1})

Then, we scanned all the documents using the covered query. The results have been provided in Table 6-1 and Figure 6-1.

Table 6-1: Unoptimized vs covered query in MongoDB

Un-optimized vs Covered queries						
Query	Power(W)	Time(s)	Energy (J)	Speedup	Powerup	Greenup
Un-optimized	126.6609	162.7915	20619.32	276.1452	0.5541	498.3509
Covered	70.1851	0.589514	41.3751			

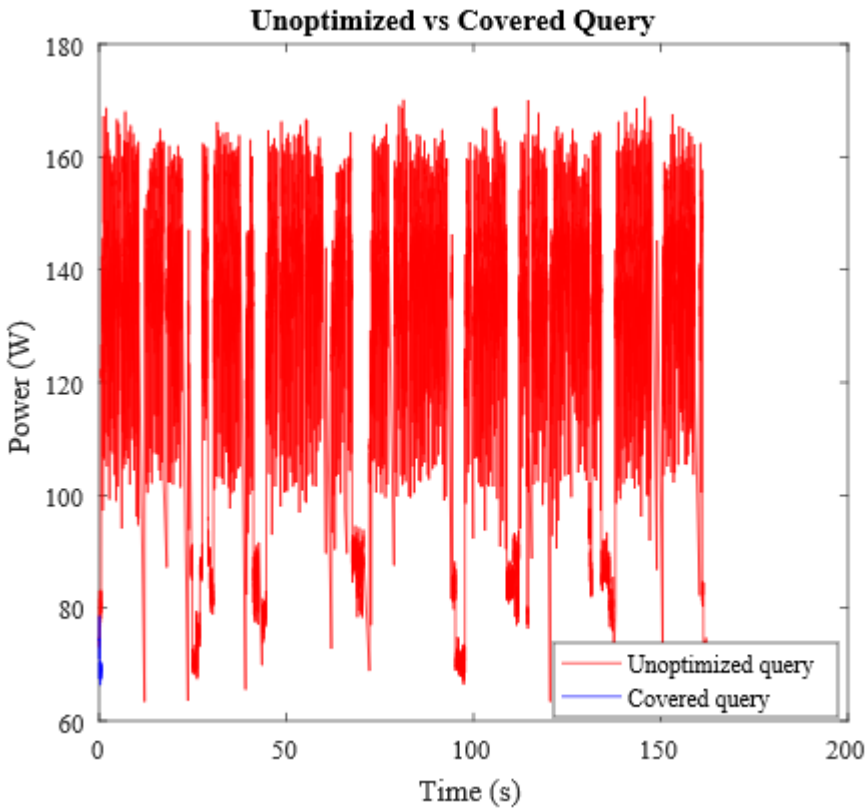

Figure 6-1: Unoptimized vs covered query in MongoDB

As indicated by the power trace and Table 6-1, covered query has a speedup of 276 times more than un-optimized query. Greenup is nearly 478 times more in case of covered. We also observe that greenup is much higher as compared to speedup. This occurs if the input size is small enough to fit into the cache. MongoDB keeps the most recently used data in DRAM. Therefore, if we have created indexes for the query and working data set fits in DRAM, MongoDB serves all queries from memory. Hence, less main memory is utilized and more power is saved.

6.2.2 Non-indexed vs Indexed queries

Once documents are inserted into a collection, querying them will be slow if MongoDB does not know which fields in the document are to be optimized for faster lookup. One of the most important concepts to understand fast access of a MongoDB collection is indexing. The indexes we choose will depend on data to be queried. We ran the same experiment using the following query for indexed and non-indexed field "user_mentions".

db.Twitter_data.remove({"entities.user_mentions.id" : "574834900"})

As evidenced from the data provided in the Table 6-2 and Figure 6-2, decrease in the run time improved performance by gaining a speedup of 283 times. We also observed a high greenup of 44 times more than un-optimized query. This is also contributed by the way caches are handled in MongoDB. Since MongoDB keeps the most recently used data in DRAM, therefore, if we have created indexes for the query and working data set fits in DRAM, MongoDB serves all queries from memory. Hence, less main memory is utilized and more power is saved.

Table 6-2: Non-indexed vs Indexed Delete query in MongoDB

Non-indexed vs Indexed Delete query						
Query	Power (W)	Time(s)	Energy(W)	Speedup	Powerup	Greenup
Non - indexed	130.5215	165.2470	21568.29	283.2482	0.5975	474.0064
Indexed	77.9913	0.5834	45.5021			

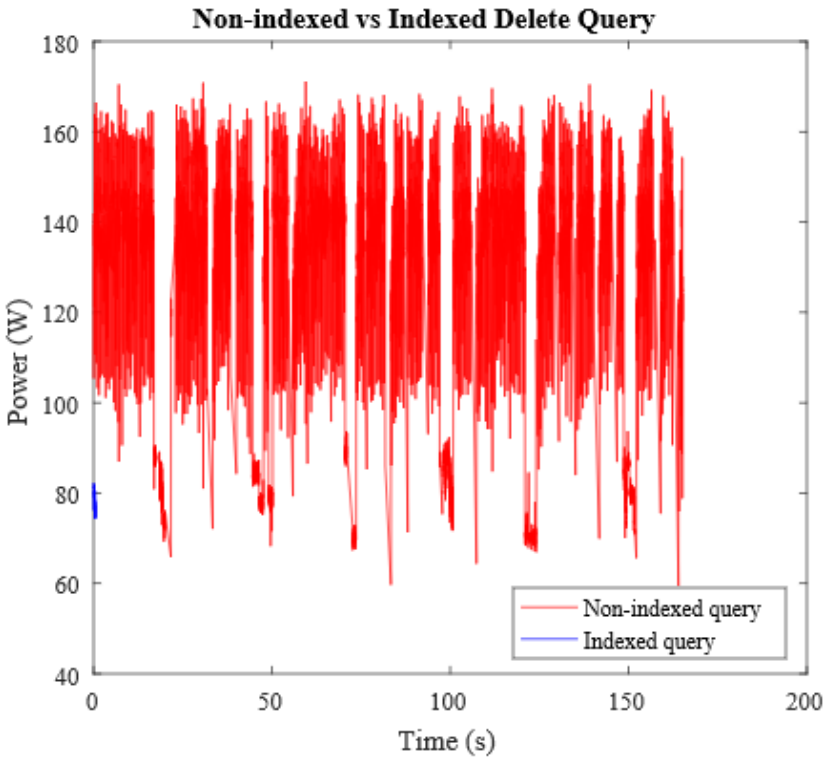

Figure 6-2: Non-indexed vs Indexed Delete query in MongoDB

We conducted another experiment to demonstrate the impact of indexing using insert query. The results for the same are displayed in Table 6-3 and Figure 6-3.

Table 6-3: Non-indexed vs Indexed Insert query in MongoDB

Non-indexed vs Indexed Delete query						
Query	Power (W)	Time(s)	Energy(W)	Speedup	Powerup	Greenup
Non - indexed	136.8845	0.64429	88.19331	1.0403	0.5252	1.9807
Indexed	71.8965	0.6193	44.5255			

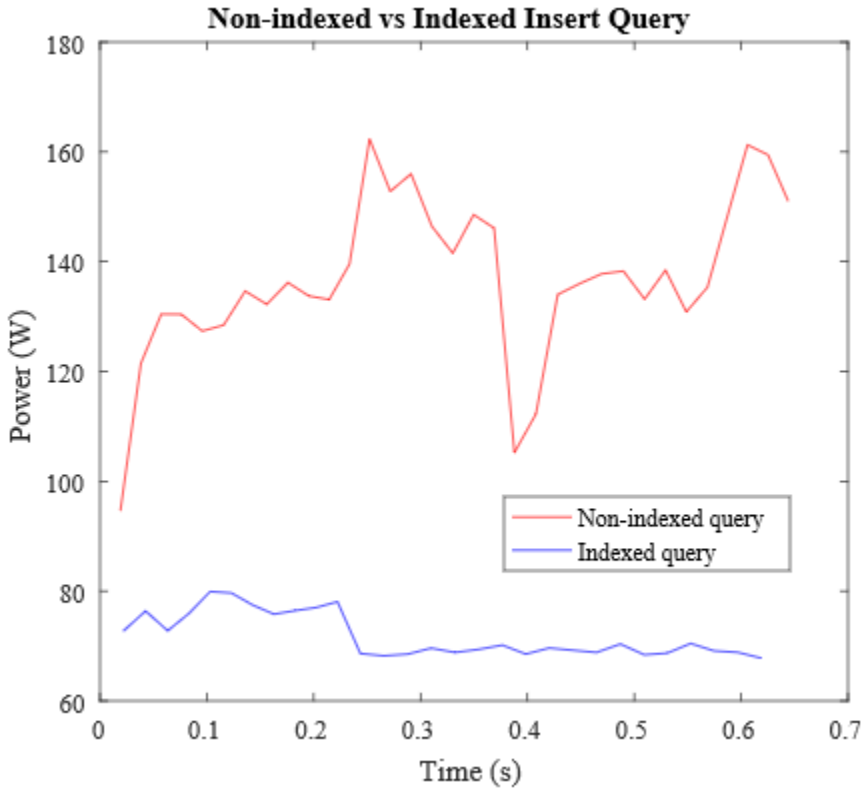

Figure 6-3: Non-indexed vs Indexed Insert query in MongoDB

As observed from Table 6-3 and Figure 6-3, indexing not only reduces the execution time of the query but also significantly reduces the power consumption. Indexed insert query has a greenup of almost 1.9 times more than non-indexed insert query.

6.2.3 Ordered vs Unordered queries

The next parameter that we examined was ordered and unordered bulk write operations [27,28]. Bulk write operations create a list of write operations to perform in bulk, which can be either ordered or unordered. Bulk operations builder used to construct a list of write operations to perform in bulk for a single collection. To instantiate the builder, use

either ***db.collection.initializeOrderedBulkOp()*** or

db.collection.initializeUnorderedBulkOp() method.

As shown in Table 6-4 and Figure 6-4, we observe that unordered updates not only take longer to execute but also consumes more power. The key to increasing speed on updates is to note how MongoDB gives a lot of control over how database operations are acknowledged by a server. This ranges from checking through acknowledgment that the operation has been acted on and up to confirming the operation has been written to the journal. This reflects how concerned the client is with the progress of the write –the more concern, the longer it will take for the various write operations to complete or fail. This is particularly low in case of ordered updates, leading to high energy efficiency.

Table 6-4: Ordered and Unordered Query in MongoDB

Ordered and Unordered Queries						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Unordered	80.792	1.081976	87.415	1.8665	0.9753	1.9137
Ordered	78.798	0.579669	45.67676			

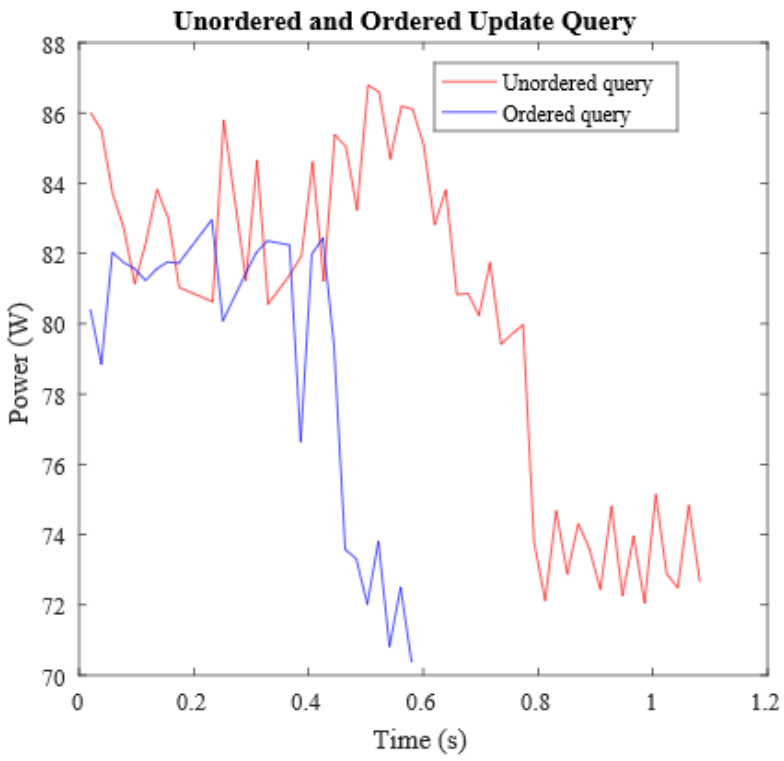

Figure 6-4: Ordered and Unordered Query in MongoDB

6.2.4 Projection Optimization using aggregation

MongoDB provides a rich set of aggregation operations that examine and perform calculations on the data sets. Running data aggregation on the MongoDB instance simplifies application code and limits resource requirements. Like queries, aggregation operations in MongoDB use collections of documents as an input and return results in the form of one or more documents. The pipeline provides efficient data aggregation using native operations within MongoDB, and is the preferred method for data aggregation. MongoDB also provides map-reduce operations to perform aggregation. In general, map-reduce operations have two phases: a map stage that processes each document

and emits one or more objects for each input document, and a reduce phase that combines the output of the map operation. Optionally, map-reduce can have a finalize stage to make final modifications to the result. Like other aggregation operations, map-reduce can specify a query condition to select the input documents as well as sort and limit the results. However, in general, map-reduce is less efficient and more complex than the aggregation pipeline. We used aggregation pipeline to find the most tweeted user from the MongoDB collection using following query.

```
db.Twitter_data.aggregate([{"$project": {
    "_id": 0, "entities.user_mentions" :1}}, {"$unwind": "$entities.user_mentions"},
        {"$group": {"_id": "$entities.user_mentions.screen_name",
            "count": {"$sum": 1}}}])
```

The results of the above experiment are as shown in Table 6-5 and Figure 6-5.

Table 6-5: Un-optimized vs Aggregated Query in MongoDB

Un-optimized vs Aggregated Query						
Query	Power(W)	Time(s)	Energy(W)	Speedup	Powerup	Greenup
Un-optimized	124.3553	287.6416	35769.76	2.4548	0.5749	4.2698
Optimized	71.49538	117.1723	8377.278			

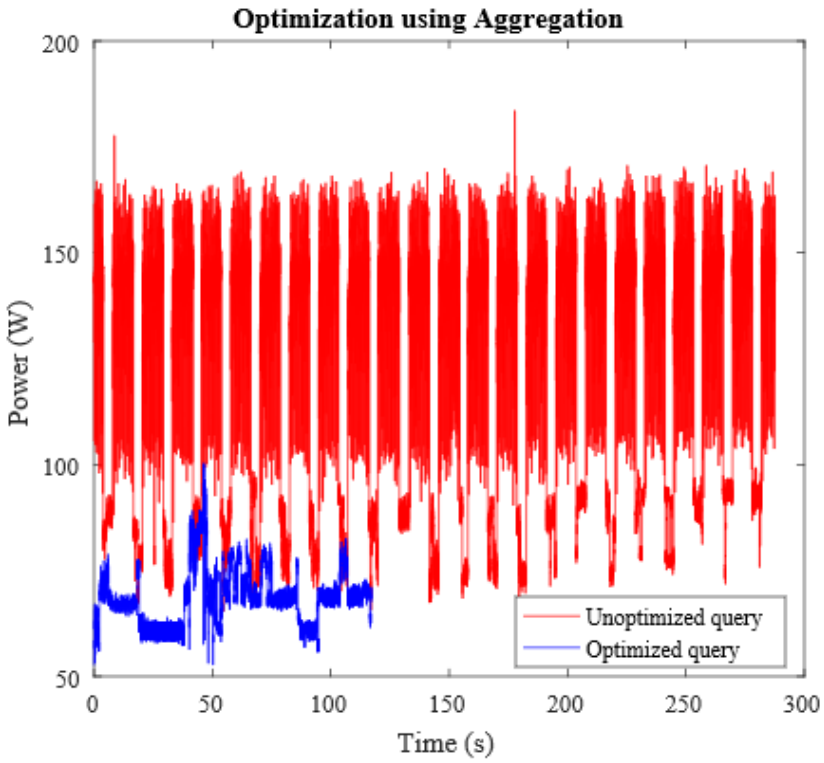

Figure 6-5: Un-optimized vs Aggregated Query in MongoDB

Aggregation proved to be quite energy efficient for complex queries. We gained a speedup of 2.4 times and greenup of 4.2 times for aggregated queries. As depicted in Figure 6.5, unoptimized query takes longer to execute and consumes more power compared to optimized query using aggregation. The aggregation pipeline can determine if it requires only a subset of the fields in the documents to obtain the results. If so, the pipeline will only use those required fields, reducing the amount of data passing through the pipeline. Since, we have limited data in the pipeline, it fits into the dataset of MongoDB leading to lesser main memory references. As a result,we save power and contribute to high energy efficiency.

6.2.5 Sharding

The next technique we used was sharding servers. Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations. MongoDB supports horizontal scaling through sharding. Horizontal scaling involves dividing entire dataset and load them over multiple servers. Additional servers can be added to increase capacity if required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Since expanding the capacity of the database only requires adding additional servers as needed (scale out). It can lower the overall cost than scale up to a high-end server. The trade-off is increased complexity in infrastructure and maintenance for the deployment. We used cluster with two nodes to measure performance and energy efficiency. The results are presented in Table 6-6.

Table 6-6: Single vs distributed shared server in MongoDB

Single vs distributed sharded server						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Single Server	77.5889	281.4785	21839.62	3.0828	1.8051	1.7077
Sharded Server	140.060	91.3054	12788.32			

For the sharded server, we observed that even though the execution time was reduced to almost half after distributing data on two nodes, power consumption was high. Multi-node servers help in gaining performance by distributing work over multiple nodes, however there is an increase of power consumption because of the idle power of nodes that are not contributing any work to the program execution.

6.3 Impact Analysis of DVFS on optimized queries

To study the impact of DVFS, we carried out several experiments to compare the performance and power usage for CPU frequency governor as ondemand and performance. Some of the experiments have been presented below to show the impact of DVFS.

In one of the experiments, we studied the impact of governors on execution of the query to find the most tweeted user using aggregation as shown in Figure 6-6.

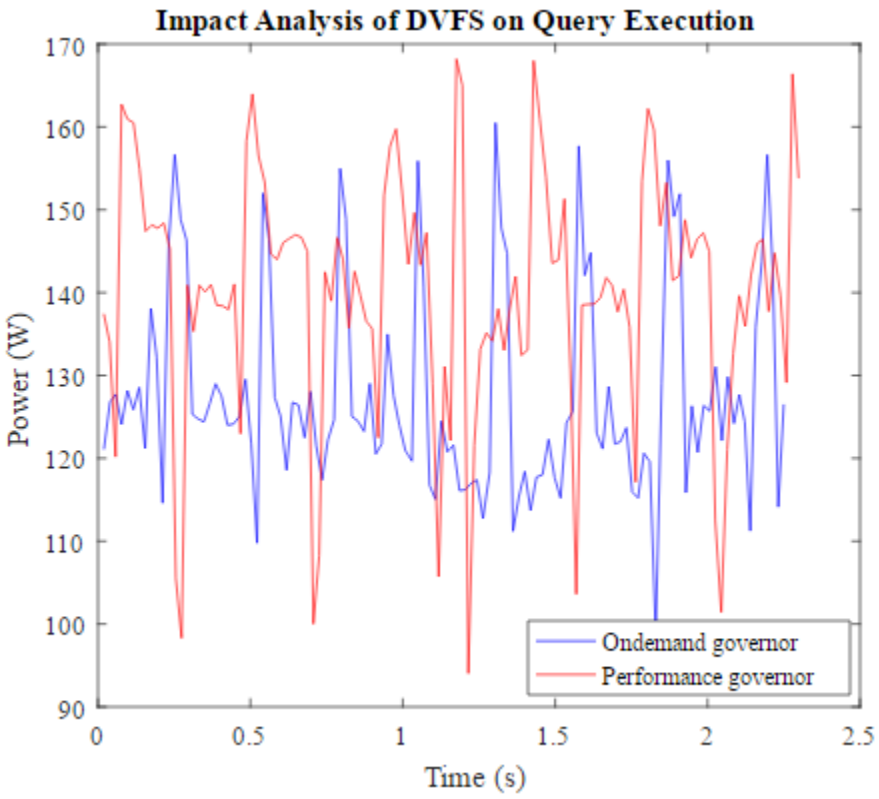

Figure 6-6: Impact of DVFS on query execution in MongoDB

We observed lot of parallelism in case of MongoDB owing to its distributed nature. However, in case of "ondemand governor", power consumption was less as compared to "performance" governor.

We conducted another experiment to analyze the effect of DVFS on search query. Same has been depicted in Figure 6-7.

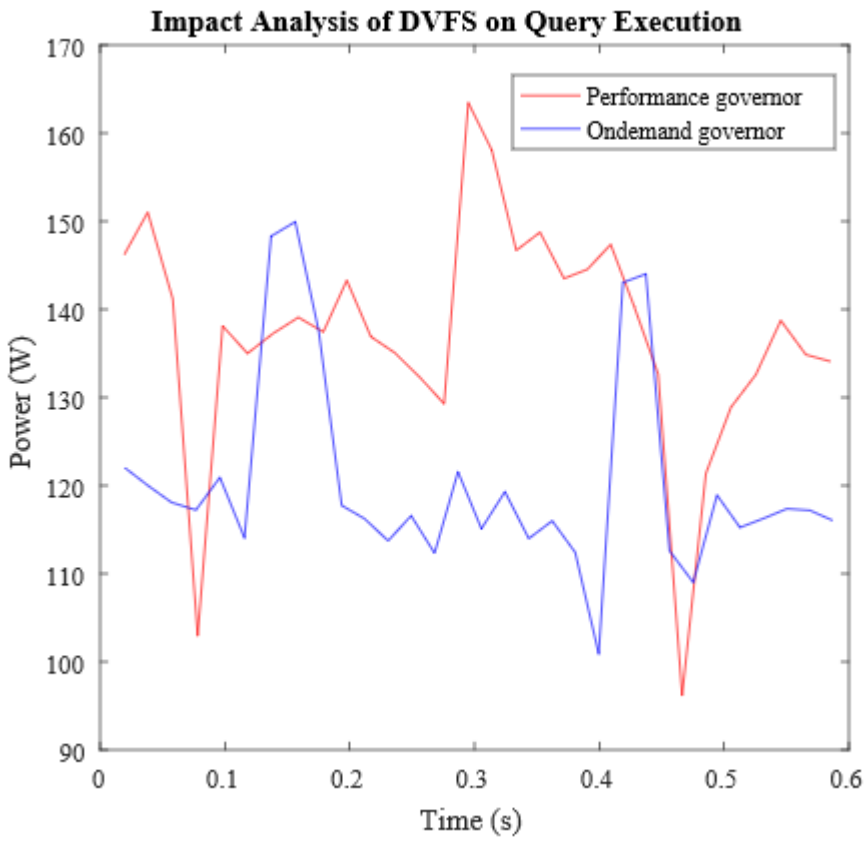

Figure 6-7: Impact of DVFS on query execution in MongoDB

We observed high power spikes in case of "performance" governor since it utilized highest CPU frequency. There was lesser power in case of "ondemand" governor.

6.4 Conclusion

In this chapter, we studied various optimization techniques that contributes to high performance and high energy efficiency. Firstly, we observed that indexes improve the efficiency of read operations by reducing the amount of data that query operations need to process. We gained a speedup of more than 270 times for indexed queries. Similarly, for indexed insert query greenup of almost 1.9 times was observed. Secondly, we found

that ordered updates were 0.97 times more energy efficient than un-ordered bulk updates. The key to increasing performance for updates depends on how fast database operations are acknowledged by the server. Thirdly, we gained a speedup of 2.4 times and greenup of 4.2 times for aggregated queries. Aggregation pipeline reduces the amount of data passing through the pipeline leading to smaller datasets that fits into the caches of MongoDB and lesser main memory references. As a result,we save power and contribute to high energy efficiency. Fourthly, we found that multi-node servers help in gaining performance by distributing work over multiple nodes, however there is an increase of power consumption because of the idle power of nodes which are not contributing to the program execution. Lastly, we also studied the impact of DVFS on query execution and found that "ondemand" governor optimizes CPU utilization, thus contribute towards energy efficient query.

7. PERFORMANCE AND ENERGY ANALYSIS OF CASSANDRA

In this chapter, we focus on analyzing Cassandra which is a NOSQL/non-relational database, using Twitter data. There are many factors that can affect database performance and responsiveness including index use, query structure, data models and application design, as well as operational factors such as architecture and system configuration.

7.1 Twitter Data Analysis using Cassandra

To analyze Twitter data using Cassandra, we streamed Twitter data using Streaming APIs and created keyspaces in Cassandra database to store them. We used PyCassa [37], is a Thrift-based python client library for Apache Cassandra to execute Cassandra queries to measure power on the Marcher System. Various optimization techniques studied have been described in the following sub-sections.

7.2 Query Optimization Techniques

This section illustrates techniques for optimizing application performance and energy efficiency for Cassandra.

Cassandra works optimally when the data we need to access is already in memory. Disks are comparatively slow. Therefore, when data needs to be read from disk, it works best when it is performed as a single sequential operation. To design an effective data model in Cassandra, it's good to keep the following best practices in mind:

- Use clustering columns in the tables so that rows are ordered on disk in the same order they are read.
- Use the built-in caching mechanisms to limit the number of reads from disk.

The following sub-sections provide in-depth analysis of various optimization strategies used to improve performance and energy efficiency of Cassandra.

7.2.1 Tuning the row caches

With row caching enabled, Cassandra will detect frequently accessed partitions and store rows of data into DRAM to reduce the data access it needs from disks. This results in some great optimizations. We can specify the number of rows to cache per partition. To study the impact of row caches, we analyzed several queries.

Row caching was quite helpful in case of optimizing an update query. The results are shown in Table 7-1 and Figure 7-1.

Table 7-1: Un-optimized vs Optimized Update Cassandra Query

Un-optimized vs Optimized Update Query						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	135.7991	51.6214	7010.14	150.8515	0.9247	163.1257
Optimized	125.5811	0.3422	42.97385			

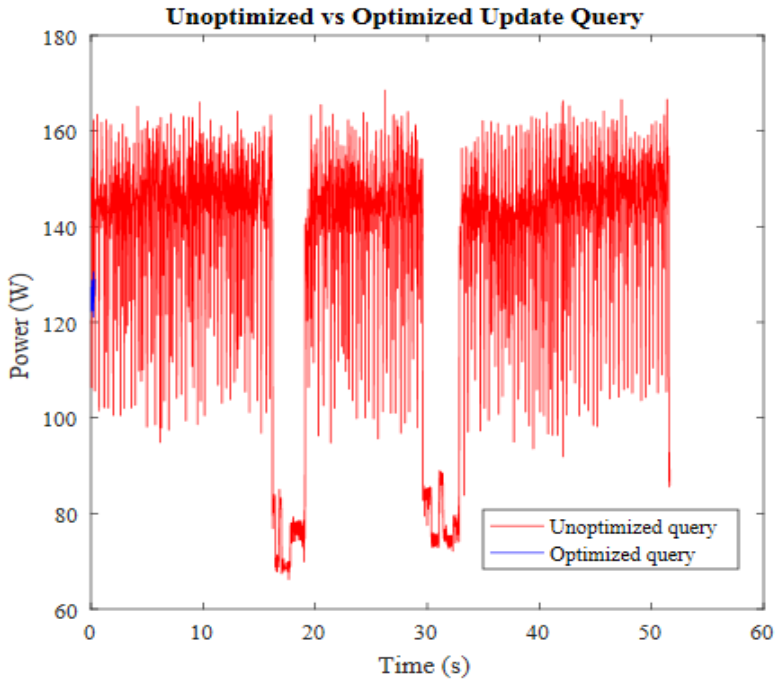

Fig 7-1: Un-optimized vs Optimized Update Cassandra Query

As indicated by Table 7-1 and Figure 7-1, execution time of optimized query is quite low as compared to unoptimized query. We observed a speedup of 150 times and greenup of 163 times in case of optimized query taking advantage of row caching.

We conducted another experiment to study the impact of row caching by executing search query. Results are provided in Table 7-2 and Figure 7-2.

Table 7-2: Un-optimized vs Optimized Search Cassandra Query

Un-optimized vs Optimized Search Query						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	125.9736	54.1462	6820.992	1.4335	0.9728	1.4736
Optimized	122.5507	37.7695	4628.679			

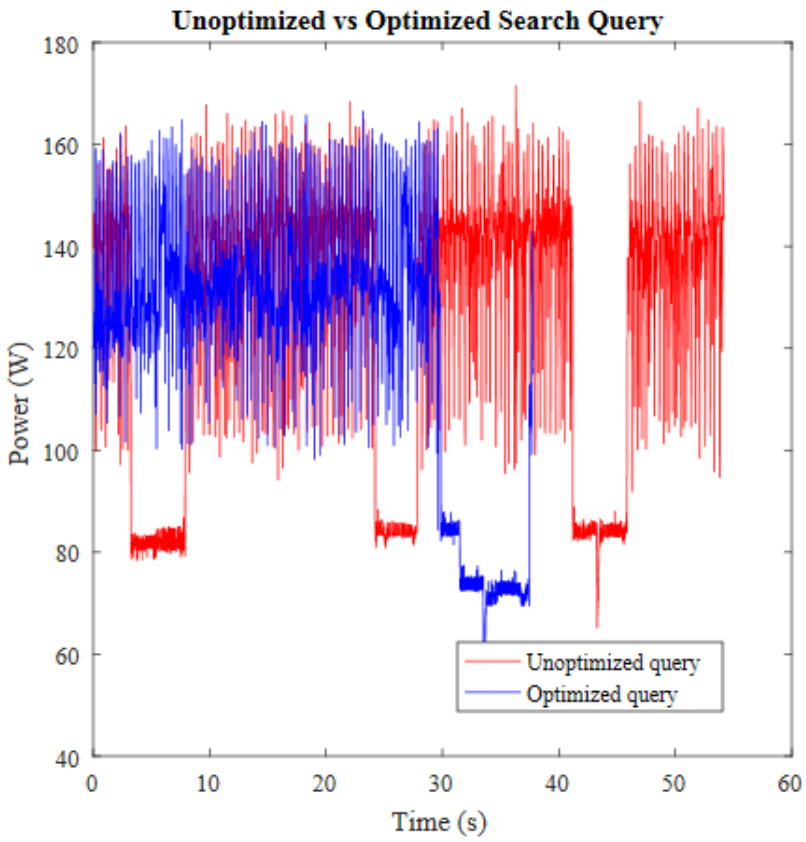

Fig 7-2: Un-optimized vs Optimized Search Cassandra Query

We observed a speedup of 1.43 times and and energy efficiency of 1.47 times in case of optimized query. Also, as per the Figure 7-2, we observe extensive parallelism in query execution path. This can be explained by the distributed nature of Cassandra database design. Cassandra is implemented as a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster.

7.2.2 Compaction

Cassandra periodically merges multiple SSTables into a smaller set of larger SSTables using a process called compaction. Compaction merges row fragments together, removes deleted columns, and rebuilds primary and secondary indexes. Since the SSTables are sorted by the row key, this merge is efficient (no random disk I/O). Once a newly merged SSTable is complete, the input SSTables are marked as obsolete and eventually deleted by the JVM garbage collection (GC) process. However, during compaction, there is a temporary spike in disk space usage and disk I/O. Compaction has impact on read performance in two ways. While a compaction is in progress, it temporarily increases disk I/O and disk utilization which can influence read performance for reads that are not fulfilled by the cache. However, after a compaction has been completed, off-cache read performance improves because there are fewer SSTable files on disk that need to be checked in order to complete a read request. Cassandra includes compaction strategies and each is optimized for a different use case. Size Tiered Compaction Strategy (STCS) triggers a compaction when multiple SSTables of a similar size are present. Additional parameters allow STCS to be tuned to increase or decrease the number of compactions it performs and how tombstones are handled. This compaction strategy is good for insert-heavy and general workloads as depicted Table 7-3 and Figure 7-3.

Table 7-3: Un-optimized vs Optimized Insert Cassandra Query

Un-optimized vs Optimized Insert Query						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	137.7482	1.2294	169.3476	2.6302	0.9347	2.8137
Optimized	128.7654	0.4674	60.18495			

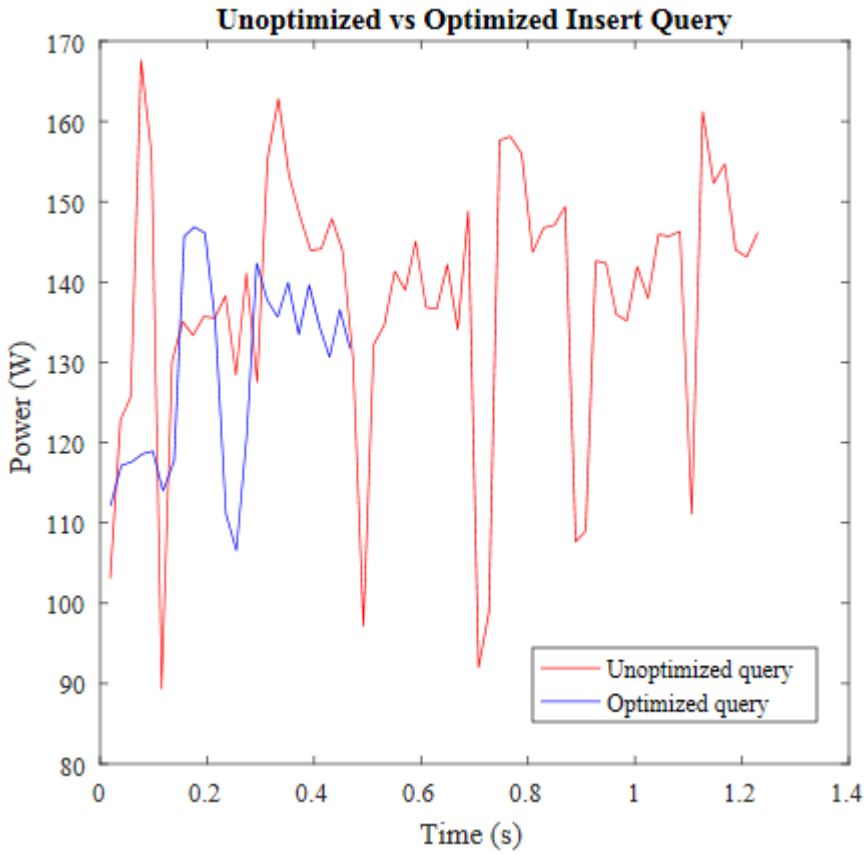

Figure 7-3: Un-optimized vs Optimized Insert Cassandra Query

As indicated in Table 7-3 and Figure 7-3, we gained a speedup of 2.6 times and greenup of 2.8 times in case of optimized query. As discussed, high performance and low power

consumption is contributed by Size Tiered Compaction Strategy (STCS).

Another strategy for compaction is Leveled Compaction Strategy (LCS). This strategy groups SSTables into levels, each of which has a fixed size limit which is 10 times larger than the previous level. SSTables are of a fixed, relatively small size (160MB by default) so if Level 1 might contain 10 SSTables at most, then Level 2 will contain 100 SSTables at most. SSTables are guaranteed to be non-overlapping within each level – if any data overlaps when a table is promoted to the next level, overlapping tables will be re-compacted. This compaction strategy is the best for read-heavy workloads (because tables within a level are non-overlapping, LCS guarantees that 90% of all reads can be satisfied from a single SSTable) or workloads where there are more updates than inserts. We used this strategy to find the most tweeted user. The following queries were executed to find the most tweeted user.

CREATE OR REPLACE FUNCTION state_group_and_count(state map<text, int>, type text) CALLED ON NULL INPUT RETURNS map<text, int> LANGUAGE java AS ' Integer count = (Integer) state.get(type); if (count == null) count = 1; else count++; state.put(type, count); return state; ' ;

CREATE OR REPLACE AGGREGATE group_and_count(text) SFUNC state_group_and_count STYPE map<text, int> INITCOND {};

select group_and_count(screen_name) from tweet.tweet_shard;

The results of the above query are displayed in Table 7-4 and Figure 7-4.

Table 7-4: Un-optimized vs Optimized Cassandra Query

Un-optimized vs Optimized queries						
Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	142.2996	831.8498	118371.9	4.5151	0.8735	5.1687
Optimized	124.3047	184.2364	22901.45			

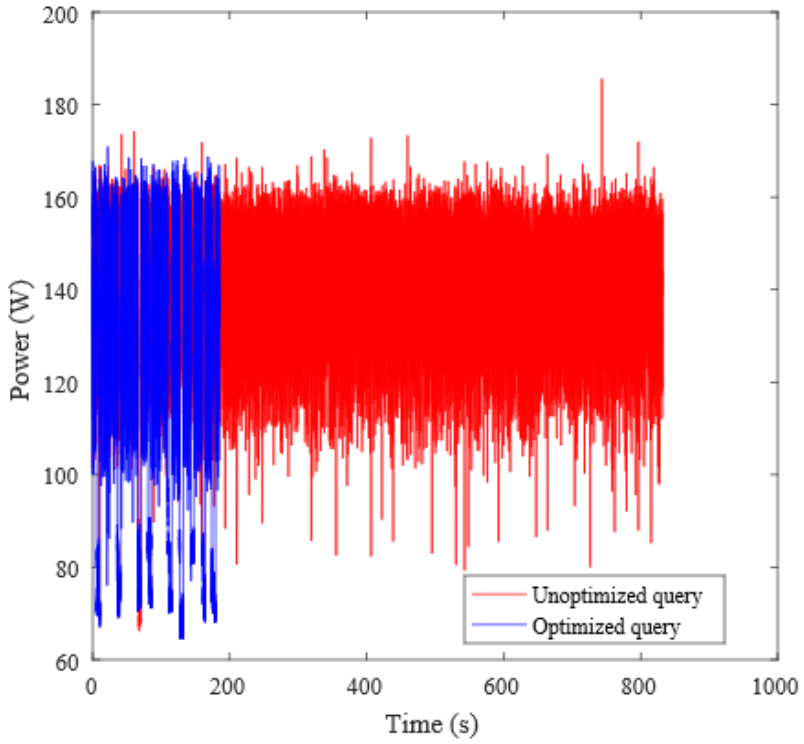

Fig 7-4: Un-optimized vs Optimized Cassandra Query

As shown in Table 7-4 and Figure 7-4, optimizeed query has a speedup of 4.5 times and greenup of 5 times as compared with non-optimized query. Also, powerup is less than 1.

This emphasizes the power saving benefits by leveraging cache size of the system and the chunk size of datasets that can minimize cache miss rate. Also, LCS guarantees that 90% of all reads can be satisfied from a single SSTable.

7.3 Impact Analysis of DVFS on optimized queries

In this section, we analyze the impact of DVFS on query execution in Cassandra. We study "performance" and "ondemand" governors for CPU frequency to understand and explore the factors that contribute to higher energy efficiency.

We conducted an experiment using an insert query to study the effect of DVFS. The results are shown in Figure 7-5.

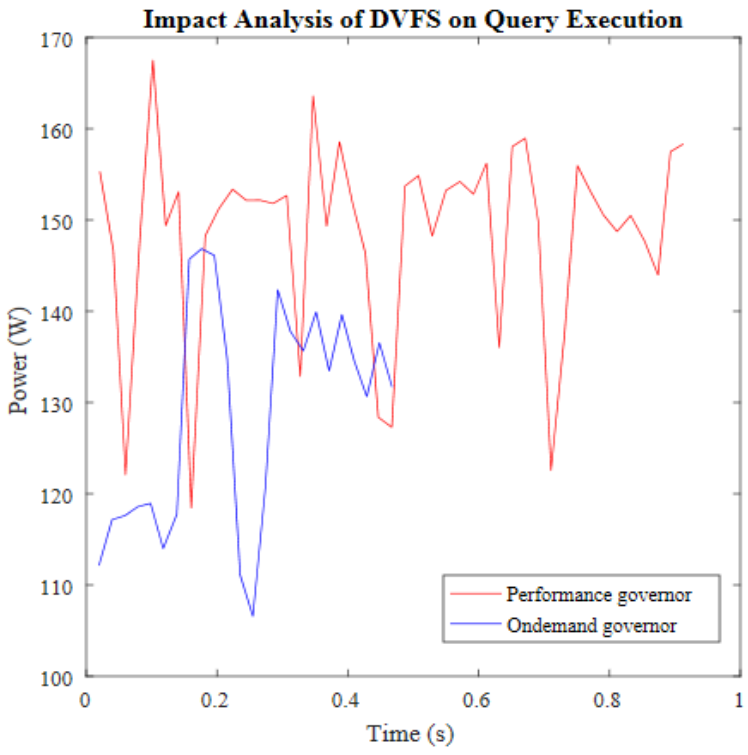

Fig 7-5: Impact of DVFS on Insert query execution in Cassandra

According to Figure 7.5, for "performance" governor, both power consumption and query execution time were high as compared to "ondemand" governor. This can be explained as sometimes higher power and processing speeds can result in higher execution time [35]. The slowdowns occur at higher frequencies when the early arrival of a single thread causes the atomic journal commit to lock with less batched threads than in the lower frequency case. In the lower frequency case, the difference between the lead thread and other threads is much smaller, therefore less time is spent in waiting. Slower processor frequencies effectively increase the number of threads that access the shared resource while reduce the overall commits required at higher processor frequencies.

We conducted another experiment to study the impact of DVFS on performance and energy efficiency using a delete query. The results are shown in Figure 7-6.

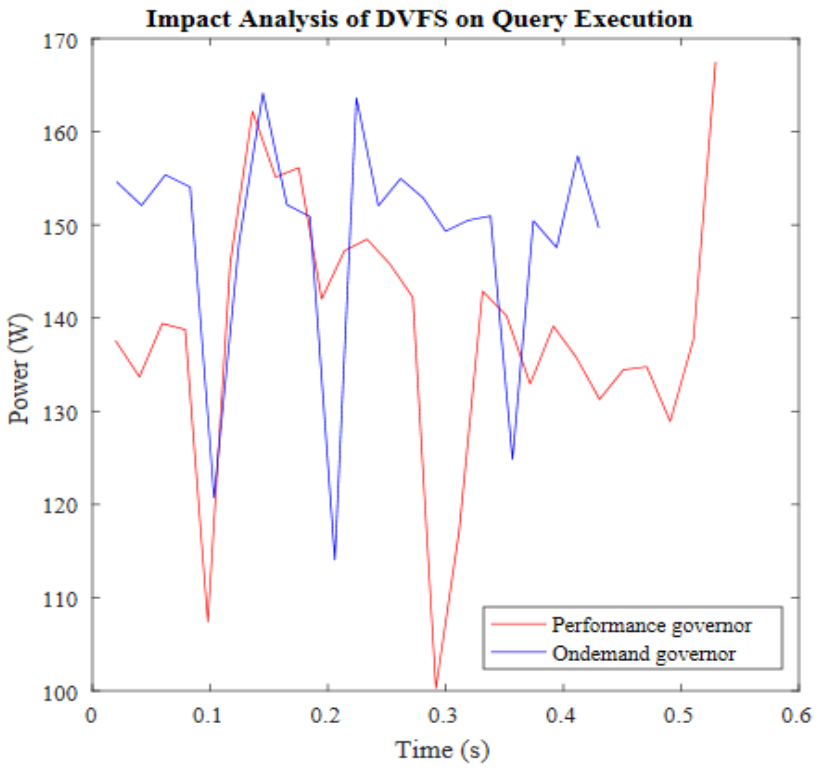

Fig 7-6: Impact of DVFS on Delete query in Cassandra

We observe sharp power spikes for both the "performance" and "ondemand" governors. However, power consumption in case of "ondemand" governor is lesser as compared to performance governor making it more energy efficient.

7.4 Conclusion

This chapter summarized various optimization techniques in Cassandra to improve performance and energy efficiency. We particularly explored techniques that contribute to better energy efficiency. Firstly, we observed that Cassandra works optimally when the data to be accessed is already in memory. This can be achieved by using clustering columns in the tables so that rows are ordered on disk in the same order they are read.

Secondly, we can use the built-in caching mechanisms to limit the number of reads from disks. Thirdly, on enabling row caching, Cassandra detects frequently accessed partitions and store rows of data into DRAM to limit reads from disk. Fourthly, compaction improves off-cache read performance since there are fewer SSTable files on disk to be read to complete read request after compaction. Lastly, we studied the impact of DVFS on energy efficiency of query execution. We found that high CPU frequency can result in slowdown sometimes due to longer waiting times for synchronizing write commits. Slower processor frequencies effectively increase the number of threads that access the shared resource by reducing the number of commits required by higher processor frequencies.

8. COMPARISION OF SQL, MONGODB AND CASSANDRA

In this chapter, we focus on cross database comparison of three databases studied in the previous chapters.

8.1 Cross database comparison using YCSB Benchmark

We performed a series of performance and energy analysis on Apache Cassandra, MongoDB and MySQL using the Yahoo! Cloud Serving Benchmark (YCSB). When it comes to performance, it should be noted that there is (to date) no single "winner takes all" among the databases studied or any other database engine for that matter. Depending on the use cases and deployment conditions, it is almost always possible for one NoSQL database to outperform another and yet lag its competitor when the rules of engagement change. While it is always recommended that anyone assessing a database's performance should test it under specific use cases and deployment conditions intended for a particular production application, general competitive benchmarks of usual-and-customary application workloads can be useful to evaluate different databases.

Each test started with an empty database which was then loaded with an initial set of randomly generated data. Once the data was loaded, each workload (described below) ran in sequence. In between each workload sometimes database health and readiness checks were performed. For example, tests for Cassandra checked for any ongoing compaction processes, and waited until those completed before continuing to the next workload.

When running the YCSB benchmark on the three databases (MySQL, Cassandra, and MongoDB), we acquired multiple power readings from the Marcher system for all the workloads, which are depicted in graphs presented below along with energy efficiency. For Workload A, energy consumption of various databases is presented in Figure 8-1.

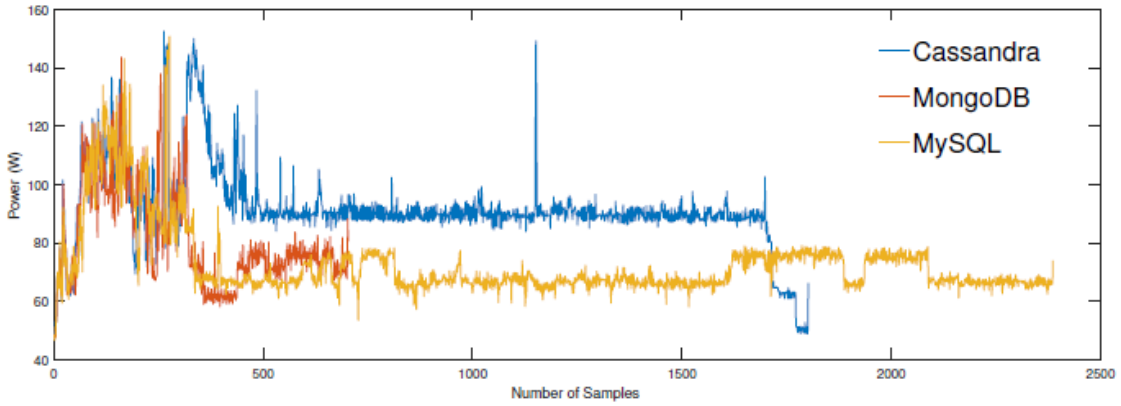

Figure 8-1: Cross-database comparison using YCSB Workload A

Although MySQL uses less power, the greater amount of time taken to complete the workload leads to a higher total energy usage. Similarly, Cassandra uses less time but has a higher power usage, so the energy efficiency is lower as well. MongoDB, on the other hand, is able to shorten its run time as well as use less power than Cassandra, which is an indication of a much more energy efficient database. Furthermore, Figure 8-1 shows that for the first approximate 350 records, all three databases use higher levels of power, an indication of parallelization. However, while Cassandra and MySQL discontinued many parallel processes, MongoDB was able to successfully complete the run quickly and with better energy efficiency. Power usage of all the workloads have been consolidated in the

Table 8-1.

Table 8-1: Cross-database comparison using YCSB

Workloads	MongoDB		Cassandra		MySQL	
	Power(W)	Time(s)	Power(W)	Time(s)	Power(W)	Time(s)
A	1211.3719	12.9392	3102.3698	33.3806	4037.7207	46.7543
B	1212.0568	12.6675	3085.7648	32.9867	4075.1242	47.2474
C	1219.7487	12.8663	3152.9750	33.8913	4097.5581	47.1394
D	1242.8923	12.9538	3070.0029	32.9745	4077.8034	47.6150
E	1224.6925	12.8508	3176.5620	34.13622	4169.83203	48.2778

It is evident from Table 8-1 that MongoDB is most efficient for almost all the workloads in terms of performance as well as energy. MongoDB consumes approximately 85% less power than Cassandra and almost 92% less power than MySQL for the same workloads run under similar conditions.

8.2 Cross database comparison using Twitter data

In this section, we conduct a cross database comparison using Twitter data. To measure the energy efficiency of three databases, we used query to find the most tweeted user. The results of the query are presented in Table 8-2 and Figure 8-2.

Table 8-2: Cross-database comparison for most tweeted user

Most Tweeted User						
Database	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
MySQL	62.361	4.0827	254.6013	45.12609	1.993308	89.95021
MongoDB	76.6249	117.123	775.9282	1.573016	1.622249	29.5149
Cassandra	124.3047	184.2363	22901.44			

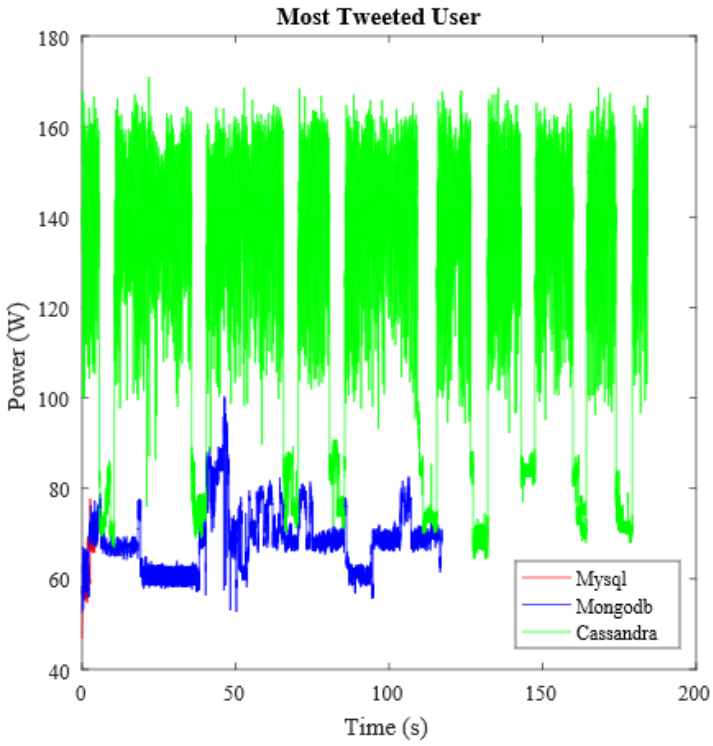

Figure 8-2: Cross-database comparison to find the most tweeted user

As per the power readings presented in Table 8-2, MySQL seems to be most efficient in terms of execution time as well as power whereas Cassandra is least efficient. MySQL has a speedup of 45 times and greenup of 90 times as compared to Cassandra. MongoDB

seems to be less efficient than MySQL as aggregation is used to find the most tweeted users which is slower as compared to groupby and order by functions used in MySQL to carry out the same task. In Cassandra, although extensive parallelism is involved in executing the query to find the most tweeted user, there is an overhead in additional scripts used for carrying out these tasks due to the insufficient support of aggregation functions. Cassandra is particularly not suitable for such kind of querying.

We futher extended our experiment to analyze the behaviour of update query in all three databases. Results are shown in Table 8-3 and Figure 8-3.

Table 8-3: Cross-database comparison of Update query

Update Query						
Database	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
MySQL	63.6411	6.91431	440.0343			
MongoDB	78.7984	0.5796	45.67155	11.92945	0.807645	9.634757
Cassandra	69.2863	2.1702	150.3651	3.186024	0.918524	2.926439

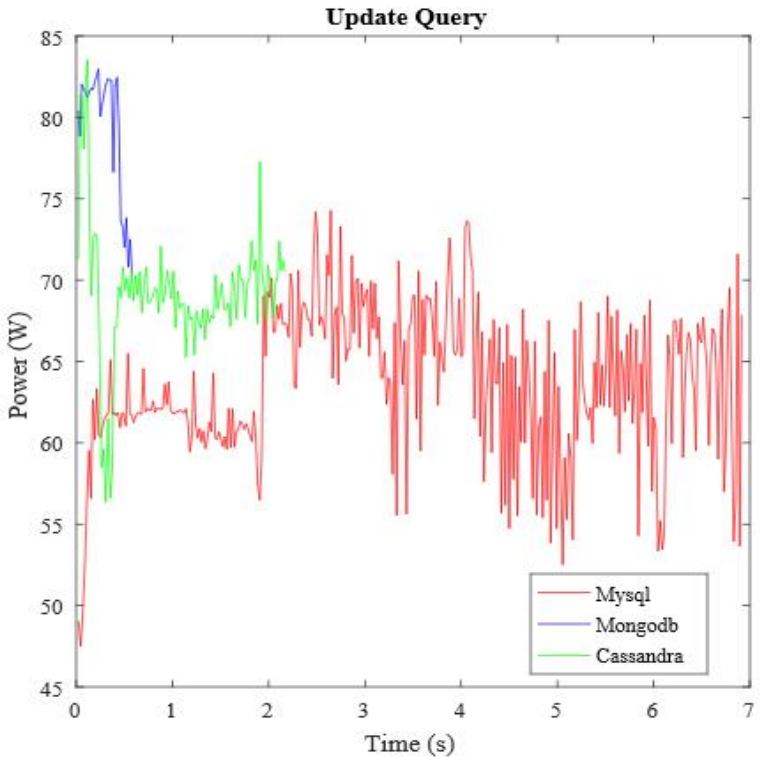

Fig 8-3: Cross-database comparison of Update query

As shown in Table 8-3 and Fig 8-3, we can observe that Mongodb is most energy efficient and MySQL is least efficient. We do not observe any drastic difference between the power consumption of the three databases, however MongoDB seems to be quite efficient in terms of query execution time. MongoDB has a speedup of 11.9 times and greenup of 9.6 times as compared to other databases. Also, we observe powerup >1 which indicates that we achieved better performance at the expense of consuming more power. Since the Speedup obtained is more than the power penalty spent, the optimized code still saves energy. This type of behavior is particularly observed in case of parallel executions.

Our next experiment, compared the energy efficiency of the delete query using three databases. Results are shown below in Table 8-4 and Figure 8-4.

Table 8-4: Cross-database comparison of Delete query

Delete Query						
Database	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
MySQL	56.7188	0.3748	21.25821	1.2916	2.135482	2.7582
MongoDB	79.5758	0.2348	18.6844	2.0617	1.522096	3.1381
Cassandra	121.122	0.4841	58.63516			

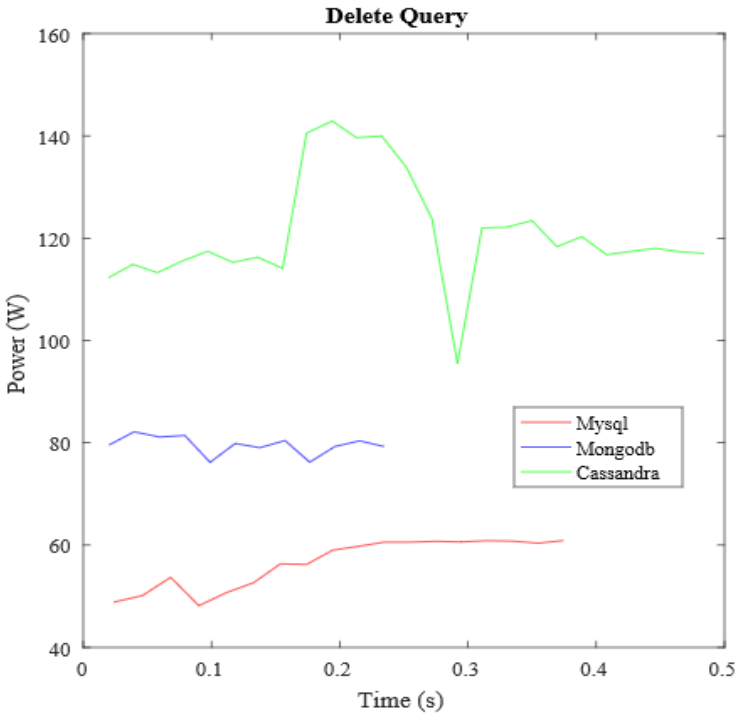

Fig 8-4: Cross-database comparison of Delete query

As shown in Table 8-4 and Figure 8-4, again Mongodb is the most efficient database in case of delete query exectution and Cassandra seems to be the least efficient one. Indexing in Mongodb contributes to its shorter execution time and lower power consumption, thereby making it highly energy efficient.

Another experiment was done to sudy the behaviour of insert query for all three databases. The results are presented in Table 8-5 and Figure 8-5.

Table 8-5: Cross-database comparison of Insert query

Insert Query						
Database	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
MySQL	56.3147	0.3196	17.99818	1.4624	2.286533	3.3439
MongoDB	57.3588	0.3542	20.31649	1.3195	2.244911	2.9623
Cassandra	128.7654	0.4674	60.18495			

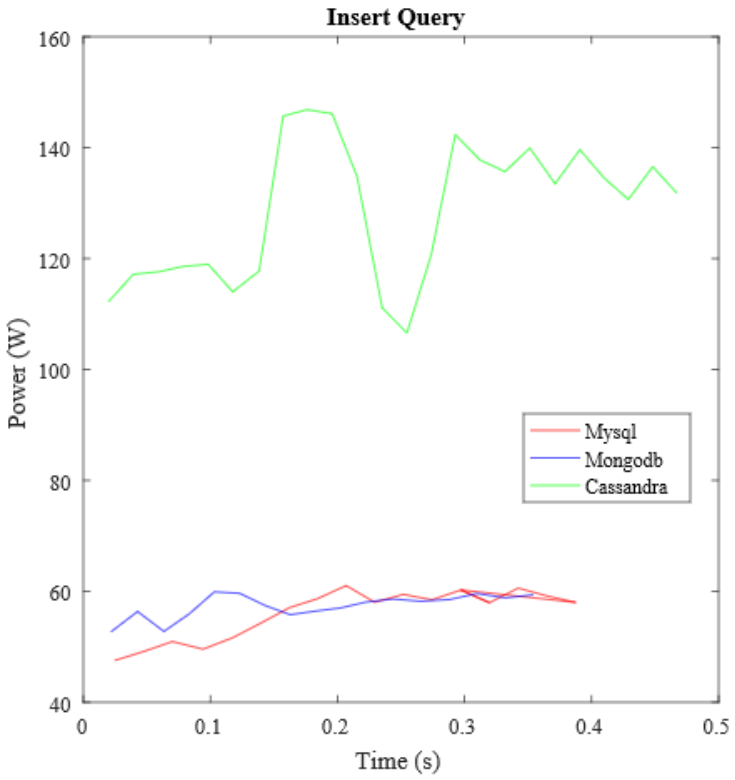

Table 8-5: Cross-database comparison of Insert query

The results of insert query are shown in Table 8-5 and Fig 8-5. We observe that MongoDB and MySQL have overlapping power consumptions. However, MySQL is more efficient with a speedup of 1.4 times and greenup of 3.3 times as compared to Cassandra.

Finally, we compared the databases using search query. Results are presented in the following Table 8-6 and Figure 8-6.

Table 8-6: Cross-database comparison of Search query

Search Query						
Database	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
MySQL	130.0075	2.2467	292.0879	0.9426	16.8099	15.8457
MongoDB	70.1851	0.5895	41.37412	1.7461	64.0659	111.8661
Cassandra	122.5507	37.7669	4628.36			

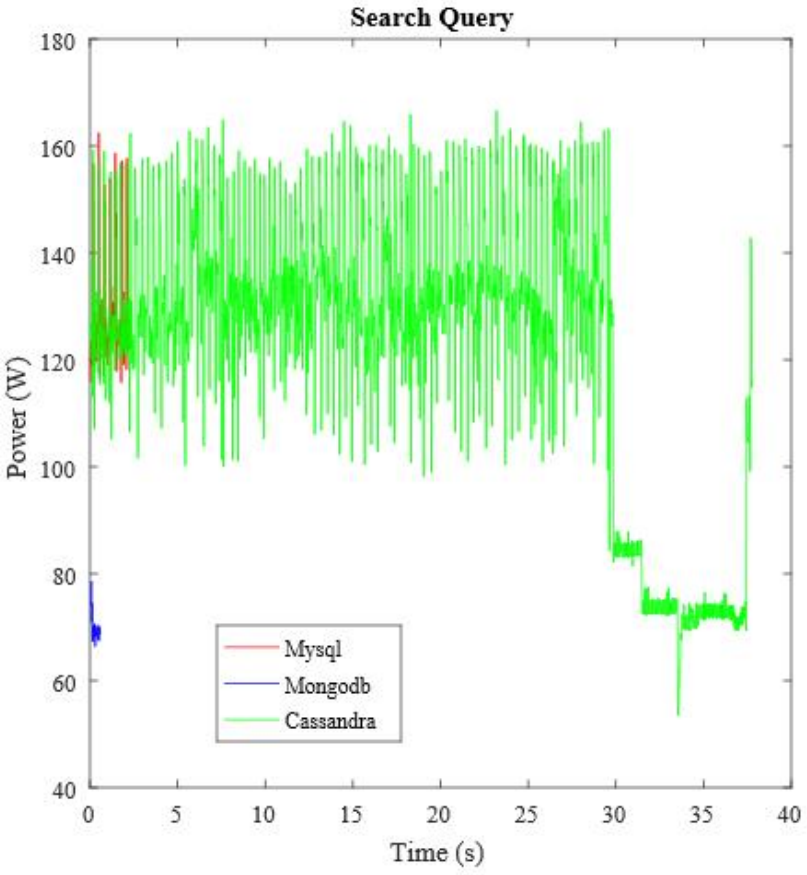

Fig 8-6: Cross-database comparison of Search query

As indicated by results given in Table 8-6 and Figure 8-6, Mongodb performs the best and consumes the least amount of energy. It takes advantage of its full indexing technique to attain high energy efficiency. We observe speedup of 64 times and greenup of 111 times when compared to Cassandra.

9. CONCLUSION

In this thesis, we conducted a comprehensive study on optimizing databases for higher performance and lower energy consumption via software approaches. We first developed a tool that can obtain accurate real-time power consumption information of various queries running on both relational databases and NoSQL databases. We then studied a series of optimization techniques (for MySQL, MongoDB, and Cassandra respectively) that can reduce energy consumption without compromising performance. Last but not the least, we compare the performance and energy efficiency of all three databases and evaluate their advantages and disadvantages at different scenarios.

A number of important conclusions can be drawn from this research project.

First, MongoDB is a very efficient NoSQL database. It exceeds MySQL and Cassandra for almost all YCSB workloads and most of the Twitter data queries in terms of both performance and energy efficiency. However, there are scenarios where MySQL proved to be more efficient than NoSQL databases due to its simplicity and relational design. Meanwhile, Cassandra has excellent single-row read performance as long as eventual consistency semantics are sufficient for the use-case but its performance degrades as reads spans to multiple rows. It should be noted that the YCSB benchmark as well as the Twitter data queries do not cover all possible workloads. There is no single winner in all tasks and scenarios. Depending on the use cases and deployment conditions, it is almost

always possible for one database to outperform another and yet lag its competitor when the rules of engagement change.

Second, performance optimization is neither equivalent to nor conflicting with energy efficiency optimization. Using the Greenup, Powerup and Speedup metrics, we have found numerous examples where performance and energy efficiency are improved simultaneously (i.e. a win-win situation) and many of these cases showed that the performance improvement is not proportional to energy efficiency improvement (i.e. they are not equivalent). It appears that optimization techniques that can improve the data access rate at caches are more likely to improve energy efficiency more than performance because the power cost to access data is reduced as well.

Third, DVFS has a large impact on the energy efficiency of databases. In most cases, DVFS helps to improve energy efficiency without compromising performance. We also observed that high CPU voltage and frequency can sometime hurt both performance and energy efficiency. It is highly recommended that DVFS should be enabled whenever possible, which is probably the easiest way for database administrators to save energy without degrading performance.

10. FUTURE WORK

In the future, we will expand our experiments to more databases with bigger and more complex data sets. With wider range of experiments, we will find more optimization techniques leading to energy efficient databases with high performance. In this study, we performed our experiments on maximum of two nodes. In the future, we would like to extend our experiments to a cluster of nodes to analyze performance and energy efficiency tradeoffs.

REFERENCES

[1] "Google Search Statistics." Internet Live Stats. N.p., 13 July 2016. Web. 13 July 2016.

[2] "Stream Analytics Twitter Sentiment Analysis Trends." Jeff Stokes., 29 September 2016.

[3] Oracle8 Parallel Server Concepts & Administration Documentation,Release 8.0

[4] BF. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. "Benchmarking Cloud Serving System with YCSB." Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10, 2010, pp.143-154.

[5] https://en.wikipedia.org/wiki/Dynamic_frequency_scaling

[6] Etienne Le Sueur and Gernot Heiser, "Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns".

[7] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia Molina, Johannes Gehrke, Le G, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Raghu Ramakrishnan, Sunita Sarawagi, Michael Alexander S. Szalay, Gerhard WeikumThe Claremont Report on Database Research, 2008

[8] "Electric Power Monthly." EIA. N.p., n.d. Web. 16 Sept. 2016 https://www.eia.gov/electricity/monthly/epm table grapher.cfm?t=epmt 5 3

[9] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In CIDR, 2009.

[10] D. Tsirogiannis, S. Harizopoulos and M. Shah. "Analyzing the energy efficiency of a database server." Proceedings of the 2010 international conference on Management of data - SIGMOD '10, 2010, pp.231-232.

[11] Willis Lang, Stavros Harizopoulos and Jignesh M. Patel, "Towards Energy-Efficient Database Cluster Design", University of Wisconsin,2012.

[12] Z. Xu, Y. Tu, and X. Wang. "Power-Aware Throughput Control for Database Management System." 10th International Conference on Autonomic Computing (ICAC `13), 2013, pp. 315-324.

[13] Balaji Subramaniam and Wu-chun Feng."On the Energy Proportionality of Distributed NoSQL Data Stores". Department. of Computer Science, Virginia Tech.

[14] T. Harder, V. Hudlet, and D. Schall. "Enhancing Energy Efficiency of Database Applications Using SSDs." Proceedings of the Third C* Conference on Computer Science and Software Engineering C3S2E '10, 2010, pp. 1-6.

[15] G. Graefe. "Database Servers Tailored to Improve Energy Efficiency." Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management SETMDM `08, 2008, pp. 24-28.

[16] R. Gon_calves, J. Saraiva, and O. Belo. "Defining Energy Consumption Plans for Data Querying Processes." IEEE Fourth International Conference on Big Data and Cloud Computing (BdCloud), 2014, pp. 641-647.

[17] Y. Tu, X. Wang, B. Zeng, and Z. Xu. "A System for Energy-Efficient Data Management." ACM SIGMOD Record, vol. 43(1), pp. 21-26, Mar. 2014.

[18] S. Harizopoulos, MA. Shah, J. Meza, and P. Ranganathan. "Energy Efficiency: The New Holy Grail of Data Management System Research." 4th Biennial Conference on Innovative Data Systems Research (CIDR), 2009.

[19] O. Belo, R. Gon_calves, and J. Saraiva. "Establishing Energy Consumption Plans for Green Star-Queries in Data Warehousing Systems." 2015 IEEE International Conference on Data Science and Data Intensive Systems, 2015, pp. 226-231.

[20] R. Goncalves, J. Saraiva, and O. Belo. "Defining Energy Consumption Plans for Data Querying Processes." IEEE Fourth International Conference on Big Data and Cloud Computing (BdCloud), 2014, pp. 641-647.

[21] Z.Xu. "Building a Power-Aware Database Management System." Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research (IDAR 2010), 2010, pp. 1-6.

[22] Z. Xu, Y. Tu, and X. Wang. "PET: Reducing Database Energy Cost via Query Optimization. "VLDB Journal, vol. 5(12), pp. 1954-1957, Aug. 2012.

[23] http://www.slideshare.net/joshmckenzie/Cassandra-21-read-write-path

[24] Kandhan, and JM. Patel. "Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race." IEEE Computer Society Technical Committee on Data Engineering, 2011.

[25] S. Abdulsalam, Z. Zong and Q. Gu, "Using the Greenup, Powerup, and Speedup metrics to evaluate software energy efficiency." Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International, 2015.

[26] "Indexes." MongoDB Documentation. N.p., n.d. Web. 11 Aug. 2016. https://docs.MongoDB.com/manual/indexes/

[27] "Bulk Write Operations." MongoDB Documentation. N.p., n.d. Web. 11 Aug. 2016. https://docs.MongoDB.com/manual/core/bulk-write-operations/

[28] "Bulk()." MongoDB Documentation. N.p., n.d. Web. 11 Aug. 2016. https://docs.MongoDB.com/manual/reference/method/Bulk/

[29] R. Niemann, N. Koratis, R. Zicari, and R. Gobel. "Does query performance lead to energy efficiency? A comparative analysis of energy efficiency of database operations under different workload scenarios." Internet: http://arxiv.org/abs/1303.4869, 2013 [Jul. 05, 2016]

[30] Z. Xu, Y. Tu, and X. Wang." Exploring Power-Performance Tradeoffs in Database Systems." ICDE Conference, 2010.

[31] Z. Xu, Y. Tu, and X. Wang. "Online Energy Estimation of Relational Operations in Database Systems." IEEE Transactions on Computers, vol. 64(11), pp.3223-3236, Nov. 2015.

[32] D. Florescu and D. Kossmann. "Rethinking Cost and Performance of Database Systems."ACM Sigmod Record, vol. 38(1), pp. 43-48, Mar. 2009.

[33] M. Korkmaz, A. Karyakin, M. Karsten. “Towards Dynamic Green-Sizing for Database Servers." Int'l Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS), 2015.

[34] https://www.thoughtworks.com/insights/blog/nosql-databases-overview

[35] Hung-Ching Chang, Bo Li, Matthew Grove and Kirk W.Cameron, “How Processor Speedups Can Slow Down I/O Performance”

[36] Ashish Mishra, Nilay Khare. “Analysis of DVFS Techniques for Improving the GPU Energy Efficiency”. Open Journal of Energy Efficiency, 2015, 4, 77-86

[37] http://pycassa.github.io/pycassa/

[38] https://api.mongodb.com/python/current/