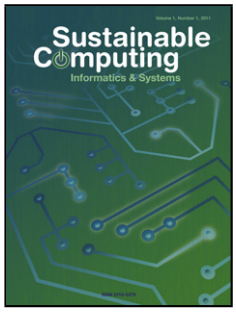

Title: Improving the energy efficiency of relational and NoSQL databases via query optimizations

Authors: Divya Mahajan, Cody Blakeney, Ziliang Zong

PII: S2210-5379(18)30111-2
DOI: https://doi.org/10.1016/j.suscom.2019.01.017
Reference: SUSCOM 315

To appear in:

Received date: 25 March 2018
Revised date: 19 September 2018
Accepted date: 30 January 2019

Please cite this article as: { https://doi.org/

Improving the Energy Efficiency of Relational and NoSQL Databases via Query Optimizations

Divya Mahajan, Cody Blakeney, and Ziliang Zong
Department of Computer Science, Texas State University
divya.mahajan30@gmail.com cjb92@txstate.edu ziliang@txstate.edu

Highlights of key contributions:

1. We conduct a comprehensive study (first of its kind to the best of our knowledge) on the impact of numerous query optimization techniques for MySQL, MongoDB, and Cassandra on performance, power, and energy consumption.
2. We develop an easy-to-use power measurement tool that can accurately measure the fine- grained real-time power consumption of various queries running on MySQL, MongoDB, and Cassandra.
3. We present a methodology using Speedup, Powerup, and Greenup to reveal the correlations between performance, power, and energy efficiency of relational and NoSQL databases.
4. We prove that in some scenarios energy efficiency optimization is neither merely a byproduct nor a conflicting goal of performance optimization. There are optimization techniques that can help energy more than performance.
5. We compare the performance and energy characteristics of relational and NoSQL databases using the Yahoo! Cloud Server Benchmark (YCSB) and ~100GB of customized Twitter data.
6. We reveal the impact of different DVFS policies on the performance and energy efficiency of MySQL, MongoDB and Cassandra.

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 a few studies have investigated the energy efficiency of database systems. To the best our knowledge, there are currently no comprehensive studies that analyze the impact of query optimizations on performance and energy efficiency across both relational 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 paper, we investigate a series of query optimization techniques for improving the energy-efficiency of relational databases and NoSQL databases. 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 various optimization techniques. We conduct cross database analysis on relational database (MySQL) and NoSQL based databases (MongoDB and Cassandra) to compare their performance and energy efficiency. Additionally, we study 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 are able to achieve significant energy savings without performance degradation. Moreover, we investigate the impact of Dynamic Voltage and Frequency Scaling (DVFS) on the performance and energy efficiency of MySQL, MongoDB and Cassandra.

***Keywords*—**Energy-efficiency; Relational Databases; NoSQL Databases; MySQL; MongoDB; Cassandra; DVFS

I. INTRODUCTION

Energy efficiency has become a critical design and operational criteria for computing systems ranging from data centers, small clusters, stand-alone servers, to mobile and embedded devices. Unfortunately, Database Management Systems (DBMS) running in server environments have largely ignored the energy efficiency issue, but we can no longer afford such oversight. For example, Google currently processes about 40,000 queries per second or 3.5 billion queries per day [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 makes

Twitter another exemplary big dataset where many social media analytics tools 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 databases to be highly efficient in terms of both performance and energy efficiency.

Despite the fact that the majority of data is stored and processed using different forms of databases, either relational or NoSQL, the current academic research and industrial practices on databases emphasize more on performance than energy efficiency. To the best our knowledge, there are currently no comprehensive studies that analyze the impact of query optimizations on the performance and the energy efficiency across both relational 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 tools and analysis methodologies that can be applied to various databases. However, understanding the energy benefit of various query optimizations is paramount for both relational and NoSQL databases, especially for servers that response to millions of queries on a daily basis. Maximizing the energy efficiency of each single query could significantly reduce the accumulated cost of large-scale database systems.

Meanwhile, it is worth noting that there are ongoing debates for improving database energy efficiency. Numerous database researchers believe that hardware optimizations (e.g. replacing HDDs with SDDs [2] [21]) are more effective than software optimizations. Some studies conclude that energy savings are merely a byproduct of performance optimizations [3]. Nonetheless, 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). They believe that tradeoffs are inevitable in a multi-objectives optimization problem. All these arguments are reasonable in certain scenarios but they do not reveal the whole picture of database optimizations. For example, the best-case scenario, where optimizations can reduce energy consumption without degrading performance and energy saving is larger than performance improvement, has been overlooked due to the lack of measurement tools and analysis methodologies. In fact, researchers may doubt the existence of such best-case scenarios because it sounds too idealistic.

In this paper, we strive to 1) explore a simple but appropriate methodology that can reveal the correlations of performance, power, and energy in the context of relational and NoSQL database optimizations; 2) find out whether or not performance improvement is linear to energy reduction; 3) investigate cases where energy savings can be achieved without degrading performance; 4) investigate cases where more energy reduction can be observed discard the portions caused by performance improvement; 5) compare the performance and energy efficiency of different relational databases and NoSQL databases; and 6) evaluate the impact of DVFS on the performance and energy efficiency of relational and NoSQL Databases.

Our contributions can be summarized as follows:

- We conduct a comprehensive study (first of its kind to the best of our knowledge) on the impact of numerous query optimization techniques for MySQL, MongoDB, and Cassandra on performance, power, and energy consumption.
- We develop an easy-to-use power measurement tool that can accurately measure the fine-grained real-time power consumption of various queries running on MySQL, MongoDB, and Cassandra.
- We present a methodology using Speedup, Powerup, and Greenup to reveal the correlations between performance, power, and energy efficiency of relational and NoSQL databases.
- We prove that in some scenarios energy efficiency optimization is neither merely a byproduct nor a conflicting goal of performance optimization. There are optimization techniques that can help energy more than performance.
- We compare the performance and energy characteristics of relational and NoSQL databases using the Yahoo! Cloud Server Benchmark (YCSB) [8] and ~100GB of customized Twitter data.
- We reveal the impact of different DVFS policies on the performance and energy efficiency of MySQL, MongoDB and Cassandra.

The rest of the paper is organized as follows. Section II discusses the related work. Section III briefly introduces relational and NoSQL databases. Section IV presents the configuration of our system, the evaluation metrics, and the YCSB benchmark and Twitter datasets used for generating experimental results. Section V evaluates the performance and energy efficiency impact of various query optimizations for NoSQL databases (MongoDB and Cassandra). Section VI evaluates the performance and energy efficiency impact of various optimizations for a relational database (MySQL). Section VII compares the performance and energy efficiency of relational and NoSQL databases. Lastly, Section VIII concludes the paper, discusses the limitations of our current study, and points out future work.

II. Related Work

The majority of existing literature on database optimization primarily focused on performance with only very few studies investigated the energy efficiency aspect of database system. With the emergence of big data, energy efficiency started to attract more attentions recently as an equally important optimization goal due to the unimaginable growth rate of data size and rapidly increased ownership and operation cost of large-scale databases.

Schall et al. showed that the energy efficiency of database systems can be greatly enhanced by replacing hard drives with solid state disks (SSDs) [21]. Beckmann et al. endorsed this argument by proving that the sorting algorithms can be improved by a factor of 3 using SSDs [2]. Although faster and low power hardware can improve energy efficiency of databases, upgrading hardware is costly in general and may interrupt services occasionally. This issue was pointed out by several researchers and they argued that energy efficiency can be improved by software approaches. For example, Graefe argued that both hardware and software optimizations can contribute to energy efficiency [22]. Niemann et al. conducted a comprehensive study on the impact of changing data types, joining tables, and eliminating duplicates on the energy efficiency of *PostgreSQL* database. They concluded that energy savings are often derived from performance improvement and sometimes are conflicting with performance improvement. However, their study is limited to *PostgreSQL* and does not include NoSQL databases. Harizopoulos et al. argued that hardware was only part of the solution. They analyzed the influence of various factors (e.g. energy-aware system wide knobs tuning, query optimization, algorithm design, resource consolidation, etc.) and concluded that software approaches also played an important role [23] in saving energy in database systems. Tsirogiannis et al. followed this line of work and conducted a detailed investigation on analyzing the energy characteristics of basic operations of relational databases [24]. They found that most energy efficient configuration is often the highest performing one. Unfortunately, their experiments are only for relational databases and they did not quantitatively reveal the correlations between performance, power, and energy. For query optimizations, Xu et al. proposed an energy-aware query optimizer in [25], an energy-aware query optimization framework in [28], and energy cost estimation models of query plans in [30], which aim to select query plans based on both execution time and estimated power consumption. Lang et al. proposed a technique to reorder query executions for energy conservation [26]. They, along with Florescu et al., suggested that database research community should pay more attention to cost optimization in addition to performance improvement [29] [31].

All the aforementioned literature targeted on conventional SQL databases. We were only able to find one paper that focused on the energy efficiency of NoSQL databases [27]. In this paper, Li et al. analyzed the waiting energy consumption of NoSQL databases and found that the database servers wasted energy when they should be in idle state but actually did not. Their approach aimed to maximize the idle time of NoSQL database servers for energy conservation, which was orthogonal to our proposed approach. Another contribution of the paper was to investigate the efficacy of existing benchmarks for NoSQL database evaluation. They also concluded that the test cases provided by the YCSB benchmark only include basic insert, read, update, and scan operations, which are insufficient to comprehensively evaluate NoSQL database.

To the best of our knowledge, our work is the first of its kind to propose a simple but applicable methodology in evaluating performance and energy efficiency of both relational and NoSQL databases. This methodology and the proposed Speedup, Powerup and Greenup metrics can be directly applied to other databases besides MySQL, MongoDB, and Cassandra. Our work provided strong evidence to prove that energy efficiency optimization of databases is not straightforward and many different scenarios should be considered case by case. Additionally, we clarified some fallacies and misconceptions in optimizing databases for better energy efficiency by verifying that there are optimizations that can help improve performance and lower power consumption simultaneously, which will lead to the greatest energy savings.

III. Introduction to Relational and NoSQL databases

A. *Relational Databases*

Relational databases have dominated the software industry for the past several decades by providing persistent data storage, concurrency control, transactions, and having standard interfaces for data integration and reporting. A relational database organizes data as a set of formally described tables from which data can be accessed or reassembled without having to reorganize the tables. Each table contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns.

B. *NoSQL Databases*

Relational databases are not designed to store schema-less data and have limitations in scaling out to multiple servers. NoSQL databases can store and quickly process large volume of schema-less data without using the relational model. It provides better support to scale-out architectures using open source software, commodity servers, and cloud computing instead of large monolithic servers and storage infrastructure used in relation databases. There are four popular data models in today's NoSQL databases: key-value, document, column-family, and graph. Here we skip the introduction of the key-value model and graph model because MongoDB and Cassandra use the document model and the column family model respectively.

1) Document Databases

The document NoSQL database stores and retrieves data in the form of documents (e.g. XML, JSON, BSON, etc.). These documents are self-describing hierarchical tree data structures which can consist of maps, collections, and scalar values. MongoDB is an open-source document database that provides high performance, high availability, and automatic 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. 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.

2) Column Family Databases

Column family databases store data in column families as rows, which have many columns associated with a row key. Column families are groups of related data that is often accessed together. Each column family can be compared to a container of rows in a relational database table where the key identifies the row and the row consists of multiple columns. The difference is that various rows do not necessarily have the same columns, and columns can be added to any row at any time without having to add it to other rows. Cassandra is a typical column family NoSQL database, which is designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. In Cassandra, all nodes play an identical role and data is written in a way that provides both full durability and high performance. There is no master node and all nodes communicate with each other via a distributed, scalable protocol called "gossip". To improve availability, each data item can be replicated at N different nodes, 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 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 the size of the *memtable* exceeds a configurable threshold, the data is written to an immutable file on disk called an *SSTable*.

IV. SYSTEMS, METRICS AND BENCHMARKS

A. *Marcher System Configurations*

Our experiments are generated on the NSF funded power measurable high performance computing system (codenamed Marcher [7]). Each Marcher server contains two Intel Xeon E5-2600 processors with a total of 16 cores, a 32GB of DRAM, a K20 GPU and an Intel Xeon Phi coprocessor, as well as hybrid storage with hard drives and SSDs. We develop an easy-to-use API that integrates and synthesize the power readings from multiple sources, which include the Intel RAPL interface [4], the NVIDIA Management Library (NVML) interface [5], the Intel MICAccess API [6], and the Power Data Acquisition Card (PODAC) respectively (see Fig.1). Through this API, we can easily collect fine-grained power data at real-time of all major components (e.g. CPU, DRAM, Disk, GPU, and Xeon Phi). Since GPU and Xeon Phi are not used for NoSQL databases and the disk power remains identical most of the time, only the CPU and DRAM power are calculated in our experiments as total power consumption. More details about the Marcher system can be found in [7].

B. *Speedup, Powerup, and Greenup Metrics*

We propose to use the Speedup, Powerup, and Greenup metrics to evaluate each optimization and reveal the impact of a specific optimization on performance, power, and energy. For each experiment, we create an un-optimized query as the baseline for comparison to a corresponding optimized query. Speedup is defined in Equation (1) as

$$\text{Speedup} = T_\phi / T_o \quad (1)$$

where T_ϕ is the execution time of the un-optimized query while T_o is the execution time of the optimized query. Similarly, Greenup is the ratio of the energy consumption of the un-optimized query (E_ϕ) over the energy consumption of the optimized query (E_o), as show in Equation (2).

$$\text{Greenup} = E_\phi / E_o \quad (2)$$

$$\text{Powerup} = P_o / P_\phi \quad (3)$$

Powerup is defined as the ratio of the average power consumed by the optimized query over the average power consumed by the un-optimized query (see Equation (3)).

The Speedup, Powerup and Greenup metrics provide a simple but powerful methodology to evaluate the impact of optimizations on performance, power and energy efficiency. Specifically, we can categorize the impact of optimized query on performance, power and energy as follows:

- Speedup > 1, Powerup < 1, and Greenup > 1 (Case 1): Optimizations improve performance and consume less power. This is the ideal scenario because energy saving is not only achieved without sacrificing performance but also is greater than performance improvement.
- Speedup > 1, Powerup = 1, and Greenup > 1 (Case 2): Optimizations improve performance but consume the same amount of power. All energy savings come from the performance improvement.
- Speedup > 1, Powerup > 1, and Greenup > 1 (Case 3): Optimizations achieve better performance at the expense of consuming more power. Since the Speedup obtained is greater than the power penalty, the optimized query still saves energy.
- Speedup < 1, Powerup < 1, and Greenup > 1 (Case 4): Optimizations reduce power consumption at the cost of performance degradation. Since the power saving is greater than the performance penalty, the optimized query still saves energy.

C. Benchmarks

The open source Yahoo! Cloud Server Benchmark (YCSB) [8] and ~100GB Twitter data are used in our experiments to comprehensively study the impact of various optimizations on energy efficiency and fairly compare the energy efficiency of different databases.

1) YCSB

YCSB is one of the widely used comparative benchmark for relational and NoSQL databases. It supports a wide range of database bindings and provides a diverse set of desired workloads. YCSB consists of two parts: the YCSB Client (an extensible workload generator) and the core workloads (a set of workloads executed by the generator). YCSB includes six core workloads, as shown in Table 1.

Since workloads D and E insert records during the test run, we execute the workloads in following order to keep the database size consistent.

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, increasing the size of the database.

To fairly compare the performance and energy efficiency of various databases, we execute the aforementioned workloads in the same order for MySQL, MongoDB, and Cassandra under similar workload conditions.

2) Twitter Data

Although YCSB provides an effective way to compare various databases, datasets used by the YCSB benchmark are very simple, which typically consists a single table or one field document. This may not represent the use case in real world scenarios. Additionally, YCSB does not support the execution of complex queries that we want to evaluate. In order to analyze more complicated datasets with index landscape supporting complex queries, we collect a large volume of Twitter data (~100GB) using the Streaming API [9], which provides a continuous stream of updated Tweets automatically. These streams are extracted in the JSON format, which allows lightweight data interchange for MySQL, MongoDB, and Cassandra. To leverage the power measurement API of the Marcher system, we use the PyMongo API [10] to run queries on MongoDB and the PyCassa API [11] to run queries on Cassandra.

V. EXPERIMENTAL RESULTS OF NOSQL DATABASES

A. MongoDB

In this section, we present the experimental results of numerous optimizations running over the 100GB of Twitter data on MongoDB. There are many factors that can affect MongoDB performance, which include index usage, query structure, data models, and operational factors such as architecture and system configurations. We only focus on the following query optimizations in this study.

1) Covered Queries

In MongoDB, a covered query is defined as a query in which all required fields are part of an index and all fields returned in the query are also in the same index [12]. A covered query can be processed using the same index without scanning the documents. Since indexes are stored in DRAM and documents are stored in disks, it is much faster to fetch data from indexes than fetch data by scanning documents. To evaluate the impact of covered query optimization on performance, power and energy, we first created a compound index for the "user.location" field:

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

Then, we search for users' location using the un-optimized query (i.e. scan the documents on hard drives) and optimized query (i.e. covered query) respectively. Table 2 shows that the covered query runs 276 times faster and consumes 498 times less energy than the un-optimized query. This experiment clearly shows that the covered query optimization can not only improve both performance and energy efficiency but also help energy more than performance (ref. case 1 in Section III B). We also observe that this trend is more obvious when the input size is small enough to fit into the cache. Fig. 2. plots the real-time power trace when running an un-optimized query and a covered query, which demonstrates the power saving benefits when required data can be stored

in cache. The power consumption can be reduced by almost 50% in this example. When the data size gets larger and cannot fit in cache completely, the power saving benefits decrease but still can save power if part of the data are stored in cache).

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 should be optimized for faster lookup. Indexing is one of the typical optimizations for fast access of a collection [13]. The selection of the right indexes highly depends on the data to be queried. In this experiment, we delete the same record (see below), but one query has the field "user_mentions" indexed while the other does not.

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

The results shown in Table 3 and Fig. 3 prove that using indexes can significantly improve performance (Speedup of 283) and reduce energy consumption (Greenup of 474). Similar to the covered query optimization, we observe a case 1 example due to the performance and power benefits of storing and processing most recently used data in cache.

Table 4 and Fig. 4 show the experimental results of the impact of using indexes on insertion operations. Although it does not improve performance much (only by 4%), it significantly reduces the power consumption by 47%, which leads to approximately two times of total energy savings.

3) Ordered vs Unordered Queries

MongoDB allows clients to perform write operations in bulk. Bulk write operations can be either ordered or unordered. The ordered write operations are executed serially while the unordered write operations can be executed in parallel [14].

We initially expect that the ordered write operations will be slower than the unordered operations because each operation must wait for the previous operation to finish. Surprisingly, the experimental results shown in Table 5 and Fig. 5 demonstrate that unordered updates not only take longer to execute but also consume more energy overall. One possible reason is that MongoDB tightly controls how database operations are acknowledged by a server, which ranges from checking through acknowledgment that the operation has been executed to confirming the result has been written to the journal. This acknowledgment mechanism aims to ensure completeness of operations but brings overhead to performance. Since the ordered write operations are done serially, less acknowledgment and synchronization are probably required. Meanwhile, the advantage of parallel execution on unordered operations seems to be largely dwarfed by the acknowledgment overhead. There is no evidence to show that more cores are being utilized by unordered operations. In fact, the ordered operations are better in all aspects (faster, less power, and better energy efficiency) of this experiment.

4) Projection Optimization Using Aggregation

MongoDB provides a rich set of projection and aggregation operations that aim to reduce the unnecessary calculations on datasets [15]. Projection only selects the necessary data rather than the whole data fields of a document. Aggregation operations use collections of documents as an input and return results in the form of one or more documents, which simplifies application code and reduces resource usage.

The pipeline method [16] is the preferred method for data aggregation in MongoDB because it provides efficient data aggregation using native operations. Meanwhile, MongoDB provides Map-Reduce as an alternative method to perform aggregation [17], which include two phases. The map phase processes each document and the reduce phase combines the output of the map operations. Map-Reduce can specify a query condition to select the input documents as well as sort and limit the results. Since Map-Reduce is less efficient in general and more complex than the pipeline method, we evaluate the impact of the aggregation optimization using the pipeline method. Specifically, we use the following query to find the most tweeted user:

```
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}}}])
```

As shown in Table 6 and Fig. 6, aggregation proves to be energy efficient for complex queries. We observe a Speedup of 2.4 and a Greenup of 4.2 for aggregated queries compared to the un-optimized query. The aggregation pipeline can determine if it requires only a subset of the fields in the documents to obtain the results. Since the aggregation pipeline only uses necessary fields, it greatly increases the opportunities of reducing the amount of data passing through the pipeline. Consequently, it is more likely the required data can fit into cache, which greatly helps both performance and energy efficiency, as explained previously in optimizations for covered queries and indexes.

5) Sharding

Sharding is the process of distributing data across multiple servers. MongoDB uses sharding to deploy very large data sets for high throughput operations [18]. Sharding in MongoDB supports horizontal scaling, which involves dividing the entire dataset into

smaller portions and load them over multiple servers. Additional servers can be added to increase capacity if required. Although the speed or capacity of a single server may not be high, sharding distributes the workload to multiple servers, which can potentially achieve better performance than a single high-speed high-capacity server. Since expanding the size of the database only requires adding additional servers as needed (i.e. scale out), sharding is more cost effective than scaling up to a high-end server. However, sharding increases the power consumption as well as the complexity of data management.

To evaluate the impact of sharding on performance, power and energy, we split the Twitter datasets and store them on two Marcher servers. The results presented in Table 7 indicate that the performance of sharding with two servers is three times better than a single server but the power consumption is almost doubled. It is worth noting that the power in Table 7 only includes the CPU power and DRAM power. The Powerup may change if the idle power of other system components (e.g. Motherboard, GPU, Hard Drive, Fan, etc.) is calculated. Overall, the sharded servers help in improving performance and reducing the total energy consumption because the margin of performance improvement is larger than the power penalty, thanks to the increased cache and DRAM sizes of multiple servers. This experiment demonstrates a typical example of case 3 (ref. Section III B).

B. CASSANDRA

In this section, we evaluate the impact of two optimizations for Cassandra on performance, power and energy. We create keyspaces in Cassandra to store ~100GB Twitter data. To leverage the power measurement tool of the Marcher system, we use PyCassa [11], a thrift-based python library for Cassandra, to execute Cassandra queries and obtain real-time power consumption data.

Cassandra works optimally when the required data is already in memory or cache. If data has to be fetched from disks, it works better when the read operation is performed sequentially. The best practices for improving performance in Cassandra include (but not limited to) enabling row caching [19] and compaction [20]. The following two sub-sections provide in-depth analysis for these two optimizations.

1) Row Caching

With row caching enabled, Cassandra will detect frequently accessed partitions and store rows of data into DRAM or cache to reduce the data access latency. To study the impact of row caches on performance, power and energy, we analyze the update queries with row caching enabled (i.e. the optimized query) and disabled (i.e. the un-optimized query) respectively.

The results shown in Table 8 and Fig. 7 prove that row caching helps both performance and energy efficiency. We observe a Speedup of 150 and Greenup of 163 in case of optimized query taking advantage of row caching. The Greenup is larger than Speedup because the average power consumption is 8% lower when row caching is enabled. This shows again the power benefit of caching data because it is generally more power effective to process data in cache or DRAM than in disks.

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.

Compaction has impact on read performance in two ways. While a compaction is in progress, it can temporarily influence read performance (if the require data is missing in cache or DRAM) because the disks are heavily utilized during compaction. 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 provides different compaction optimization strategies for different scenarios. The 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 workloads, as depicted in Table 9. A Speedup of 2.6 and Greenup of 2.8 are achieved when the STCS strategy is used.

Another optimization strategy for compaction is the Leveled Compaction Strategy (LCS). This strategy groups SSTables into levels, each of which has a fixed size limit (10 times larger than the previous level). SSTables have a fixed and relatively small size (160MB by default). For example, if Level 1 contains up to ten SSTables, then Level 2 will contain no more than 100 SSTables. SSTables are guaranteed to be non-overlapping within each level – if any data overlaps in a table, it will be promoted to the next level and the overlapping tables will be re-compacted. This compaction strategy is best suited for read-heavy workloads because tables within the same level are non-overlapping. Table 10 shows the performance, power and energy results of running a read-heavy query, which is to find the most tweeted user, with LCS enabled and disabled respectively.

The results shown in Table 9 demonstrate that the LCS strategy can improve performance and energy efficiency by a factor of 4.5 and 5.2 compared with the non-optimized query. Meanwhile, Powerup is less than 1, which emphasizes the power saving benefits by leveraging cache size of the system and the chunk size of datasets to reduce cache miss rate.

VI. EXPERIMENTAL RESULTS OF RELATIONAL DATABASES

In this section, we present the experimental results of the relational database. We create a MySQL database and import the Twitter data into tables. In relational databases, programmers can obtain the same results by writing different SQL queries. However, the performance and energy efficiency of different queries that generate the same results could vary significantly. A number of studies have reported the performance and energy tradeoffs of relational databases [28, 29, 30, 31, 32]. In this study, we explore and report a set of SQL querying optimization techniques that can lead to better performance and energy efficiency.

A. *Indexing*

Full-table scans can result in excessive amounts of unnecessary I/O and degrade the performance a database system. The most common strategy for reducing unnecessary full-table scans is adding indexes [33]. The Primary Key for a table acts as a default index. Additional indexes can be added to a table depending on the size of data it holds. B-tree, bitmapped, and function-based indexes can be added to columns to speed up retrieval. To evaluate the impact of indexing for performance and energy efficiency, we execute the following SQL query, which counts the number of tweets from San Diego.

```
SELECT count(*) as tweets
FROM location_details
WHERE location = 'San Diego';
```

As depicted in Table 11 and Fig. 8, indexing resulted in a Speedup of roughly 66X and a Greenup of nearly 70X. The use of indexing results in both high performance as well as energy efficiency.

We conduct a similar experiment for delete operations. The results are depicted in Table 12 and Fig. 9.

For delete operations, Table 12 shows that indexing leads to a 13X performance increase and a 17X reduction in energy consumption. 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 does take time but is faster than scanning the entire table. This contributes to lower execution times and less power consumption.

We also investigate the performance and energy efficiency of more complex queries such as finding the most tweeted user:

```
SELECT username, count(*)
From tweet_details
GROUP BY username
ORDER BY count DESC LIMIT 1;
```

As shown in Fig. 10 and Table 13, we observe that an indexed query has a Speedup of 33X and a Greenup of about 40X compared to a non-indexed query. A strong conclusion can be drawn from this set of experiments, which is indexing is an effective optimization that not only accelerates the execution but also reduces the power consumption of a SQL query. In other words, it belongs to category 1 optimizations, which we have observed previously in the NoSQL optimizations where better utilization of cache plays a critical role.

B. *Avoid using Select * Clauses*

The wild card character '*' allows the reference to all columns of a table in SQL. While this feature can be convenient, it is extremely inefficient. The '*' character has to be converted to each column by obtaining the names of valid columns from the data dictionary and substituting them on the command line. We quantitatively evaluate the energy cost of using the '*'character with the following two queries:

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

Query without using '*':

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

Table 14 shows that the query without using *SELECT* * has a Speedup of 1.07X and a Greenup of 1.22X. Fig. 11 clearly depicts that the performance gains of this particular optimization is not significant (<7%), but the energy savings, especially when

considering the ubiquity of the *SELECT* * in SQL, is significant (>20%). This is a great example of cases where energy efficiency is not directly tied to performance improvement. A more than 20% of savings in energy for database transactions by simply typing out column names, or using tools to auto-populate them, is low hanging fruit for any organization with a data intensive infrastructure.

C. IN vs. EXISTS

In SQL, the *EXISTS* function searches for a single row that meets the stated criteria while the IN statement looks for all occurrences. We compare the performance and energy efficiency of IN versus EXISTS using the two queries below:

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

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

Table 15 shows that *EXISTS* is much more efficient. Compared to using IN, the use of *EXISTS* achieves a speeded up of 146X and a green up of 208X. This is another technique that leads to category 1 optimization, which can be explained by how the *EXISTS* clause works. When *EXISTS* is executed it does a partial scan of the table and stops after it finds the first matching row. However, *IN* scans every row in the table to determine if they match the criteria. Even when searching for values in columns that represent a minority of data for that field, it still remains more likely to find the value much earlier in the table than searching the entire table. Meanwhile, it also increases the chances where the search operations can process data in the cache rather than loading data from the memory.

VII. Cross Database Comparison

In this section, we perform a series of performance and energy analysis on Cassandra, MongoDB, and MySQL using both the YCSB benchmark and the Twitter data. 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 highly possible for one database to outperform another and yet lag its competitor when the rules of engagement change. The benchmarks and Twitter data used in this study may not represent the best-case scenarios of each evaluated database. In addition, we evaluate the impact of various Dynamic Voltage and Frequency Scaling (DVFS) policies on the energy consumption and performance of all three databases.

A. Database Comparison using YCSB

For fair comparison, each test starts with an empty database, which is loaded with an initial set of randomly generated data. Once the data is loaded, we run the six workloads in exactly the same order (ref. Section III C). In between each workload, occasional database health and readiness checks are performed. For example, we check if there are any ongoing compaction processes in Cassandra and wait until those are completed before continuing to the next workload. The measured performance and energy results are presented in Table 16.

In all workloads used in the YCSB, both NoSQL databases outperform MySQL, with MonogoDB as the winner. Although MySQL does consume less power than Cassandra, Cassandra still has better energy efficiency because of its significantly shorter execution time. It is notable that there is very little difference on what type of workload runs, as energy and execution time are fairly consistent for each database. Since we observe a similar pattern on different workloads, here we only report the results of workloads A, B, C, D, and E and only include the power trace for running workload A (See Fig. 12) on three databases. As it is unlikely that three database systems with very different design would perform so uniformly across vastly different tasks in the real world, the results reinforce our impression that other datasets are needed to reveal the true advantages and disadvantages of each database.

B. Database Comparison using Twitter Data

In this section, we conduct a cross database comparison using the Twitter data. For fair comparison, a number of commonly used queries (search, update, delete, insert) are used in our experiments. These queries have different syntax in different databases but they essentially query the same size of datasets and return the same results.

The first query finds the most tweeted user and the results of the query are presented in Table 17 and Fig. 13. In this experiment, MySQL has the best performance (4.08 s) and consumes the least energy consumption (254.60 J). It has a Speedup and Greenup of 45X and 89X over Cassandra. MongoDB is less efficient than MySQL as aggregation is used to find the most tweeted users, which is a slower operation than the groupby and order by functions used in MySQL. Cassandra, while utilizing extensive parallelism to execute the query, has more overhead than MongoDB as it has insufficient support for aggregation functions. This makes Cassandra particularly not suitable for this kind of query.

The second query we investigate is *update.* The results are shown in Table 18 and Fig. 14. In this experiment, we observe that MongoDB and Cassandra both are more energy efficient than MySQL even though their power consumption is higher. In this case, the energy savings come purely from performance improvement. The Greenup of MongoDB over MySQL falls under category 3, which is often the result of parallelism used to speed up a task.

The third query we evaluate is *delete.* The results are shown in Table 19 and Fig.15. We observe that both MongoDB and Cassandra databases have higher power consumption, likely because of their highly parallel nature. MongoDB is the most efficient database in case of delete query execution and Cassandra is the least efficient one due to its high power and low performance. MongoDB benefits from its shorter execution time and the power consumption is moderate, thereby making it highly energy efficient.

The fourth query we evaluate is *insert.* The results are shown in Table 20 and Fig 16. In this experiment, MySQL and MongoDB have nearly identical execution time and power consumption, both of which execute faster and use about a third of the energy of Cassandra.

The last query we evaluate is *search.* The results are shown in Table 21. In this experiment, MongoDB excels in both performance and energy efficiency. MySQL consumes an order of magnitude more energy than MongoDB while Cassandra is even worse by consuming two orders of magnitude more energy.

The evaluation results on different queries show that the performance and energy efficiency of various databases are drastically different when executing diverse workloads. There is no single database that is uniquely better in terms of performance and energy consumption than all other databases. We hope that the variety of queries we have demonstrated will be useful for people deciding which database is the most appropriate one for their workload. It is notable that in our experiments Cassandra never has the best performance nor energy efficiency. This is possibly due to the relatively small scale of our testing environment, in which the parallel design of Cassandra could not be fully utilized. It is clear that at small scale (one or two servers) the cost of Cassandra's design outweighs its benefits. Additional experiments with more servers and larger datasets will be necessary to further investigate the advantages of Cassandra.

C. *Dynamic Voltage and Frequency Scaling (DVFS)*

Dynamic Voltage and Frequency Scaling 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 DVFS governors supported by the Linux kernel are:

1. Performance: This governor sets the CPU statically to the highest frequency with 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_req.
3. Ondemand: Ondemand governor sets the CPU depending on the current CPU usage. To do this the CPU must have the capability to switch the frequency very quickly.

To evaluate the impact of DVFS on performance and energy efficiency, we conduct experiments on various databases executing queries using both the "Performance" and "ondemand" governors.

1) Effects on MySQL queries

Fig. 17 shows that MySQL's *insert* query uses less than half of the power with only a slightly performance degradation when using the "ondemand" governor, which yields much better energy efficiency.

Fig. 18 shows the execution time and power usage when running the *update* query under the "performance" mode and "ondemand" mode. This experiment shows counter-intuitive results. The "performance" mode, which presumably accelerates instruction execution, slows down the update operation and increases the power usage. It has been reported that higher processing speeds can cause performance degradation in high performance computing applications [32]. The slowdowns occur at higher frequencies when the early arrival of a single thread causes the atomic journal to 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. Since the update query is write heavy and needs synchronization, we predict that the synchronization overhead not only dwarfs but also hurts performance and energy efficiency when the CPU is running in the "performance" mode.

2) Effects on MongoDB queries

For MongoDB, we test the "find the most tweeted user using aggregation" query and the *search* query. The experimental results are presented in Fig. 19 and Fig. 20 respectively. For both queries, we observe that the "ondemand" governor is able to reduce power consumption with no negative effect on performance.

3) Effects on Cassandra queries

To study the effects of DVFS on Cassandra, we conduct two experiments using the *insert* and *delete* queries. The results are depicted in Fig. 21 and Fig. 22 respectively. For the *insert* query, we observe the same counter-intuitive results (see Fig. 21), in which the "performance" governor makes the query run slower with higher power (similar to the MySQL's *update* query). On the other hand, the "ondemand" governer reduces both power consumption and execution time. For the *delete* query, we observe sharp power spikes from both the "performance" and "ondemand" governors (see Fig. 22). As expected, the power consumption of the "ondemand" governor is less than the "performance" governor, which make it more energy efficient.

VIII. CONCLUSION AND FUTURE WORK

This paper evaluates the impact of various optimization techniques for both relational and NoSQL databases on performance, power, and energy consumption. We strive to provide additional evidence to complement the conventional wisdom that optimizations for performance will either hurt energy efficiency or help energy efficiency equally. We also conduct, as far as we can tell, first of its kind comparisons of performance and energy use characteristics of both relational and NoSQL databases.

We use the YCSB benchmark and ~100GB of Twitter data as the testing workloads and implement a power measurement tool on the NSF funded Marcher system to measure the real-time power consumption of queries with and without optimizations. Our experimental results show that (1) energy efficiency can be improved significantly for all studied database management systems without degrading performance; (2) energy efficiency is not always linear to performance improvement; and (3) optimizations do not always help and sometimes may degrade performance and increase power consumption.

For MongoDB, we find that using indexes can achieve a Speedup of more than 280 times and reduce power consumption by 40% at the same time for *delete* operations, which altogether improves energy efficiency by 474 times. Similarly, covered query can improve performance by 276 times while consuming 45% less power, which leads to a Greenup of almost 500 times. Additionally, we find that ordered queries have better performance and energy efficiency than unordered queries, which is counter-intuitive because ordered queries are executed sequentially therefore are expected to be slower. Aggregated queries help both performance and energy efficiency by a factor of 2.4 and 4.2 respectively. Sharding on multiple servers can improve performance at the cost of higher power consumption. Total energy savings can still be achieved provided that the Speedup is larger than the power increase rate. It is worth noting that our experiments contain only two sharding servers. Performance improvement and energy savings could become conflicting goals if Speedup cannot scale up linearly with the number of sharding servers.

For Cassandra, we find that the row caching optimization helps both performance and energy efficiency with a Speedup of 150 and Greenup of 163. Two optimizations (STCS and LCS) for compaction are both able to improve performance and energy efficiency if they are used appropriately (e.g. STCS for insert-heavy query and LCS for read-heavy query).

For MySQL, we find that *indexing* significantly improves performance and energy efficiency with *indexed search* having a Speedup of 66 and a Greenup of 69, and *indexed delete* having a Speedup of 13 and a Greenup of 17. Avoiding the use of the 'SELECT *' command results in more than 20% gain in energy efficiency, which is significant because of the ubiquity of the command among practitioners. Additionally, it is easy for such a benefit to remain unnoticed, as it is not connected to significant performance improvement.

In our comparison of databases using the YCSB, MongoDB performs the best and consumes less energy for most of the workloads. However, when evaluated using the Twitter data and our own "real world" type queries, the relative performance and energy efficiency of different databases vary greatly. We highly recommend configuring servers running database systems to the "ondemand" DVFS mode because it resulted in lower power consumption with little or no performance degradation in almost all experiments. This is true for all three databases and in some cases the "ondemand" governor even yields better performance.

In the future, we will expand our experiments to more databases with bigger and more complex data sets. The Cassandra database does not perform as well as MongoDB in our experiments. It is possible that the highly parallel and scalable nature of Cassandra is not well suited to our experimental setup with the maximum of two servers. Further experimentation with more servers and even larger datasets could shed light on the advantages of Cassandra. In the future, we would like to extend our experiments to a cluster of nodes to analyze performance and energy efficiency tradeoffs.

ACKNOWLEDGMENT

The work reported in this paper is supported by the U.S. National Science Foundation under Grant No. CNS-1305359.

REFERENCES

[1] http://www.internetlivestats.com/google-search-statistics/.
[2] A. Beckmann, U. Meyer, P. Sanders, and J. Singler. "Energy-efficient sorting using solid state disks", Proceedings of the Green Computing Conference 2010, pages 191–202, 2010.
[3] 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", http://arxiv.org/abs/1303.4869, 2013.
[4] https://01.org/blogs/2014/running-average-power-limit---rapl.
[5] https://developer.nvidia.com/nvidia-management-library-nvml.
[6] https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-developers-quick-start-guide.
[7] Z. L. Zong, R. Ge, and Q. J. Gu. "Marcher: A Heterogeneous System Supporting Energy-Aware High Performance Computing and Big Data Analytics", Journal of Big Data Research, 2017.
[8] https://github.com/brianfrankcooper/YCSB/wiki.
[9] https://dev.twitter.com/streaming/overview.
[10] http://api.mongodb.com/python/current/api/.
[11] http://pycassa.github.io/pycassa/api/.
[12]https://docs.mongodb.com/v3.2/indexes/#covered-queries.
[13] https://docs.mongodb.com/v3.2/indexes/.
[14] https://docs.mongodb.com/v3.2/core/bulk-write-operations/.
[15] https://docs.mongodb.com/v3.2/aggregation/.
[16] https://docs.mongodb.com/v3.2/aggregation/#aggregation-pipeline.
[17] https://docs.mongodb.com/v3.2/aggregation/#map-reduce.
[18] https://docs.mongodb.com/v3.2/sharding/.
[19] https://www.datastax.com/dev/blog/row-caching-in-cassandra-2-1.
[20]http://docs.datastax.com/en/archived/cassandra/2.0/cassandra/operations/ops_configure_compaction_t.html.
[21] D. Schall, V. Hudlet, and T. Harder. "Enhancing Energy Efficiency of Database Applications Using SSDs", Proceedings of the Third Conference on Computer Science and Software Engineering, 2010.
[22] G. Graefe. "Database servers tailored to improve energy efficiency", Proceedings of the EDBT workshop on Software engineering for tailor-made data management, 2008.
[23] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. "Energy Efficiency: The New Holy Grail of Data Management Systems Research", Proceedings of CIDR'09, 2009.
[24] D. Tsirogiannis, S. Harizopoulos and M. Shah. "Analyzing the energy efficiency of a database server", Proceedings of SIGMOD'10, 2010, pp.231-232.
[25] Z. Xu, Y. Tu, and X. Wang. "Exploring power performance tradeoffs in database systems", Proceedings of ICDE, 2010.
[26] W. Lang and J. M. Patel. "Towards Eco-friendly Database Management Systems", Proceedings of CIDR'09, 2009.
[27] T.T. Li, G. Yu, X. B. Liu, J. Song. "Analyzing the Waiting Energy Consumption of NoSQL Databases", Proceedings of the IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, 2014.
[28] 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.
[29] W. Lang, R. Kandhan, and J. M. Patel. "Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race", IEEE Computer Society Technical Committee on Data Engineering, 2011.
[30] 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.
[31] D. Florescu and D. Kossmann. "Rethinking Cost and Performance of Database Systems", ACM SIGMOD Record, vol. 38(1), pp. 43-48, Mar. 2009.
[32] H. C. Chang, B. Li, M. Grove, and K.W. Cameron. "How Processor Speedups Can Slow Down I/O Performance", Proceedings of the IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, 2014.
[33] https://docs.MongoDB.com/manual/indexes/

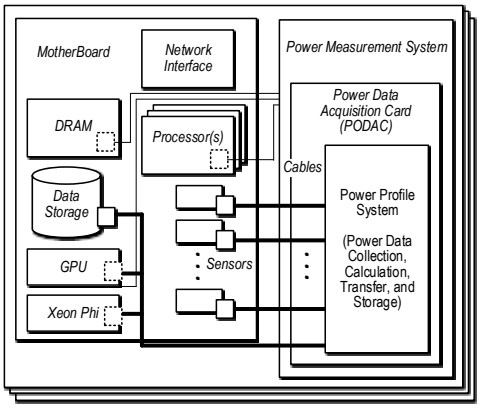

Fig. 1. Power measurement of Marcher servers

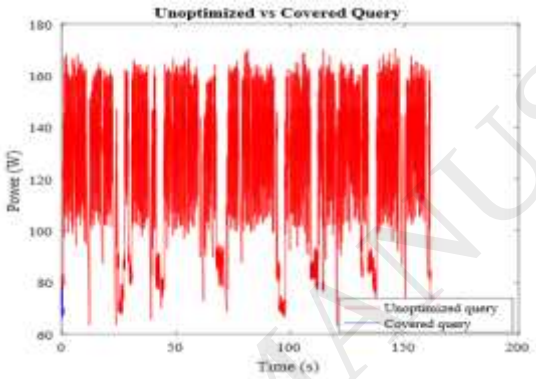

Fig. 2. Power: un-optimized query vs covered query

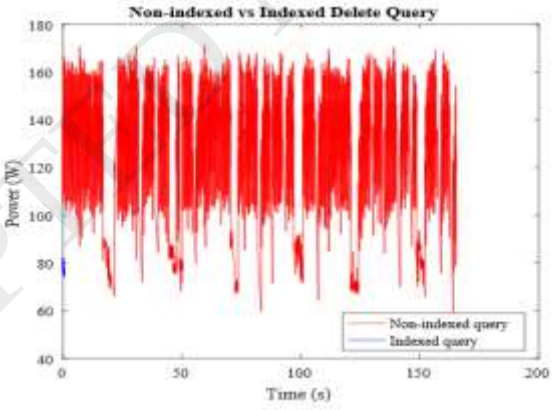

Fig. 3. Power: non-indexed vs. indexed query for deletion

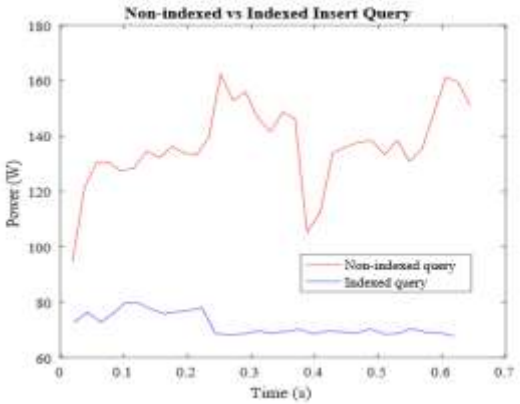

Fig. 4. Power: non-indexed query vs. indexed query for insertion

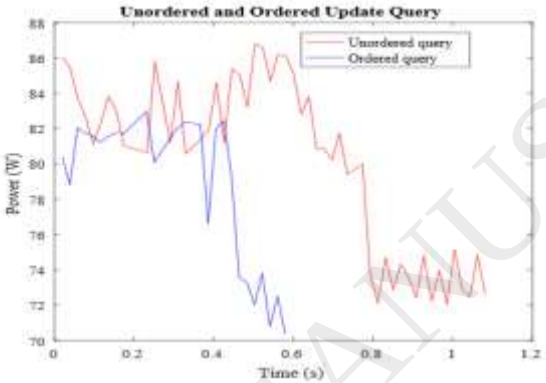

Fig. 5. Power: ordered query vs. unordered query

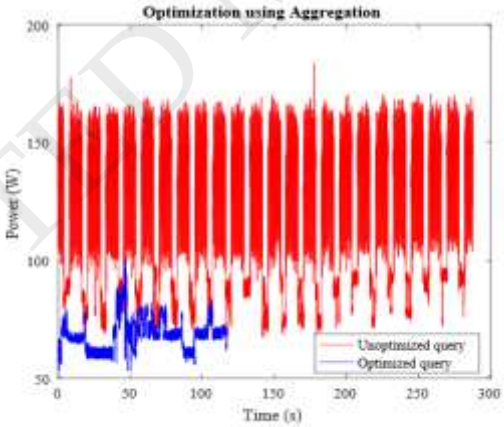

Fig. 6. Power: un-optimized query vs. aggregated query

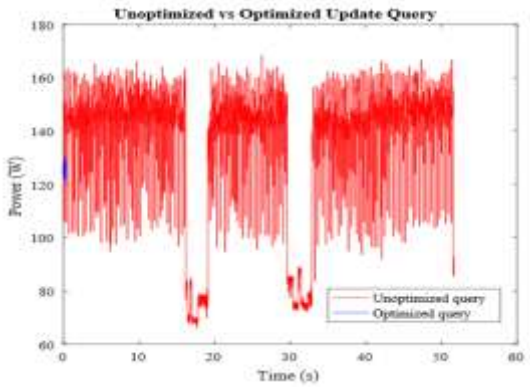

Fig. 7. Power: un-optimized query vs. optimized query

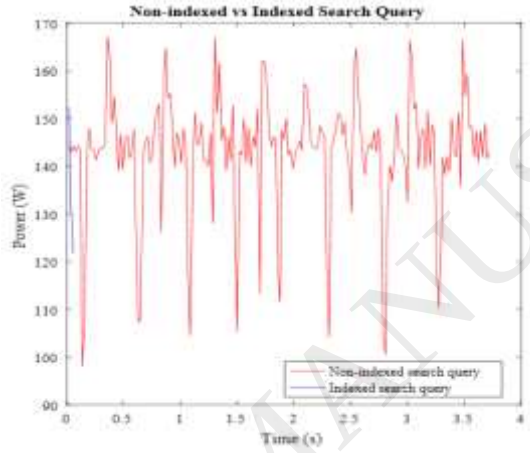

Fig. 8. Power: Non-indexed vs Indexed search

sfsdf

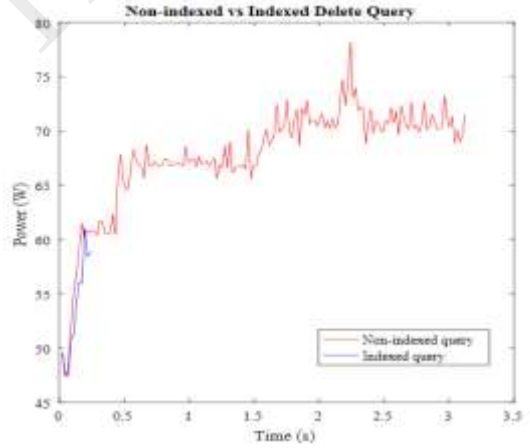

Fig. 9. Non-indexed vs Indexed delete

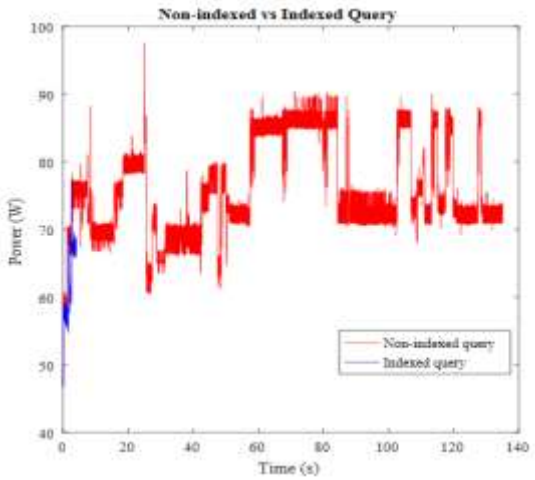

Fig. 10. Non-indexed vs indexed Query to find most tweeted user

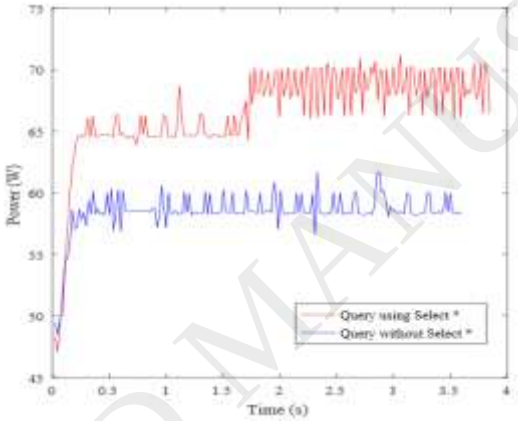

Fig. 11. Query execution using select * clauses

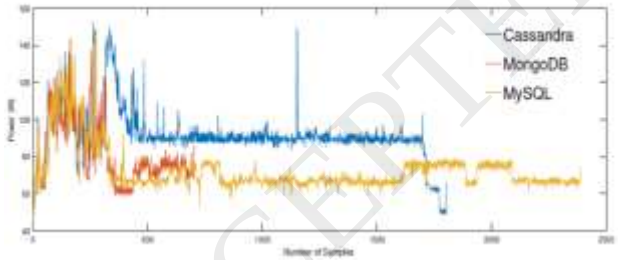

Fig. 12. Cross-database comparison using YCSB Workload A

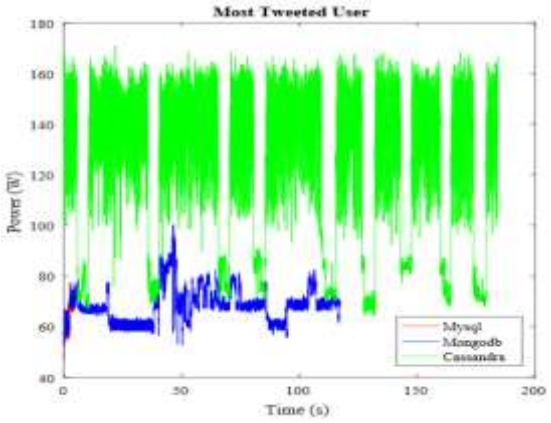

Fig. 13. Cross-database comparison to find the most tweeted user

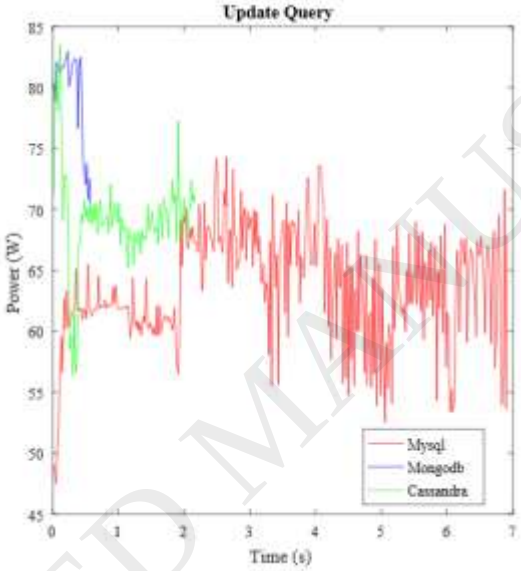

Fig. 14. Cross-database comparison for update

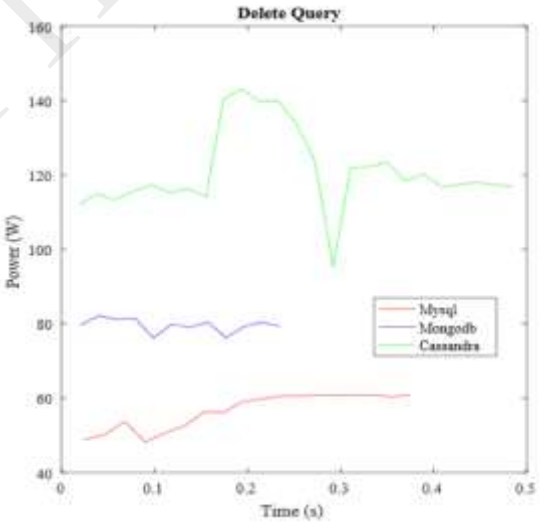

Fig. 15. Cross-database comparison for delete

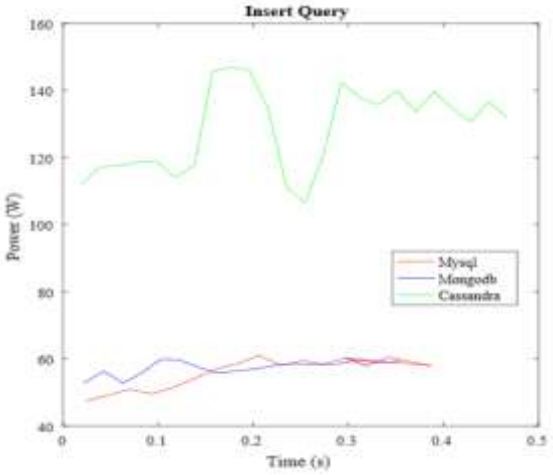

Fig. 16. Cross-database comparison for insert

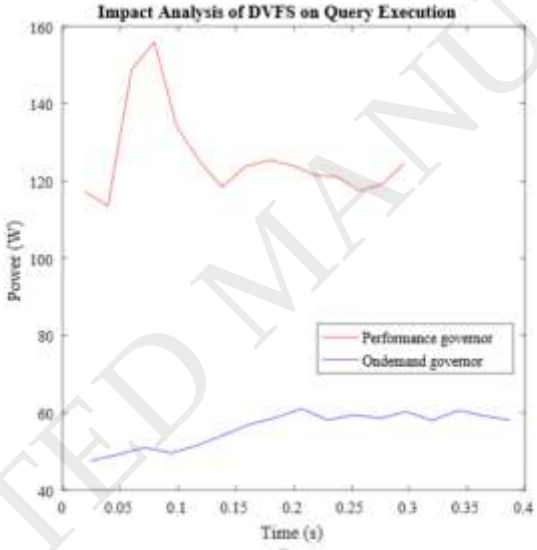

Fig. 17. The impact of DVFS on MySQL – insert query

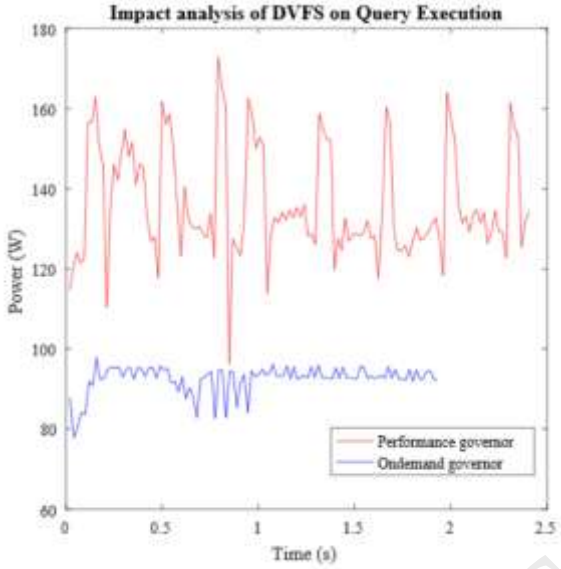

Fig. 18. The impact of DVFS on MySQL – update query

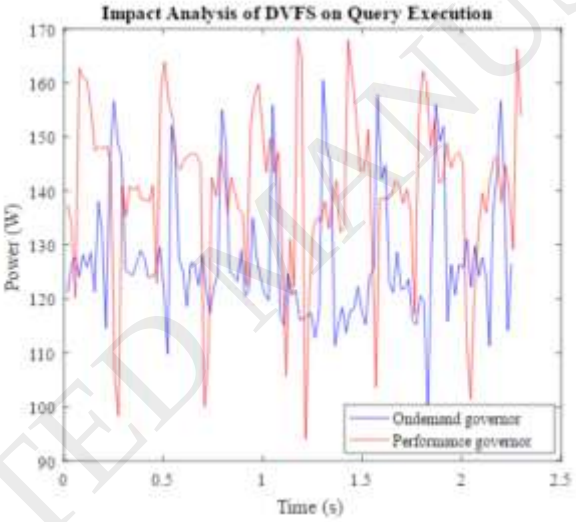

Fig. 19. The impact of DVFS on MongoDB - find most tweeted user

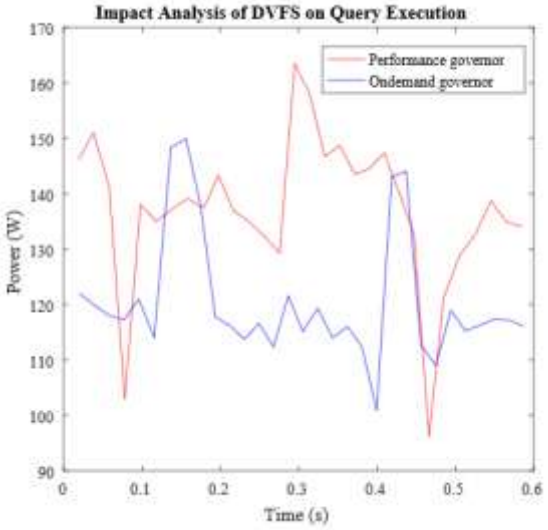

Fig. 20. The impact of DVFS on MongoDB – search query

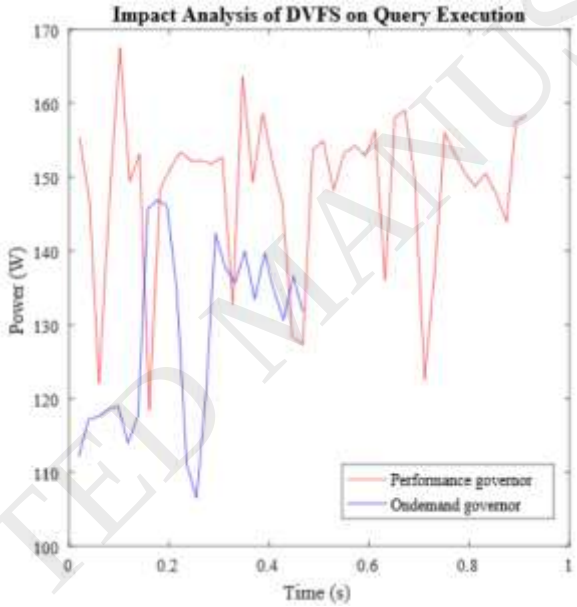

Fig. 21. The impact of DVFS on Cassandra – insert query

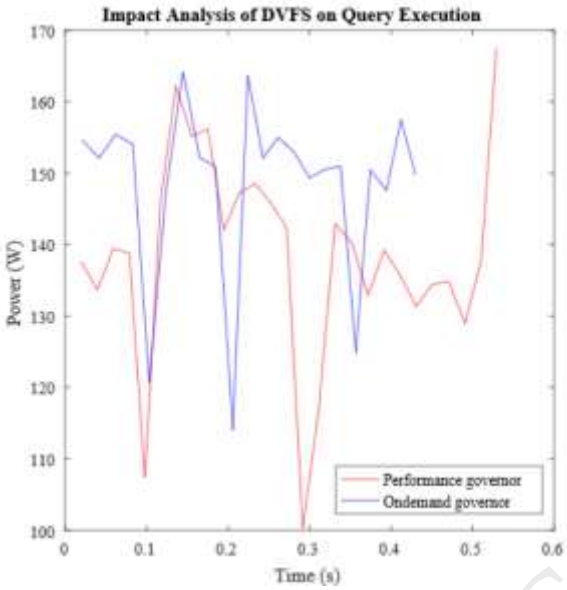

Fig. 22. The impact of DVFS on Cassandra – delete query

Table 1: YCSB Workloads

YCSB Workloads		
Workload	Type	Operation
A	Update Heavy	Read: 50%, Write: 50%
B	Read Heavy	Read: 95%, Write: 5%
C	Read Only	Read: 100%
D	Read Latest	New records are inserted Heavy read for new records
E	Short Ranges	Query short ranges of records
F	Read-modify-write	Read a record, modify it, and write back the changes

Table 2: Unoptimized vs Covered Query

Query	Power(W)	Time(s)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	126.67	162.79	20619.32	276.15	0.55	498.35
Covered	70.19	0.59	41.38			

Table 3: Non-indexed vs. Indexed Delete Query

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Non-indexed	130.52	165.25	21568.29	283.25	0.60	474.01
Indexed	77.99	0.58	45.50			

Table 4: Non-indexed vs. Indexed Insert Query

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Non-indexed	136.88	0.64	88.19	1.04	0.53	1.98

Indexed	71.90	0.62	44.53			

Table 5: Ordered vs. Unordered Query

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Unordered	80.79	1.08	87.42	1.87	0.98	1.91
Ordered	78.80	0.58	45.68			

Table 6: Unoptimized Query vs. Aggregated Query

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	124.36	287.64	35769.76	2.45	0.57	4.27
Optimized	71.50	117.17	8377.28			

Table 7: Single Server vs. Sharded Servers

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Single Server	77.59	281.48	21839.62	3.08	1.81	1.71
Sharded Server	140.06	91.31	12788.32			

Table 8: Row Caching Enabled vs. Disabled

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Un-optimized	135.80	51.62	7010.14	150.85	0.92	163.12
Optimized	125.58	0.34	42.97			

Table 9: STCS Enabled vs. Disabled

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
W/O STCS	137.75	1.23	169.35	2.63	0.93	2.81
STCS	128.77	0.47	60.18			

Table 10: LCS Enabled vs. Disabled

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
W/O STCS	142.30	831.85	118371.9	4.52	0.87	5.17
STCS	124.30	184.24	22901.45			

Table 11: Non-indexed vs. Indexed Search Query

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Non-indexed	142.88	3.72	531.68	66.34	0.95	69.89
Indexed	135.62	0.06	7.61			

Table 12: Non-indexed vs. Indexed Delete Query

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Non-indexed	67.89	3.12	211.87	13.47	0.78	17.29
Indexed	52.88	0.23	12.25			

Table 13: Non-indexed vs. Indexed Query to Find Most Tweeted User

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup

Non-indexed	75.87	135.04	10246.12	33.08	0.82	40.24
Indexed	62.36	4.08	254.60			

Table 14: Select * Clause

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
With Select *	66.66	3.85	256.44	1.07	0.88	1.22
Without Select *	58.44	3.60	210.44			

Table 15: IN vs. EXIST

Query	Power(W)	Time(S)	Energy(J)	Speedup	Powerup	Greenup
Using 'IN'	68.94	13.59	937.02	146.34	0.70	208.75
Using 'EXISTS'	48.33	0.09	4.49			

Table 16: Cross-database Comparison using YCSB

Workloads	MongoDB		Cassandra		MySQL	
	Energy (J)	Time (S)	Energy (J)	Time (S)	Energy (J)	Time (S)
A	1211.37	12.94	3102.37	33.38	4037.72	46.75
B	1212.06	12.67	3085.76	32.99	4075.12	47.25
C	1219.75	12.87	3152.97	33.89	4097.56	47.14
D	1242.89	12.95	3070.00	32.97	4077.80	47.62
E	1224.69	12.85	3176.56	34.14	4169.83	48.28

Table 17: Cross-database Comparison for Most Tweeted User

Database	Power (W)	Time (S)	Energy (J)
MySQL	62.36	4.08	254.60
MongoDB	76.62	117.12	775.93
Cassandra	124.30	184.24	22901.44

Table 18: Cross-database Comparison for Update

Database	Power (W)	Time (S)	Energy (J)
MySQL	63.64	6.91	440.03
MongoDB	78.80	0.58	45.67
Cassandra	69.29	2.17	150.37

Table 19: Cross-database Comparison for Delete

Database	Power (W)	Time (S)	Energy (J)
MySQL	56.72	0.37	21.26
MongoDB	79.58	0.23	18.68
Cassandra	121.12	0.48	58.64

Table 20: Cross-database Comparison for Insert

Database	Power (W)	Time (S)	Energy (J)
MySQL	56.31	0.32	18.00
MongoDB	57.36	0.35	20.32
Cassandra	128.77	0.47	60.18

Table 21: Cross-database Comparison for Search

Database	Power (W)	Time (S)	Energy (J)
MySQL	130.00	2.25	292.09
MongoDB	70.19	0.59	41.37
Cassandra	122.55	37.77	4628.36