Titre: Title:	Understanding the Impact of Databases on the Energy Efficiency of Cloud Applications
Auteur: Author:	Bechir Bani
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Bani, B. (2016). Understanding the Impact of Databases on the Energy Efficiency of Cloud Applications [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2256/

UNIVERSITÉ DE MONTRÉAL

UNDERSTANDING THE IMPACT OF DATABASES ON THE ENERGY EFFICIENCY OF CLOUD APPLICATIONS

BECHIR BANI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AOÛT 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

UNDERSTANDING THE IMPACT OF DATABASES ON THE ENERGY EFFICIENCY OF CLOUD APPLICATIONS

présenté par: BANI Bechir
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. MULLINS John, Ph. D., président
M. GUÉHÉNEUC Yann-Gaël, Doctorat, membre et directeur de recherche
M. KHOMH Foutse, Ph. D., membre et codirecteur de recherche
M. QUINTERO Alejandro, Doctorat, membre

DEDICATION

To my family.

ACKNOWLEDGEMENTS

Before any development on this research experience, it is appropriate to begin this thesis with thanks as much to those who gave me their trust and allowed to grow and learn, and those who have contributed to good working atmosphere.

I would like to take this opportunity to express my sincere gratitude to my supervisors Prof. Yann-Gaël Guéhéneuc for his faith in me, his extensive support, and giving me confidence from the first day that it has constantly renewed. And Prof. Foutse Khomh for his great guidance and his deep interest in the search for knowledge, which enabled me to go further.

In addition, I would like to convey my gratitude to my committee members, Prof. John Mullins, Prof. Yann-Gaël Guéhéneuc, Prof. Foutse Khomh and Prof. Alejandro Quintero for their valuable feedback on this thesis.

I would like to thank all my colleagues who work in PTIDEJ, SWAT, MCIS and SOCCER labs for their kindness, generosity and favour during this master.

My thoughts go to all those who have been a real help for me especially my dear parents, brothers and sister for their help, their support, their encouragement and availability.

Finally, i cannot forget the support of my close friends especially Houssem for his great help and his valuable advice throughout this master.

RÉSUMÉ

Aujourd'hui, les applications infonuagiques sont utilisées dans toutes les industries ; de la finance, au commerce de détail, en passant par l'éducation, la communication, la manufacture, les services publics et les transports. Malgré leur popularité et leur large adoption, peu d'informations sont disponibles sur l'empreinte énergétique de ces applications et, en particulier, celle de leurs bases de données, qui constituent l'épine dorsale de ces applications infonuagiques. Pourtant, la réduction de la consommation d'énergie des applications est un objectif majeur pour la société et continuera de l'être à l'avenir.

Deux familles de bases de données sont actuellement utilisées dans les applications infonuagiques: Les bases de données relationnelles et non-relationnelles. Aussi, nous examinons la consommation d'énergie des trois bases de données utilisées par les applications infonuagiques : MySQL, PostgreSQL et MongoDB, respectivement relationelle, relationelle, et non-relationelle. Nous réalisons une série d'expériences avec trois applications infonuagiques (une application "multi-thread RESTful", "DVD Store", et "JPetStore").

Nous étudions également l'impact des patrons infonuagiques sur la consommation d'énergie parce que les bases de données dans les applications infonuagiques sont souvent implémentées conjointement avec des patrons infonuagiques tels que le "Local Database Proxy", le "Local Sharding Based Router", ou la "Priority Message Queue".

Nous mesurons la consommation d'énergie en utilisant l'outil Power-API pour garder une trace de l'énergie consommée au niveau de processus par les variantes des applications infonuagiques. Cette estimation énergétique au niveau processus donne une précision plus exacte que d'une estimation au niveau d'un logiciel en général. En plus de cela, nous mesurons le temps de réponse de l'application infonuagique pour mettre en contraste le temps de réponse avec l'efficacité énergétique, afin que les développeurs soient conscients des compromis entre ces deux indicateurs de qualité lors de la sélection d'une base de données pour leur application.

Nous rapportons que le choix des bases de données peut réduire la consommation d'énergie d'une application infonuagique quelque soit les trois types des patrons infonuagiques étudiés. Nous avons montré que la base de données MySQL est la moins consommatrice d'énergie, mais est la plus lente parmi les trois bases de données étudiées. PostgreSQL est la plus consommatrice d'énergie entre les trois bases de données, mais est plus rapide que MySQL, mais plus lente que MongoDB. MongoDB consomme plus d'énergie que MySQL, mais moins que PostgreSQL et est la plus rapide parmi les trois bases de données étudiées.

ABSTRACT

Cloud-based applications are used in about every industry; from financial, retail, education, and communication, to manufacturing, utilities, and transportation. Despite their popularity and wide adoption, little is still known about the energy footprint of these applications and, in particular, of their databases, which are the backbone of cloud-based applications. Reducing the energy consumption of applications is a major objective for society and will continue to be so in the near to far future.

Two families of databases are currently used in cloud-based applications: relational and non-relational databases. Consequently, in this thesis, we study the energy consumption of three databases used by cloud-based applications: MySQL, PostgreSQL, and MongoDB, which are respectively relational, relational, and non-relational. We devise a series of experiments with three cloud-based applications (a RESTful multi-threaded application, DVD Store, and JPetStore).

We also study the impact of cloud patterns on the energy consumption because databases in cloud-based applications are often implemented in conjunction with patterns like Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue.

We measure the energy consumption using the Power-API tool to keep track of the energy consumed at the process-level by the variants of the cloud-based applications. We measure the response time of the cloud-based application because we wanted to contrast response time with energy efficiency, so that developers are aware of the trade-offs between these two quality indicators when selecting a database for their application.

We report that the choice of the databases can reduce the energy consumption of a cloud-based application regardless of the three cloud patterns that are implemented. We showed that MySQL database is the least energy consuming but is the slowest among the three databases. PostgreSQL is the most energy consuming among the three databases, but is faster than MySQL but slower than MongoDB. MongoDB consumes more energy than MySQL but less than PostgreSQL and is the fastest among the three databases.

CO-AUTHORSHIP

Earlier study in the thesis is published as follows:

- A Study of the Energy Consumption of Databases and Cloud Patterns
  Béchir Bani, Foutse Khomh and Yann-Gaël Guéhéneuc, in *Proceedings of the 14th International Conference On Service Oriented Computing (ICSOC), Banff, Alberta, Canada, 10-13 October, 2016.*

  My contribution: Methodology, analysis and paper writing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

With the continuous development of the Internet and cloud computing, companies use databases to store and perform analyses on large data-sets in cloud environments. Cloud-based applications are used in about every industry today; from financial, retail, education, and communications, to manufacturing, utilities, and transportation. Yet, despite their popularity and wide adoption, little is known about the energy footprint of these applications and, in particular, of their databases. Therefore, we set to devise and carry out experiments to assess the energy footprint and the response time of the databases in cloud-based applications. Two families of databases are currently used in cloud-based applications: relational and non-relational databases. Companies demand high performance databases when reading and writing data [1].

In addition, they want to benefit from best practices encoded in the form of cloud patterns [2]. Cloud patterns are general and tough "good" solutions to recurring design problems for cloud-based applications. Design Patterns were introduced by Beck and Cunningham [3] and applied to object-oriented systems by Gamma et al. [4]. From that moment, design patterns have been expanded to all disciplines of software engineering, including cloud computing. In addition, cloud patterns [2] were processed to include the requirements of the cloud infrastructure, and they were borrowed from parallel computing [5]. We should mention that, in most cases, the "Priority Message Queue pattern" is used to manage inter-process communication [1] between components. However, in a cloud environment, the Message Queue pattern is used to improve scalability and availability [6]. Yet, reducing the energy consumption of applications is a major objective for society and will continue to be so in the near to far future.

1.1 Research statement

Some previous works have benchmarked databases with cloud-based workloads [7]. However, to the best of our knowledge, none of these works investigated the combined impact of databases and cloud patterns on the energy consumption of cloud-based applications. Consequently, the benefits and trade-offs of different databases and combinations of cloud patterns are mostly intuitive and not validated.

In this thesis, we evaluate the impact on energy consumption and response time of three

[1] https://en.wikipedia.org/wiki/Inter-process_communication

cloud patterns: Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue, with three databases: two relational databases, Postgresql and MySQL, and one NoSQL database, MongoDB. To achieve this evaluation, we use three versions of three cloud-based applications (a RESTful multi-threaded application, DVD Store, and JPetStore) that use respectively MySQL, Postgresql, and MongoDB databases. We also implement the three studied patterns in each version of the RESTful multi-threaded application.

We measure energy consumption using the Power-API profiler [8], which provides an application programming interface to estimate the energy consumed by an application at the process-level.

1.2 Research objectives

Our specific research objectives are as follows:

1. Propose an approach to collect energy measures of cloud-based applications implemented with cloud patterns in conjunction with databases in a cloud environment.
2. Evaluate the impact on energy consumption of three cloud patterns: Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue, individually, and also their combination, with three databases: MySQL, PostgreSQL, and MongoDB.
3. Highlight the contrast response time with energy efficiency of databases so that developers are aware of the trade-offs between these two quality indicators when selecting a database for their application.

1.3 Thesis outline

The rest of this thesis is organised as follows:

- Chapter 2 outlines background in the areas of relational and NoSQL databases, cloud patterns, and the Power-API profiler.
- Chapter 3 provide a literature review in the areas of software energy consumption and performance.
- Chapter 4 describes our methodology to study the impact of databases and cloud patterns on energy consumption.

- Chapter 5 presents the results of applying our methodology and discusses the impact of databases and cloud patterns on the energy consumption and performance of cloud-based applications.
- Chapter 6 summarises and concludes our work. A discussion of limitations and future work is also provided.

CHAPTER 2 BACKGROUND

In this chapter, we present a variety of databases, including a description of SQL and NoSQL databases and the differences between them. We introduce a description of the cloud patterns studied in this thesis. We include a description and a discussion about the Power-API software profiler and we explain how it works and we highlights its precision.

2.1 Databases

The database is an organized system to back up and recover data effectively, managed by a database management system (DBMS) [9]. In this section, we present the two families of databases that we use in our study: two relational databases, Postgresql and MySQL, and one NoSQL database, MongoDB. We choose these three databases because they are frequently used in cloud-based applications [10], [11], [12], [13] and also because they are the most popular databases available today.

2.1.1 Relational Databases

A Relational Database is a collection of data items organized by tables, records and columns, with well defined relationships between tables. Relational Databases provide a programming interface for database interaction [14]. Relational databases are still hard to scale with cloud-based applications. RDBMSs support indexes which is a mechanism that allows sorting a number of records on multiple fields. An example relational database table storing information about a football player looks as follows:

```
id |   name   | lastName | country |birthYear|     club      |  location

7  |Cristiano |  Ronaldo | Portugal|  1985   |  Real Madrid  |   Spain
```

Contrary to NoSQL databases, relational databases follow the ACID Transaction support [15]. Most RDBMSs guarantee ACID transactions. ACID is an acronym for the following four properties [16]:

- Atomicity: each transaction should be executed entirely, otherwise it is not executed. Accordingly, it is called an atomic transaction.
- Consistency: each transaction takes the database from one valid state to another.
- Isolation: each transaction is isolated from any other transaction.
- Durability: each transaction is saved by the system even in the event of a system failure or restart. In other words, a transaction must persist and thus never be lost.

In this work, we use MySQL [17] and PostgreSQL [18] as two relational databases. We choose these two databases because they are the most popular relational databases in the last few years [19].

MySQL database

MySQL is a relational database system that was developed in 1996 by Michael Widenius for the Swedish Company TcX. At the begining, MySQL database was developed for open source distributions (i.e., Linux and Solaris) [20]. While not being open source, MySQL does have a non-restrictive licensing that allows organizations to use the product for in-house projects for free [20].

MySQL has performance criteria: it is very fast, portable, simple, supports a variety of programming interfaces and has the ability to be accessed from anywhere over the Internet. All these mentioned criteria enabled MySQL to be a popular and a very known RDBMS [20].

As indicated in figure 2.1 [1], MySQL architecture contains three levels that work together and in a complementary manner to respond to a request. The first level is the "*Connection management*" which ensures the connection to the MySQL database through TCP/IP using SSH or SSL encryption protocol. The second level contains the "*SQL parsing*", "*SQL execution*" and "*SQL caching*". The "*SQL parsing*" parses the obtained query and then forward it to the "*SQL execution*" unit to be executed. Besides that, "*SQL caching*" saves the result obtained by a query in a memory for a period of time. This period of time depends on the frequency of application of the corresponding request from the database. Finally, the third level corresponds to the storage engine. This storage engine could be either by default "*MyISAM*" or "*InnoDB*" or "*HEAP*" or "*NDB*".

[1] http://books.gigatux.nl/mirror/highperfmysql/0596003064/hpmysql-CHP-2-SECT-1.html

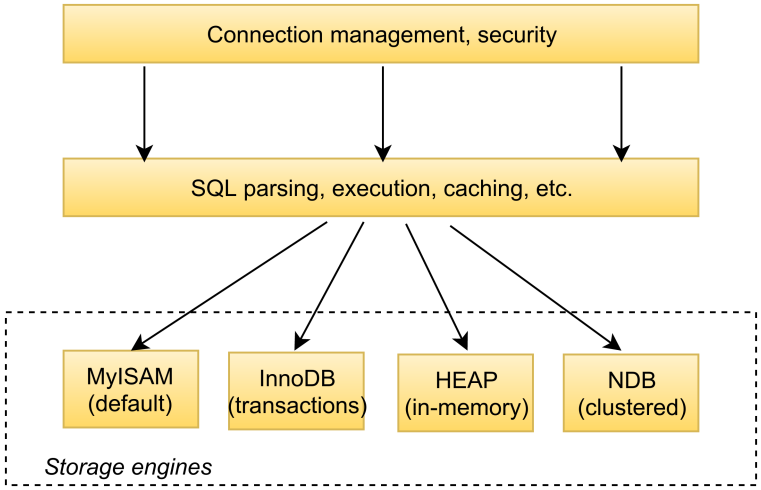


Figure 2.1 MySQL architecture

PostgreSQL Database

PostgreSQL was created by professor "Michael Stonebraker" [2]. PostgreSQL is a public domain software that operates as a single machine. Contrary to MySQL, which does not support parallelism in query execution, PostgreSQL does support it and all its operations use a *Multi-Version Concurrency Control* (MVCC). This RDBMS runs on Unix or Linux distributions. For our project, we use PostgreSQL version 9.1.

PostgreSQL database has an architecture based on the concept of process, where the execution units present the process. While most of the databases use threads instead of processes for executing transactions, we believe that these systems have a similar architecture to PostgreSQL. As indicated in Figure 2.2 [3], this architecture has a PostgreSQL engine (yellow color) which contains a "postmaster", "backends", and a "shared memory". The *postmaster* acts as a central process that listens for requests received on a definite port. The *Postmaster* creates a process for each connected client, and the rest of the communication is done directly between the "back-end" and "client", using a channel based on the "buffered sockets" (by using TCP/IP or CPI sockets). The PostgreSQL database has a mechanism that allows to set the maximum threshold of "backend" executed in such a way that the maximum level is reached. In this case, PostgreSQL blocks new clients. This locking mechanism effectively limits the number of simultaneous transactions by avoiding a deterioration in peak situations. In addition, we should mention that this mechanism is necessary because the process management is expensive. In addition, we believe that this process management can affect

[2]https://www.postgresql.org/about/history/
[3]http://www.cs.mcgill.ca/ kemme/papers/phd-letter.pdf

the energy consumption. PostgreSQL architecture also supports a “shared memory”, which contains stored data required for use by the various “backends”. The shared memory involve the “back-end process”, “lock table” and “buffer pool”. In general, access to shared memory is controlled by the synchronization primitive: `semaphore`.

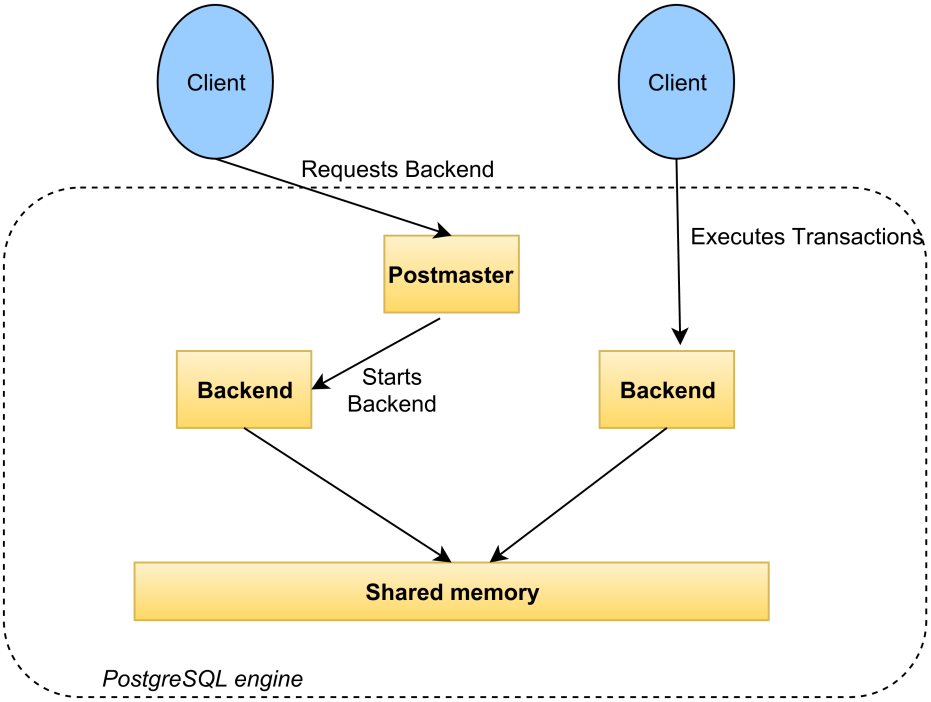


Figure 2.2 PostgreSQL architecture (process structure)

2.1.2 NoSQL Databases

NoSQL databases [1] are non-relational and distributed databases [21]. Contrary to relational databases (as mentioned in 2.1.1), NoSQL databases follow the CAP theorem [22], that is Consistency, Availability and Partition tolerance.

NoSQL databases are categorized based on the way they store data such as document store (e.g., MongoDB [23]) and key-value stores (e.g., BigTable [24], Dynamo [25]). In our study, we examine MongoDB database [23] because it is the most popular NoSQL database available today; it is widely used by eBay, IBM, Expedia, and The New York Times.

MongoDB

MongoDB uses different concepts compared to relational databases. Table 2.1 shows the terminology of its concepts. A MongoDB deployment holds a set of databases. Compared

to relational databases, where each database backup data as tables, MongoDB stores each database in the form of collections. Each collection holds a set of documents. A document, that has a format known as BSON, is a set of fields. A field is a `key-value` pair where `Keys` are always strings and `Value` types can be string, object, boolean, or integer. Similarly to many databases, each document contains an automatically-added field called `id`, which assigns the document a unique `id`.

Table 2.1 Terminology between MongoDB and RDBMS

RDBMS	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Join	Embedding & Linking

An example of MongoDB document storing information about a football player looks as follows:

```
{
  _id: 7000e717b7543fe00k23,
  name: Cristiano,
  lastName: Ronaldo,
  country: Portugal,
  birthYear: 1985,
  club: Real Madrid,
  location: Spain
}
```

All values are atomic. The example below shows the same example mentioned previously but including some complex values:

```
{
  _id: 7000e717b7543fe00k23,
  name: Cristiano,
  lastName: Ronaldo,
```

```
  country: Portugal,
  birthYear: 1985,
  address: {
   city:    Madrid,
   street:  Camino Sintra - 28050 Valdedebas.
  },
  club: Real Madrid,
  location: Spain,
  hobbies: [cooking, music, swimming]
}
```

MongoDB provides a technique called *embedding* allowing to store a document within a document. The `address` field shows the use of this technique.

While there are no schemas in MongoDB, the names of the databases and the names of their respective collections are stored as `metadata`. Also, the defined indexes are stored as metadata.

`Indexes` are supported in MongoDB. They work similarly as to how indexes work in RDBMSs, allowing clients to index arbitrary fields within a collection. Even embedded fields can be indexed, e.g., *address.city* from the second example above. Fields that have been indexed support both random access retrieval and range queries. Indexes can be created at any time in the lifetime of a collection. The `id` field of all documents is automatically indexed.

As indicated in Figure 2.3 [4], MongoDB architecture contains 3 levels: "*MongoDB Query Language*", "*MongoDB Data Model*" and "*MongoDB Storage Engines*". The *MongoDB Query Language* allows users to access and operate their documents in sophisticated ways. This mechanism allows to support analytical and operational applications. The *MongoDB Data Model* allows to store and combine data of any structure easily. Besides that, the *MongoDB Data Model* allows to customize the schema of the database without having a significant impact on the performance of the query execution. The third level of the MongoDB architecture represents the *MongoDB Storage Engine*. It supports five different types of storage engines: WiredTiger, MMAPv1, In-memory, Encrypted, and 3rd party engine. We should mention that MongoDB provides `WiredTiger` as a default storage engine that provides the best of storage performance among all the mentioned storage engines [5].

[4] https://www.mongodb.com/mongodb-architecture
[5] https://docs.mongodb.com/manual/core/storage-engines/

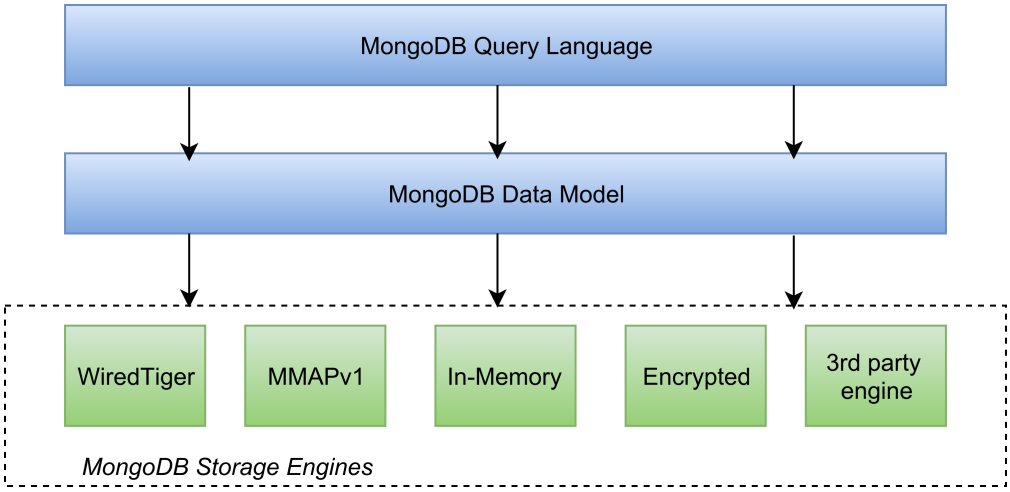


Figure 2.3 MongoDB architecture

2.2 Cloud Patterns

Cloud patterns are widely used in cloud-based applications to improve data organization and access. In this section, we present the three cloud patterns that have been used in this study. We choose these three cloud patterns in our study because they are the most relevant for data management.

2.2.1 Local Database Proxy

The *Local Database proxy pattern* is widely used to replicate the data between servers in a cloud environment. More precisely, this pattern ensures the replication of data between master and slave databases. To replicate the data, the *proxy pattern* uses a proxy to handle read and write requests, where it takes into account the type and the workload of the received request to assign it either to the master or to the slave database [26]. Besides that, this pattern provides the ability to assign `read requests` directly, where these requests are performed by the slave nodes. This pattern assigns `write requests` to the master node, which in turn ensures the replication of these requests in its slave nodes. Interestingly, the *proxy pattern* supports `elasticity`, during the execution, by providing the opportunity to add or to delete slave nodes.

2.2.2 Local Sharding-Based Router

The *sharding pattern* is widely used to split and distribute the data between multiple database nodes. The divided data between the databases nodes are called *Shards*. In addition, this

pattern ensures the *scalability* of applications when they handle read or write requests. According to a mechanism that is somewhat similar to the *proxy pattern*, the *sharding pattern* uses a *router* to attribute the received request to the appropriate database node (shards) [26]. The *sharding pattern* gives the possibility of *scalability* by adding nodes (shards) during execution. Costa et al. [27] mentioned that this pattern could be implemented through three different strategies: the first strategy is a range of value, the second strategy is hashing, and the third strategy is a *shard key*, which assigns each data corresponding to the suitable shard using this key [27].

2.2.3 Priority Message Queue

The *Priority Message Queue pattern* is highly recommended to manage queries of different types, by respecting the FIFO mechanism. Each request has a priority value where the *message queue pattern* handles the requests received, taking into account their priority. The request that has a higher priority value will be treated before the request that have less priority value [28]. As indicated in the aforementioned pattern (*sharding pattern*), this pattern gives the possibility to improve the `scalability` [6].

2.3 Power-API

Power-API is a profiler that provides power information (in watts converted to joules to measure the energy) per PID (*Process Identifier*) for each system component (e.g., CPU, memory, etc.) [29]. Power-API uses sensors and analytical models for its energy estimation. Noureddine et al. [30] described the **PowerAPI CPU model** as following:

$$P_{CPU}^{PID}(d) = P_{CPU}(d) \times U_{CPU}^{PID}(d) \tag{2.1}$$

Where $P_{CPU}^{PID}(d)$ is the CPU power consumed by the specific PID during a given duration d, $P_{CPU}(d)$ is the global CPU power during d and $U_{CPU}^{PID}(d)$ represents the process CPU usage during d.

This API allows to estimate the amount of power required by the CPU to execute a process (*At the corresponding PID*). Noureddine et al. [30] performed a test to evaluate the accuracy of Power-API profiler using `PowerSpy` [31]. The results of this experiment showed that there is only low perturbations between the energy consumption measured by `PowerSpy` and the energy estimations of Power-API profiler [29]. For this reason (i.e., high accuracy), we selected Power-API profiler for our study. In addition, according to an experiment performed

by Abtahizadeh et al. [32], Power-API profiler does not introduce noise in its measurements.

CHAPTER 3 LITTERATURE REVIEW

In this section, we present works related to software energy measurement and the impact of databases and patterns on the performance of applications. Section 3.1 discusses the energy measurement approaches and tools. Section 3.2 discusses how the application design can affect the energy consumption. Section 3.3 discusses challenges in energy consumption and security. Section 3.4 highlights related works discussing the impact of cloud patterns on applications performance. Section 3.5 focuses on the impact of databases on the performance of applications.

3.1 Energy Measurement Approaches and Tools

Many works have been introduced to measure the energy consumption of software. Estimating the energy consumption of software can be performed by modeling hardware resource usages. In this section, we present the main energy measurement approaches and tools in the literature.

3.1.1 Energy Measurement Approaches

Seo et al. [33] proposed an estimation-based approach that calculates the energy consumption of a software component using the following formula from Equation 3.1 [34]:

$$E_{Component} = E_{Computational} + E_{Communication} + E_{Infrastructure} \tag{3.1}$$

Where $E_{Computational}$ is the energy estimation of a hardware resource utilization (i.e., CPU processing, memory access, queries executed by the Disk). $E_{Communication}$ is the energy estimation of the data transferred over the network and $E_{Infrastructure}$ represents the energy estimation of the virtual machine.

The advantage of this approach is that it allows for the estimation of the energy consumption of Java applications running in virtual machines.

Kansal et al. [35] also proposed an approach to calculate the energy consumption of an application. The major difference between their approach and the Seo's approach is that it allows for the estimation of energy consumption even when the application is in *wait* and *idle* states. Applications consume energy when waiting for requests from the disk. The total energy consumption of an application is given by Equation 3.2 [34].

$$E_{App} = E_{Active} + E_{Wait} + E_{Idle} \tag{3.2}$$

Where E_{Active} represents the energy spent by the application when it is **running**. E_{Wait} is the energy cost of the application in the **wait** state and E_{Idle} represents the energy cost of the application in the **idle** state.

Trefethen et al. [36] introduced an approach based on sensors. Sensors are used between the application on which measurements are made and the power source (i.e., the source of electric current). The sensors collect accurate data from the power source. The collected data, stored in a data collection server, can be, for example, a voltage measurement. The operation of the sensors is not always continuous but periodic over intervals of time. After running the measured software, the data read part of the sensors and the software are correlated. The energy consumed by the application is given by Equation 3.3 [34].

$$E = \int_T P_W dt - P_I T \tag{3.3}$$

Where P_W represents the split-second power profile. P_I is the *idle* profile of the software, and T represents the execution time of the running software.

3.1.2 Measurement Tools

We now describe the most common energy measurement tools available in the literature beside Power-API presented in Section 2.

Eprof

Pathak et al. [37] proposed a fine-grained energy profiler called `Eprof`. The authors showed that this profiler can be used to estimate the energy consumption of an application executed in a smartphone. `Eprof` can measure the energy consumption of mobile applications only for two plateforms: Android and windows mobile phone. Pathak et al. [37] conducted multiple experiments with Eprof and reported that different versions of the same mobile application can have a different amount of energy consumed. They also showed that there is no a correlation between the amount of energy consumed by an application and its execution time.

GreenTracker

Amsel et al. [38] introduced a tool called `GreenTracker`. GreenTracker can estimate the power consumption of the CPU during the execution of a software. GreenTracker was used

to profile software from different categories: word processing software, audio and internet browsers (e.g., Opera, Google Chrome, Internet Explorer, and Mozilla Firefox). Results showed that `Internet Explorer` is the most efficient internet browser among all the experimented browsers.

Powerscope

Flinn et al. [39] introduced a profiler of energy consumption called `Powerscope`. `Powerscope` was designed mainly for measuring mobile applications. `Powerscope` is very similar to `PowerAPI` in the sense that it estimates the energy consumption of a software (i.e., mobile applications) at the process level. However, `Powerscope` follows a different estimation approach (than `PowerAPI`) to compute the energy consumed by different procedures within a specific process. This profiler uses three different software components for its operation [34]:

1. an *Energy Monitor* that records energy samples from a digital multimeter.
2. a *System Monitor* that records important system information, such as the value of the Process Identifier (PID) and the Program Counter (PC).
3. an *Energy Analyzer* that calculates the energy consumption of a software using the `energy samples` recorded by the *Energy Monitor* and the `system information` recorded by the *System Monitor*. The energy consumption is calculated following Equation 3.4:

$$E \approx V \sum_{t=0}^{n} I_t \Delta_t \tag{3.4}$$

Where E represents the total energy usage collected by the *Energy Analyzer* over n samples. V represents the value of the voltage, I_t is the current time, and Δ_t is the duration time (the interval of time).

Joulemeter

`Joulemeter` [40], [41] is a software tool developed by Microsoft that can estimate the energy consumed by a computer, a server, or a virtual machine. This tool also allows modeling the impact of different components (such as CPU utilization, memory utilization, or screen brightness) on total energy consumption. Interestingly, `Joulemeter` can be used to estimate the energy consumption in a data center or even during software development.

3.2 Energy Consumption and applications design

Pinto et al. [42] used a popular Web site, StackOverflow, as primary data source to understand the software engineers' perspectives on issues related to the energy consumption of their applications. The authors showed that the number of questions on energy consumption increased by 183% only in a year (from 2012 to 2013). Their finding suggest that software engineers are actively seeking guidelines about the energy efficiency of their applications.

The most closely related work to ours is that by Abtahizadeh et al. [32]. They conducted an empirical study that aimed to compare the energy efficiency of three cloud patterns: Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue. Similar to our work, they measured the energy consumption of applications using Power-API profiler. However, they only considered MySQL database. They showed that cloud patterns can effectively reduce the energy consumption of a cloud-based application. They showed also that the implementation of the Local Database Proxy Pattern can significantly improve the energy efficiency of a cloud-based application and that this pattern is more appropriate for cloud applications handling huge requests of read loads. They also found that a combination of the Local Database Proxy pattern with the Local Sharding-Based Router does not have a significant impact on energy consumption.

In the same direction, Manotas et al. [43] conducted an empirical study in which they investigated the impact of four Web servers (*i.e., mongrel, puma, thin, webrick*) on the energy consumption of a Web application. They showed that the energy consumption of a Web application depends on the Web server used to handle requests. They also showed that the impact of the Web server depends on the features that are used. Each Web server can increase or decrease the energy consumption of the Web application, depending on the features for which it is executed.

Capra et al. [44] conducted an empirical study that investigated the impact of Management Information Systems (MIS) on the energy consumption. Using a server machine, a hardware kit and a tool generating workloads, they showed that a server running an application consumes around 72% power more than a server in the idle mode. By experimenting with many MIS applications, they observed that each MIS application requires a different amount of energy. They also observed that the use of different operating systems, more precisely the use of Windows or Linux, have a significant impact on the energy consumption of an application.

Bunse et al. [45] compared the energy consumption of various sorting algorithms, running on an embedded system. The sorting algorithms investigated are selectionsort, insertionsort,

quicksort, heapsort, shellsort, shakersort, mergesort, and bubblesort. The results of their study show that insertionsort is the more energy-efficient algorithm. They also found no correlation between the energy consumption and the time complexity of the algorithms. Meaning that algorithms that consume more energy are not necessarily those that take longer time to be executed.

Arunagiri et al. [46] compared different implementations of algorithms used to solve the global stereo matching problem. The metrics used for comparisons are: the amount of energy consumed, the average response time and the global minimum cost achieved. Their results show that the graph cut algorithm performs better than the simulated annealing algorithm.

Sahin et al. [47] investigated the energy efficiency of 15 structural, behavioral and creational design patterns, implemented in an application. For each pattern, they examined the energy consumption of the versions of the application before and after applying design pattern. Their results show that design patterns have a significant impact on energy consumption. However, the impact on energy consumption of different types of design patterns is not the same. Certain design patterns like Decorator can increase the energy usage of an application by up to 700%. Similar study was conducted by Bunse et al. [48]. However, they investigated the energy efficiency of design patterns implemented in smartphones applications. The patterns considered in their study are: Facade, Abstract Factory, Observer, Decorator, Prototype, Template Method. They measured the energy consumption of applications running on different smartphone models (e.g., Samsung Galaxy S2, Nexus One) using the `PowerTutorApp` tool from Michigan University. Their results show that in some cases, design patterns can significantly increase the energy consumption of mobile applications. In particular, similarly to Sahin et al. [47], they found that `Decorator` pattern strongly increases the energy consumption of mobile applications.

Sahin et al. [49] conducted an empirical study in which they investigated the effect of code refactorings on the energy consumption of applications. The study was conducted using 197 applications implementing 6 commonly-used refactorings. Their results show that code refactorings affect the energy consumption of applications. More specifically, they found that all the tested code refactorings have the potential to both increase and decrease the energy consumption of an application, with the exception of `Extract Local Variable`, which consistently decreased the energy consumption of the studied applications.

In the same area, Procaccianti et al. [50] conducted an empirical study that investigated the impact of three different `ORM` approaches on the energy consumption of applications. `ORM` stands for "*Object-Relational Mapping*", which allows a link between a relational database and object-oriented programming. This technique is widely used for transforming a table

stored in a relational database in an easily manipulable object via its attributes. In Java, we can mention "`Hibernate`" among the frameworks commonly used to perform this task. Yet, this technique facilitates the manipulation of relational databases using object-oriented programming, but in this paper, they showed that this technique has a negative effect on the energy consumption of an application. In more details, the authors conducted their experiments specifically on PHP applications, testing three different `ORM` approaches that manipulate an SQL relational database: *plain SQL queries in the source code*, and *Propel* and *TinyQueries*. In addition, they guided their experiments taking into account three factors: the aforementioned three approaches, the size of the relational database, and the type of the query used to extract the desired data from the relational database (SQL). The used requests manipulate the popular `CRUD` operations (i.e., *Create, Read, Update, Delete*). Their results are presented mainly along two axes: the energy efficiency and the execution time of the PHP application. Interestingly, they showed that the tested approaches provides benefits and trade-off in terms of execution time and energy consumption: the *Propel* framework is the most energy consuming and is the slowest among the three tested approaches. The most suitable approach is the use of *plain SQL queries in the source code* because it is least energy consuming and the fastest in all the test cases. The third approach, that of *TinyQueries*, performed much better than *Propel*, but slightly worse than the *pure SQL queries*.

3.3 Energy Consumption and Security

Recently, researchers have started investigating the energy cost of security tools. Kim et al. [51] constructed a power monitor that collects power samples and defines a history of the energy consumption from these samples. They also proposed a data analyzer that generates a power signature, after noise filtering. According to experimental results on a HP iPAQ running a Windows mobile OS, the tool achieves a true positive rate of 99% in the classification of malware targeting mobiles.

Merlo et al. [52] proposed a methodology for computing the energy consumption patterns of two mobile devices subsystems i.e., CPU and wireless network card (WiFi). The proposed methodology has been implemented and tested during a ping-flood attack conducted during the execution of a legitimate application. Experimental results show that the proposed methodology is robust, accurate, and reliable for detecting the occurrence of such attack.

Hoffmann et al. [53] studied the typical power consumption of various aspects of common Android mobile phones, and found that power consumption varies greatly in practice. They attempted to measured the energy consumed by malwares and concluded that in practice, the additional power consumed by malware is too small to be detectable using average con-

sumption rate measurements.

Merlo et al. [54] introduced a new concept called "Energy-aware Intrusion Detection Systems", which recognizes malicious behavior in mobile devices according to their energy footprint. To implement this concept, the energy consumption of several hardware components of a mobile device must be measured with a sufficient level of accuracy. They investigated two measurements approaches: High Level and Low Level (HL and LL). HL measures can provide important information not only in terms of energy consumption, but also the ability to isolate the power consumption of each component. Unlike the LL approach, the measures of the HL approach are not instantaneous. The LL approach provides instantaneous information on the consumption of the entire system at a granularity of 250 milliseconds. Given that such measures don't separate the power consumption of each component (audio, screen, 3G and GPS), the authors opted for a combination of the two approaches, which they called medium level (ML) measurement. ML uses a high-level API to take action directly from the battery driver to have both instantaneous measurements and information on the consumption of each component. With this approach, they were able to easily identify ping-flood attacks.

Palmieri et al. [55] report about an attack targeting cloud services providers, where attackers performed a denial of service focusing on energy (energy-oriented denial of service). In this attack, the goal is to use available resources to increase considerably the energy consumption without triggering protection mechanisms in monitors. This type of attack can have a heavy financial impact on cloud services providers.

3.4 Impact of Cloud Patterns on Applications Performance

Geoffrey et al. [6] conducted an empirical study aimed at understanding the impact of cloud patterns on Quality of Service (QoS). Multiple versions of a multithreaded RESTful application were deployed in a cloud environment. To implement their RESTful application, They used the following three cloud patterns: Local Database Proxy Pattern, Local Sharding-Based Router, and Priority Message Queue. They used a MySQL database to store the data of the application. The authors measured the QoS of the application using the following metrics: the average and the maximum number of requests per second, and the application response time. Their obtained results show that cloud patterns can impact the QoS of cloud-based applications. More precisely, they reported that the implementation of the Local Database Proxy pattern can significantly impact the QoS of a cloud-based application in terms of average and maximum number of requests processed per second and also the average response time. Besides that, they mentioned that their implementation of some combinations

of the aforementioned patterns can significantly affect the QoS of the studied multithreaded application.

Ardagna et al. [56] evaluated the impact of five scalability patterns on the performance of a Platform as a Service (PaaS). These five patterns are: Clustered, Single, Shared, Multiple Clustered Platform, and multiple Shared patterns. For each of the aforementioned patterns, the authors measured the number of transactions processed per second and the response time. Their obtained results show that each pattern can affect the way virtual machine resources are added and removed.

3.5 Performance of Relational and NoSQL databases

Researchers have compared the response time of SQL and MongoDB databases [57]. To the best of our knowledge, there is no previous work that investigated the impact of NoSQL databases on the energy consumption of cloud-based applications.

Hammes et al. [58] presented a comparative study of SQL and NoSQL databases in the cloud, where they highlighted the performance of both PostgreSQL database and MongoDB database, implemented on a cloud server. In this paper, they examined the performance of the databases using two distinct real-world scenarios: the first scenario represents a highly structured data and the second scenario represents unstructured data. By testing Create, Read, Update, and Destroy operations (around 1,400,000 operations) for the two databases (PostgreSQL and MongoDB) with the two distinct scenarios, they observed that PostgreSQL databases perform better than MongoDB databases in cloud environments.

Rajat et al. [59] also presented a comparative study of these popular databases, i.e., MySQL and MongoDB. They examined different operations (CRUD operations) to measure the performance of the two databases. They used three categories of dataset (i.e., Small, Medium and Large). Similarly to our results, the authors showed that MongoDB database performs better than MySQL for complex queries, especially those involving multiple joins. They also showed that MySQL databases perform better than MongoDB databases for small datasets. In addition to these findings, they outlined some guidelines for software developers; advising them to use MySQL databases rather than MongoDB in the case of medium data without complex queries and MongoDB rather than MySQL databases in the case of medium data involving complex queries and joins.

Loannis et al. [60] proposed a cloud-enabled framework for monitoring NoSQL databases. Their proposed framework was applied on three popular NoSQL databases: HBase, Cassandra and Riak. Results showed that Cassandra databases are fast for write operations. They

also scale well during node additions, without a turning point phase. HBase is reported to be the fastest among the three studied databases. HBase scales very well for node additions. However, Riak databases can re-balance nodes automatically. Based on these results and measurements, the authors presented a prototype implementation of their automatic cluster that facilitates the execution of self-acting elastic operations of any NoSQL engine. Their proposed prototype offer developers and architects the possibility to test and verify the degree of scalability of any application using these noSQL databases.

Dory et al. [61] introduced an approach to measure the elasticity of NoSQL databases deployed in a cloud environment. The proposed approach was assessed on MongoDB, HBase, and Cassandra databases using a realistic load (48 nodes). In addition, they used *Rackspace*[1] as the cloud infrastructure. The popular *Wikipedia* database was used to generate the dataset of the study. We should mention that this study gives measurements only for systems, using the aforementioned NoSQL databases, that scale up. In other words, as a limitation for this work, these elasticity measures does not work with a system that scales down. The authors showed that the technical choices and the architecture of each of the studied NoSQL database can affect the operation of adding new nodes (i.e., to scale up).

[1]https://www.rackspace.com/cloud/servers

CHAPTER 4 METHODOLOGY

In this chapter we introduce our research questions, describe the objects of our study, as well as our experimental design, describe the cloud environment used for the experiments, clarify our data extraction process and analysis method.

We want to empirically evaluate the impact of three different Databases (MySQL, PostgreSQL, and MongoDB) on the energy consumption of cloud-based applications. We also want to evaluate the impact of three cloud patterns (i.e., Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue) on the energy consumption of these three different databases.

4.1 Research Questions

Our study aims at answering the following research questions:

- RQ1: Does the choice of MySQL, PostgreSQL, and MongoDB databases affect the energy consumption of cloud applications (when no cloud patterns are implemented)?
- RQ2: Does the implementation of Local Database Proxy, Local Sharding Based Router, and Priority Message Queue patterns affect the energy consumption of cloud applications using MySQL, PostgreSQL, and MongoDB Databases?
- RQ3: Do the interactions between Local Database Proxy, Local Sharding Based Router, and Priority Message Queue patterns affect the energy consumption of cloud applications using MySQL, PostgreSQL, and MongoDB databases?

4.2 Objects

We choose three systems for each experiment, two applications developed in Java and one application developed as a combination of PHP and Microsoft .NET. We performed each experiment on three different systems, because one system could be intrinsically more energy consuming.

At first, for Experiment 1, we implement and deploy a multi-threaded distributed application that communicates through REST calls. We use GlassFish 4 as application server. The application interacts with one of the three chosen databases management system. We used the *Sakila sample database* [62] provided by MySQL. *Sakila database* contains a large number

of records, and for this reason we believe that it is interesting for experiments. We adapted the schema of the *Sakila database* to PostgreSQL and MongoDB databases.

For Experiment 2 and 3, we use DVDStore and JPetStore systems. These two cloud-based applications have been used in multiple other studies from the literature [63]. DVDStore [1] is an open-source simulation of an e-commerce site. It has been released under the open-source GNU General Public License (GPL). We use DVD Store as a test workload, which includes a back-end database component, a driver programs, and a Web application layer. DVD Store is provided with the implementation of Microsoft SQL Server, Oracle, MySQL, and PostgreSQL databases. We refactor the code of DVD Store to allow it to connect to a MongoDB database.

Similarly to DVD Store application, we also modified the code of JPetStore to implement connections to MySQL, PostgreSQL, and MongoDB databases. JPetStore [2] is an e-Commerce Web application that offers various types of pets online. JPetStore is coupled with a DAO layer and uses spring MVC and struts.

As shown in Figure 4.1, our cloud environment contains three servers. We used a switch to connect these servers on a private network. The first server is the master that executes the cloud-based application (the RESTful application, DVD Store application, or JPetStore application). The master server has the following characteristics: Intel CPU Xeon X5650 with 8 GB RAM, and 40 GB disk space. The two other servers contains eight slave database nodes *running on VMware ESXi*: 4 on each server, where each slave virtual machine has a virtual processor Intel CPU QuadCore i5 with 1 GB RAM, and 24 GB disk space.

4.3 Design

In our experiments, we use a combination of databases and cloud patterns encoded using a letter and a number. The Local Database Proxy pattern has three implementation strategies: Random Allocation (P1), Round-Robin (P2), and Custom Load Balancing (P3). The Local Sharding Based Router pattern also has three strategies: Modulo Algorithm (P4), Consistent Hashing (P5), and Lookup Algorithm (P6). The Priority Message Queue pattern is called P7. The databases are named: MySQL (D1), PostgreSQL (D2), and MongoDB (D3).

The Round-Robin strategy chooses an instance of the pool in a round-robin fashion, whereas the Random Allocation strategy selects the instance randomly. On the other hand, the Custom strategy uses a more sophisticated method to pick the best instance to choose. The

[1] `http://linux.dell.com/dvdstore/`

[2] `https://github.com/mybatis/jpetstore-6`

choice is based on the response time and the number of open connections on the slave nodes. Sharding pattern requires using many clones of the same database in different shards. We used a subset of the *Sakila database* because the sharing pattern requires the use of an independent data. Three flavors of the Sharding pattern are used. In the Modulo strategy, the primary key is divided by the number of shards and the remainder is used to select the server, which executes the request. Concerning the Lookup strategy, this strategy use a table with a number of slots bigger than the number of servers to select the instance. Finally, the consistent hashing algorithm uses hashes to select the server.

However, in the Priority Message Queue pattern, requests are processed by the server based on their priority. There is only one strategy to implement this pattern.

We use a client application that sends 100 database manipulation requests to a server. We measure the average response time of the server. The generated requests are basically a mix of select and write. Response time (read and write requests) is the main metric used to measure the performance of the application.

We perform our experiments using different numbers of clients, which are simulated using a multi-threaded architecture. The number of clients simulated varies from 100 to 1500 clients (100, 250, 500, 1000, and 1500). Each execution is done using different databases and different cloud patterns.

To get precise measurement results, we repeat each scenario five times and we compute the average for each performance metric. The total reaches 150,000 concurrent requests, which reflects the workload of real cloud based applications.

4.4 Data Extraction Process

This section explains in details the data collection for estimating the energy consumed by a cloud-based application. The material used for data collection is described in Section 4.2. The procedure below details the steps taken to have the results of the energy assessment of the cloud-based application (see Figure 2.2).

Energy Data Collection Procedure

CollectData(*VMs, CloudApp, Profiler*)
Begin
Start*CloudApp*()
Execute*CloudApp*(x) // *Seconds*
  for all *VM* ∈ *VMs* **do**
    Start*Profiler*()
    Execute*Profiler*(x) // *Seconds*
    Finish*ExecProfiler*()
  end for
Finish*ExecCloudApp*()
End

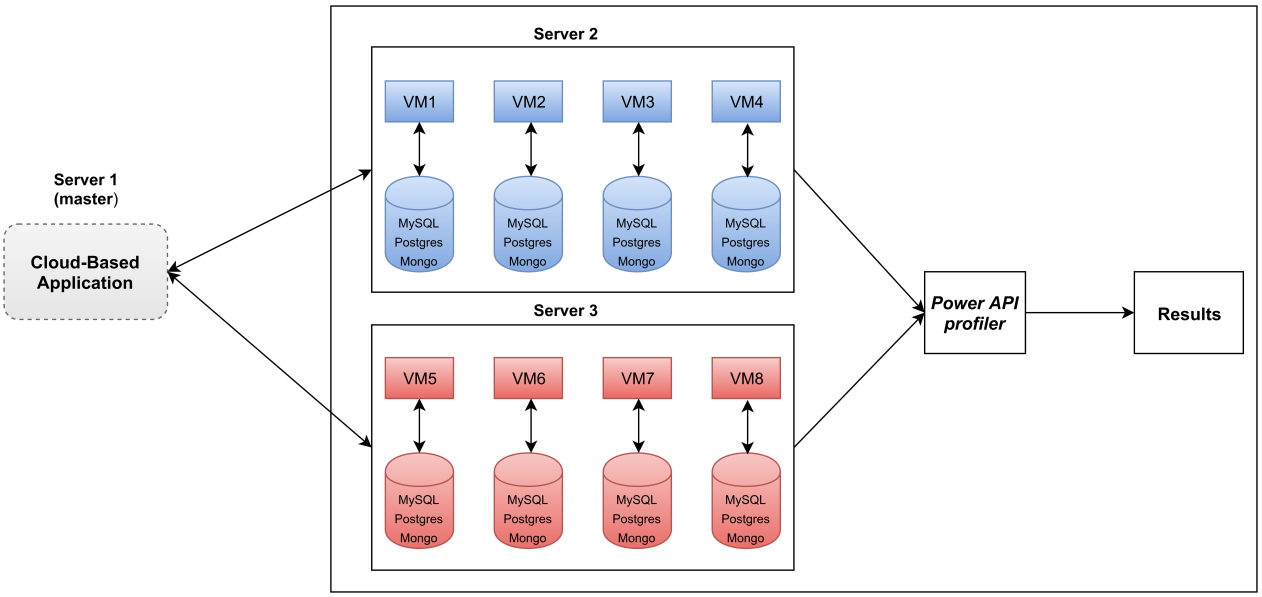


Figure 4.1 Energy Consumption Data Extraction Process

4.5 Independent Variables

MySQL, PostgreSQL, and MongoDB databases are the independent variables of our study. Also, the three studied cloud patterns: Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue patterns, as well as the strategies of these cloud patterns are considered as independent variables.

4.6 Dependent Variables

We measure the application response time (corresponding to select and insert requests) in *nanoseconds* and then convert it to *milliseconds*. We measure the energy consumption using the Power-API profiler (in watts, which we convert to joules (J)). Both measures are dependent variables.

4.7 Hypotheses

To answer our research questions, we formulate the following null hypotheses, where P0 is the experiment in which we compare the energy consumption and response time of the three versions of each application using respectively MySQL, PostgreSQL, and MongoDB databases. Px (x $\in$ {1 ... 6}), and P7 are the different patterns.

In each experiment we compare two versions of a same application implementing two different databases Dy, Dz (y, z $\in$ {1, 2, 3} and y$\neq$z), with the same (combination) of patterns.

- H^1_{0yz}: There is no difference between the average amount of energy consumed by applications implementing databases D_y and D_z (without any cloud pattern).
- H^1_{xyz}: There is no difference between the average amount of energy consumed by applications implementing databases D_y and D_z in conjunction with patterns Px.
- H^1_{xyz7}: There is no difference between the average amount of energy consumed by applications implementing databases D_y and D_z in conjunction with the combination of patterns Px and P7.

To have more in-depth understanding of the trade-offs between energy consumption and response times of the cloud-based applications, we also formulate the following null hypotheses:

- H^2_{0yz}: There is no difference between the average response time of databases D_y and D_z by applying the design P0.
- H^2_{xyz}: There is no difference between the average response time of databases D_y and D_z by applying the design Px.
- H^2_{xyz7}: There is no difference between the average response time of databases D_y and D_z by applying the combination of designs Px and P7.

4.8 Analysis Method

To analyze our collected data (i.e., response time and energy consumption measurements), we perform the Mann-Whitney U test [64] to test the following hypotheses: H^1_{0yz}, H^2_{0yz}, H^1_{xyz}, H^2_{xyz}, H^1_{xyz7}, H^2_{xyz7}. The Mann-Whitney U test is a non-parametric statistical test whose relevance is reflected in the assessment of two independent distributions.

We also computed the Cliff's δ effect size [65] because effect sizes are important to understand the magnitude of the difference between two distributions. In addition, Cliff's δ represents the degree of interlock between two sample distributions [65]. Cliff's δ is also more reliable and robust than Cohen's d effect size [66]. Cliff's δ effect size value ranges from -1 to +1. We should mention also that Cliff's δ effect size value is zero when two sample distributions are the same [67]. In all our tests, we reject the corresponding null hypothesis (i.e., there is a significant difference between the the two distributions) when its p-value < 0.05.

Interpreting the Effect Sizes:

As indicated in Table 4.1, the Cliff's δ value may correspond to three different categories of effect sizes: small, medium and large. The effect size is negligeable is the Cliff's δ value is less than 0.147, small if the Cliff's δ value is greater than 0.147, but less than 0.33, medium if the the Cliff's δ value is between 0.33 and 0.474, and large if the Cliff's δ value is greater than 0.474 [68].

Table 4.1 Mapping Cohen's d to Cliff's δ.

Cohen's Standard	Cohen's d	% of Non-overlap	Cliff's δ
small	0.20	14.7%	0.147
medium	0.50	33.0%	0.330
large	0.80	47.4%	0.474

CHAPTER 5 RESULTS AND DISCUSSIONS

In this chapter, we now present the results of our research questions. At the end, we also discuss our findings.

5.1 Results

5.1.1 RQ1: Does the choice of MySQL, PostgreSQL, and MongoDB databases affect the energy consumption of cloud applications (when no cloud patterns are implemented)?

Tables 5.1 and 5.2 summarizes the results of Mann-Whitney U test and Cliff's δ effect sizes for the energy consumption and the response time.

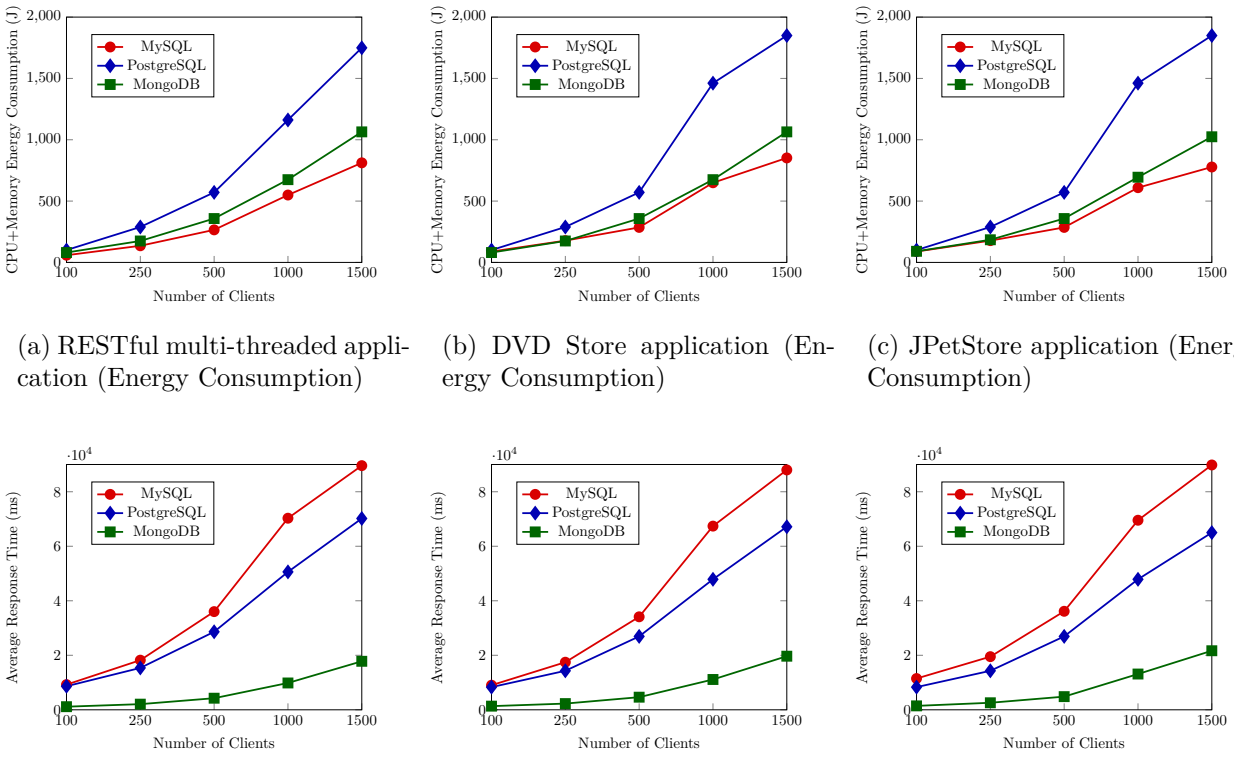
Table 5.1 Energy Consumption p-value and Cliff's δ

Pattern	MySQL	PostgreSQL	p-value	Cliff's δ	MySQL	MongoDB	p-value	Cliff's δ	PostgreSQL	MongoDB	p-value	Cliff's δ
P0	262.5	568.2	0.01	medium	262.5	354.7	0.24	small	568.2	354.7	0.09	small
P1	490.2	1391.1	$< 10e{-6}$	large	490.2	890.0	$< 10e{-6}$	large	1391.1	890.0	0.09	small
P2	495.2	1529.9	$< 10e{-6}$	large	495.2	915.9	$< 10e{-6}$	large	1529.9	915.9	0.04	medium
P3	495.0	1476.5	$< 10e{-6}$	large	495.0	904.5	$< 10e{-6}$	large	1476.5	904.5	0.04	medium
P4	1331.9	6330.2	$< 10e{-6}$	large	1331.9	5826.4	$< 10e{-6}$	large	6330.2	5826.4	0.23	small
P5	611.6	4245.1	$< 10e{-6}$	large	611.6	3821.8	$< 10e{-6}$	large	4245.1	3821.8	0.23	small
P6	824.1	4929.4	$< 10e{-6}$	large	824.1	4194.4	$< 10e{-6}$	large	4929.4	4194.4	0.23	small
P1+P7	442.7	1379.8	$< 10e{-6}$	large	442.7	814.3	$< 10e{-6}$	large	1379.8	814.3	0.03	medium
P2+P7	468.8	1482.5	$< 10e{-6}$	large	468.8	891.9	$< 10e{-6}$	large	1482.5	891.9	0.03	medium
P3+P7	490.2	1391.1	$< 10e{-6}$	large	490.2	890.0	$< 10e{-6}$	large	1391.1	890.0	0.09	small
P4+P7	1255.5	5777.4	$< 10e{-6}$	large	1255.5	5622.9	$< 10e{-6}$	large	5777.4	5622.9	0.82	negligible
P5+P7	492.2	3884.5	$< 10e{-6}$	large	492.2	3386.6	$< 10e{-6}$	large	3884.5	3386.6	0.23	small
P6+P7	775.9	4526.8	$< 10e{-6}$	large	775.9	4127.4	$< 10e{-6}$	large	4526.8	4127.4	0.23	small

Table 5.2 Response Time p-value and Cliff's δ

Pattern	MySQL	PostgreSQL	p-value	Cliff's δ	MySQL	MongoDB	p-value	Cliff's δ	PostgreSQL	MongoDB	p-value	Cliff's δ
P0	36018.6	28615.7	0.09	small	36018.6	4253.8	$< 10e{-6}$	large	28615.7	4253.8	$< 10e{-6}$	large
P1	30430.0	27867.8	0.23	small	30430.0	3639.8	$< 10e{-6}$	large	27867.8	3639.8	$< 10e{-6}$	large
P2	29504.1	27036.5	0.23	small	29504.1	3214.2	$< 10e{-6}$	large	27036.5	3214.2	$< 10e{-6}$	large
P3	29825.2	26129.6	0.23	small	29825.2	3275.0	$< 10e{-6}$	large	26129.6	3275.0	$< 10e{-6}$	large
P4	170693.1	138026.6	0.09	small	170693.1	26259.5	$< 10e{-6}$	large	138026.6	26259.5	$< 10e{-6}$	large
P5	165250.7	145382.6	0.09	small	165250.7	27897.8	$< 10e{-6}$	large	145382.6	27897.8	$< 10e{-6}$	large
P6	168786.5	130585.0	0.09	small	168786.5	24680.3	$< 10e{-6}$	large	130585.0	24680.3	$< 10e{-6}$	large
P1+P7	27826.2	22299.8	0.48	negligible	27826.2	3747.1	$< 10e{-6}$	large	22299.8	3747.1	$< 10e{-6}$	large
P2+P7	26703.4	25706.8	0.48	negligible	26703.4	3127.5	$< 10e{-6}$	large	25706.8	3127.5	$< 10e{-6}$	large
P3+P7	29339.7	23153.6	0.23	small	29339.7	4210.2	$< 10e{-6}$	large	23153.6	4210.2	$< 10e{-6}$	large
P4+P7	37584.7	29287.7	0.23	small	37584.7	2716.3	$< 10e{-6}$	large	29287.7	2716.3	$< 10e{-6}$	large
P5+P7	38153.7	26445.6	0.09	small	38153.7	2869.7	$< 10e{-6}$	large	26445.6	2869.7	$< 10e{-6}$	large
P6+P7	34183.0	27507.3	0.23	small	34183.0	20609.3	0.03	medium	27507.3	20609.3	0.09	small

Average Amount of Consumed Energy: Results presented in Table 5.1 and in Figure 5.1 show that, without using any pattern (in other words, by applying the design P0),



(a) RESTful multi-threaded application (Energy Consumption)

(b) DVD Store application (Energy Consumption)

(c) JPetStore application (Energy Consumption)

(d) RESTful multi-threaded application (Response Time)

(e) DVD Store application (Response Time)

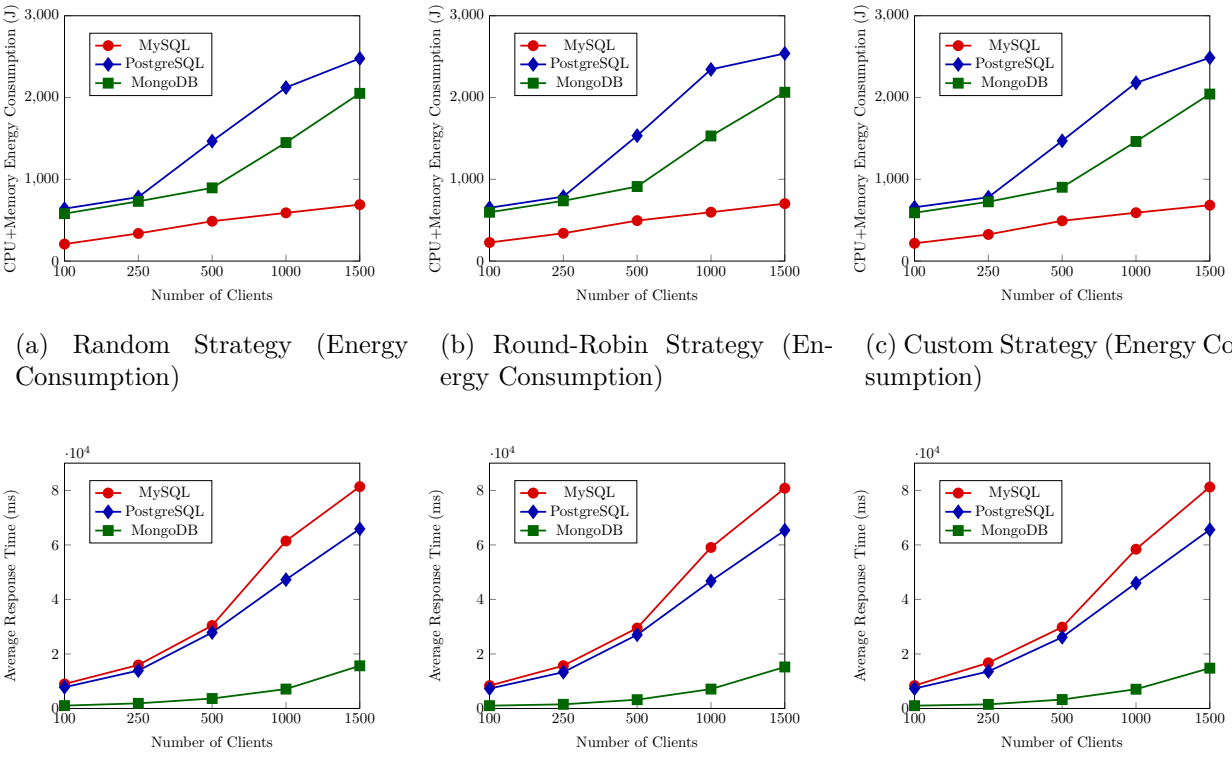
(f) JPetStore application (Response Time)

Figure 5.1 Results obtained without implementing cloud patterns

there is a statistically significant difference between the average amount of energy consumed by application using MySQL and application using PostgreSQL. The effect size in this case is medium. Therefore, we reject H^1_{0yz} for D_y, D_z (y=1, z=2). However, there is not a statistically significant difference between the average amount of energy consumed by application using MySQL and application using MongoDB. Therefore, we cannot reject H^1_{0yz} for D_y, D_z (y=1, z=3). Similarly, there is not a statistically significant difference between the average amount of energy consumed by application using PostgreSQL database and application using MongoDB database. In these two cases the effect size is small. Therefore, we cannot reject H^1_{0yz} for D_y, D_z (y=2, z=3).

Average Response Time: Results presented in Table 5.2 and in Figure 5.1 show that, by applying the design P0, there is not a statistically significant difference between the average response time of application using MySQL database and application using PostgreSQL database. Therefore, we cannot reject H^2_{0yz} for D_y, D_z (y=1, z=2). However, there is a statistically significant difference between the average response time of application using MySQL database and application using MongoDB database. Similarly, there is a statistically signifi-

cant difference between the average response time of application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^2_{0yz} for D_y, D_z ((y=1, z=3), (y=2, z=3)).



(a) Random Strategy (Energy Consumption)

(b) Round-Robin Strategy (Energy Consumption)

(c) Custom Strategy (Energy Consumption)

(d) Random Strategy (Response Time)

(e) Round-Robin Strategy (Response Time)

(f) Custom Strategy (Response Time)

Figure 5.2 Results obtained with the Local Database Proxy pattern

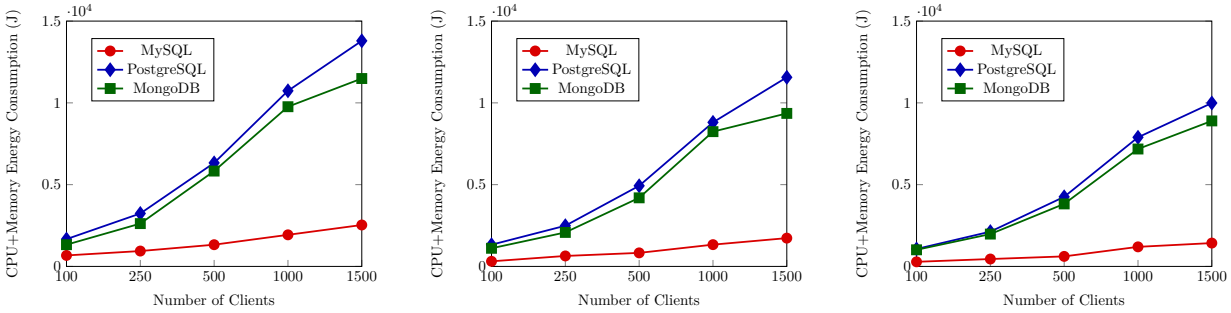
5.1.2 RQ2: Does the implementation of Local Database Proxy, Local Sharding Based Router, and Priority Message Queue patterns affect the energy consumption of cloud applications using MySQL, PostgreSQL, and MongoDB Databases?

We now report on the results and answers to RQ2.

Average Amount of Consumed Energy: As presented in Table 5.1, our results show that by applying the Local Database Proxy pattern (see Figure 5.2), there is a statistically significant difference between the average amount of energy consumed by application using MySQL database and application using PostgreSQL database. Similarly, also, between application using MySQL and application using MongoDB. Similarly also by application using Post-

greSQL database and application using MongoDB database (where the effect size is large). But, except for the case where the proxy pattern is implemented using the random strategy, there is not a statistically significant difference between application using PostgreSQL database and application using MongoDB database. Therefore we reject H^1_{xyz} for P_x, D_y, D_z (x $\in$ {2, 3}, (y=1, z=2), (y=1, z=3)), but we cannot reject H^1_{xyz} for P_x, D_y, D_z (x=1, y=2, z=3).

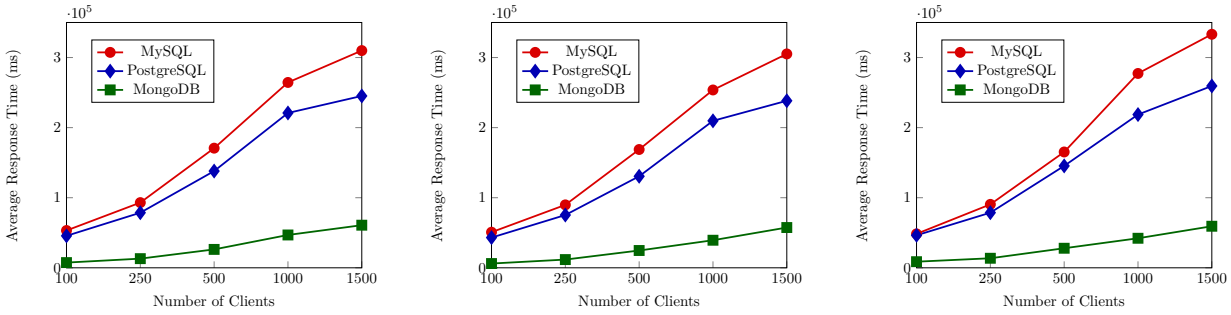
When applying the Local Sharding-Based Router (see Figure 5.3), there is a statistically significant difference between the average amount of energy consumed by application using MySQL database and application using PostgreSQL database. Similarly also between application using MySQL and application using MongoDB (the effect size is large). But, there is not a significant difference between application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^1_{xyz} for P_x, D_y, D_z (x $\in$ {4, 5, 6}, (y=1, z=2), (y=1, z=3)), but we cannot reject H^1_{xyz} for P_x, D_y, D_z (x $\in$ {4, 5, 6}, y=2, z=3).



(a) Modulo Strategy (Energy Consumption)

(b) LookUp Strategy (Energy Consumption)

(c) Consistent Strategy (Energy Consumption)



(d) Modulo Strategy (Response Time)

(e) LookUp Strategy (Response Time)

(f) Consistent Strategy (Response Time)

Figure 5.3 Results obtained with the Local Sharding-Based Router pattern

Average Response Time: As indicated in Table 5.2 and in Figure 5.2, results show that by applying the Local Database Proxy pattern, there is not a statistically significant difference

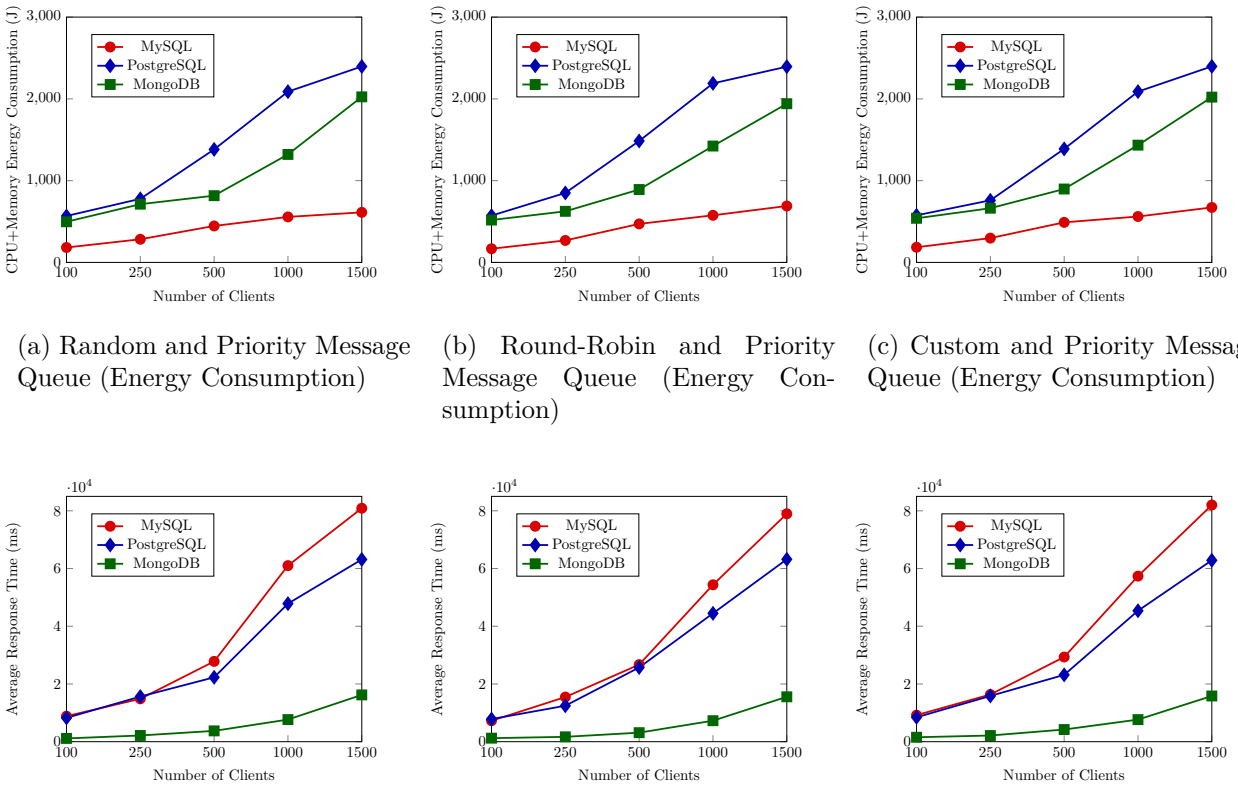
between the average response time of application using MySQL database and application using PostgreSQL database. Therefore, we cannot reject H^2_{xyz} for P_x, D_y, D_z (x $\in$ {1, 2, 3}, (y=1, z=2)). However, there is a statistically significant difference between the average response time of application using MySQL database and application using MongoDB database. Similarly, there is a statistically significant difference between the average response time of application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^2_{xyz} for P_x, D_y, D_z (x $\in$ {1, 2, 3}, (y=1, z=3), (y=2, z=3)).

Further results, when applying the Local Sharding-Based Router (see Figure 5.3), there is not a statistically significant difference between the average response time of application using MySQL database and application using PostgreSQL database. Therefore, we cannot reject H^2_{xyz} for P_x, D_y, D_z (x $\in$ {4, 5, 6}, (y=1, z=2)). However, there is a statistically significant difference between the average response time of application using MySQL database and application using MongoDB database. Similarly, there is a statistically significant difference between the average response time of application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^2_{xyz} for P_x, D_y, D_z (x $\in$ {4, 5, 6}, (y=1, z=3), (y=2, z=3)).

5.1.3 RQ3: Do the interactions between Local Database Proxy, Local Sharding Based Router, and Priority Message Queue patterns affect the energy consumption of cloud applications using MySQL, PostgreSQL, and MongoDB databases?

We now combine the Local Database Proxy pattern with the priority Message Queue pattern and also the Local Sharding-Based Router pattern with Priority Message Queue pattern.

Average Amount of Consumed Energy: When we combine the Local Database Proxy pattern with the priority Message Queue pattern (see Figure 5.4), results show that there is a statistically significant difference between the average amount of energy consumed by application using MySQL database and application using PostgreSQL database. Similarly also between application using MySQL and application using MongoDB (the effect size is large). The same is true for application using PostgreSQL database and application using MongoDB database (where the effect size is large). However, except applying the combination of the custom strategy with the Priority Message Queue pattern, there is not a statistically significant difference between application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^1_{xyz7} for P_x, D_y, D_z (x $\in$ {1, 2, 3}, (y=1, z=2), (y=1, z=3)), but we cannot reject H^1_{xyz7} for P_x, D_y, D_z (x = 3, y=2, z=3)



(a) Random and Priority Message Queue (Energy Consumption)

(b) Round-Robin and Priority Message Queue (Energy Consumption)

(c) Custom and Priority Message Queue (Energy Consumption)

(d) Random and Priority Message Queue (Response Time)

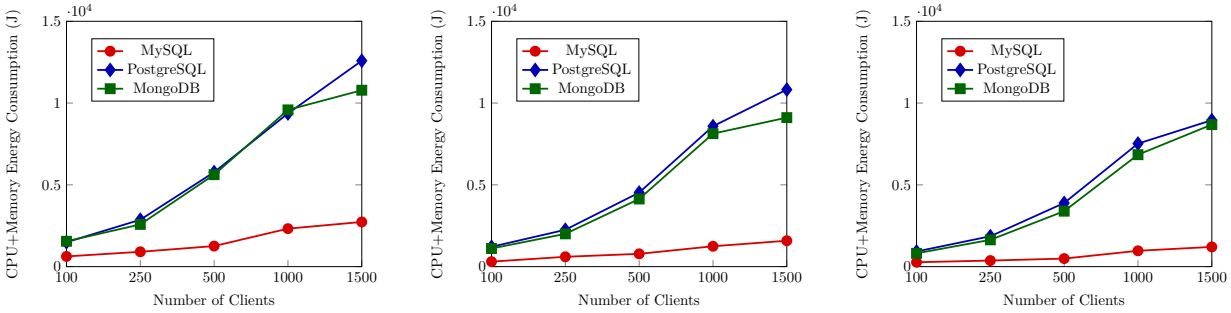
(e) Round-Robin and Priority Message Queue (Response Time)

(f) Custom and Priority Message Queue (Response Time)

Figure 5.4 Results obtained for the combination of Proxy pattern with the Message Queue pattern

Also, when we combine the Local Sharding-Based Router pattern with the priority Message Queue pattern (see Figure 5.5), results show that there is not a statistically significant difference between the average response time of application using MySQL database and application using PostgreSQL database. Therefore, we cannot reject H^2_{xyz7} for P_x, D_y, D_z (x $\in$ {4, 5, 6}, (y=1, z=2)). However, there is a statistically significant difference between the average response time of application using MySQL database and application using MongoDB database. Similarly, there is a statistically significant difference between the average response time of application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^2_{xyz7} for P_x, D_y, D_z (x $\in$ {1, 2, 3}, (y=1, z=3), (y=2, z=3)).

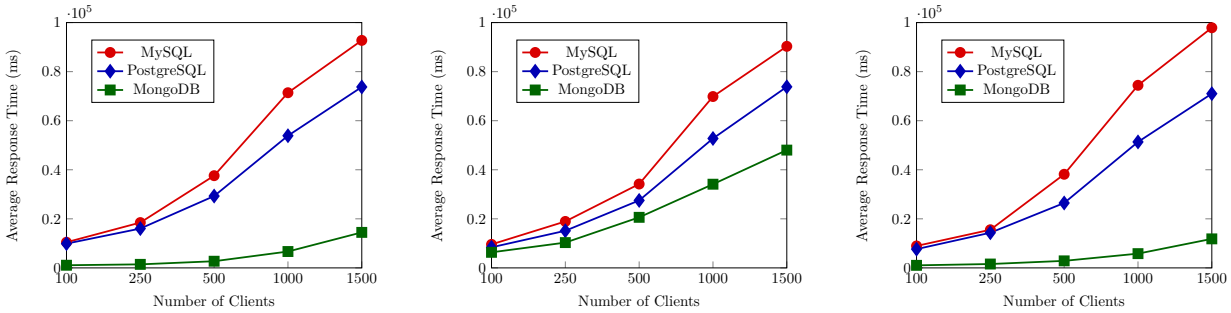
Average Response Time: When applying the Local Database Proxy pattern with the priority Message Queue pattern (see Figure 5.4), there is not a statistically significant difference between the average response time of application using MySQL database and application using PostgreSQL database. Therefore, we cannot reject H^2_{xyz7} for P_x, D_y, D_z (x $\in$ {1,



(a) Modulo and Priority Message Queue (Energy Consumption)

(b) LookUp and Priority Message Queue (Energy Consumption)

(c) Consistent and Priority Message Queue (Energy Consumption)



(d) Modulo and Priority Message Queue (Response Time)

(e) LookUp and Priority Message Queue (Response Time)

(f) Consistent and Priority Message Queue (Response Time)

Figure 5.5 Results obtained for the combination of Sharding pattern with the Message Queue pattern

2, 3}, (y=1, z=2)). However, there is a statistically significant difference between the average response time of application using MySQL database and application using MongoDB database. Similarly, there is a statistically significant difference between the average response time of application using PostgreSQL database and application using MongoDB database. Therefore, we reject H^2_{xyz7} for P_x, D_y, D_z (x $\in$ {1, 2, 3}, (y=1, z=3), (y=2, z=3)).

Besides that, when we combine the Local Sharding Based Router pattern with the priority Message Queue pattern (see Figure 5.5), results show that there is not a statistically significant difference between the average response time of application using MySQL database and application using PostgreSQL database. Therefore, we cannot reject H^2_{xyz7} for P_x, D_y, D_z (x $\in$ {4,5,6}, (y=1, z=2)). However, there is a statistically significant difference between the average response time of application using MySQL database and application using MongoDB database. As for the combination of the Lookup strategy and the Priority Message Queue pattern there is not a significant difference. Similarly, there is a statistically significant dif-

ference between the average response time of application using PostgreSQL database and application using MongoDB database. Regarding the combination of the Lookup strategy and the Priority Message Queue pattern there is not a significant difference. Therefore, we reject H^2_{xyz7} for P_x, D_y, D_z (x $\in \{4, 5\}$, (y=1, z=3), (y=2, z=3)), and we cannot reject H^2_{xyz7} for P_x, D_y, D_z (x = 6, (y=1, z=3), (y=2, z=3)).

5.2 Discussion

We showed that MySQL database is the least energy consuming but is the slowest among the three databases. PostgreSQL is the most energy consuming among the three databases, but is faster than MySQL but slower than MongoDB. MongoDB consumes more energy than MySQL but less than PostgreSQL and is the fastest among the three databases.

We explain these results by the fact that PostgreSQL database generates multiple parallel processes to run the requests sent by the RESTful cloud-based application, while MySQL and MongoDB generate only one process at a time to handle requests sent by the cloud-based application. We attribute the high energy consumed by PostgreSQL to these multiple processes.

As mentioned in Section 2, the two relational databases MySQL and PostgreSQL follow the ACID model while the MongoDB database follow the BASE (Basically Available, Soft state, Eventual consistency) model. Based on this aspect, we believe that the NoSQL database studied (i.e., MongoDB) is faster than the other two relational databases because the requests processed by relational databases must be executed one by one and can not be executed in a Simultaneous way. This aspect is similar to the phenomenon of mutual exclusion used in the treatment process.

5.3 Threats to validity

Our experiments, as any other experiment, are subject to threats to their validity. This section discusses these threats following the guidelines provided by Wohlin et al. [69].

Construct validity threats concern the relation between theory and observations. In this study, the construct validity threats are mainly due to measurement errors. As shown in 4.1, the precision of our energy measurement approach is likely to affect our findings. These measurements are subject to perturbations depending of hardware and network. To lessen these perturbations that could be caused by the network or the hardware of our private cloud environment, we did several experiments. We conducted each experiment (i.e., for each

number of clients) five times. After that, we computed average values of these measurements.

Internal validity threats concern our choice of tools and applications. Despite that fact that we studied three different databases, three cloud patterns, and three cloud-based applications, some of our findings may still be specific to our studied applications. Our results could also be impacted by our choice of Power-API as the energy measurement tool. Different tools and applications could yield different results. Therefore, future studies should consider using different relational and NoSQL databases, other cloud-based applications implementing the cloud patterns, and also another tool, with high accuracy, to measure the energy consumption of a cloud-based application.

External validity threats concern the possibility to generalize our findings. Further validation should be done on different cloud-based applications and with different relational and NoSQL databases. Applying different cloud patterns to these databases can extend our understanding of the impact of databases on the energy consumption of cloud applications, providing software engineers with guidelines about the usage of relational and NoSQL databases when developing cloud-based applications.

Reliability validity threats concern the possibility of replicating this study. As shown in chapter 4, we describe our private cloud environment, highlight the cloud-based applications, the databases and the cloud patterns used in our study for the purpose of providing all the necessary details to replicate our study.

Finally, the *conclusion validity* threats refer to the relation between the treatment and the outcome. In our study, we used carefully the statistical tests (used in chapter 4). The choice of using non-parametric tests was not arbitrary. We used them becasue they do not require a normal distribution.

CHAPTER 6 CONCLUSION

This chapter summarises the findings of this thesis, discusses the major limitations of our approach, and finally highlights some future directions of research.

6.1 Summary

Nowadays, reducing energy consumption is a challenge for cloud-based applications. We contrasted the performance of various combinations of databases and cloud patterns in terms of energy consumption and response time of the cloud-Based applications. We carried on a series of experiments on different versions of a RESTful multi-threaded application implemented with three different databases and three different cloud patterns: PostgreSQL, MySQL, and MongoDB and Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue. We also used two standard cloud applications (DVD Store application and JPetStore application) whose code we refactored to support MySQL, PostgreSQL, and MongoDB databases to validate our results and have a comprehensive view of the impact of the three studied databases on different cloud-based applications.

We studied, at first, the impact on energy consumption of three different databases: MySQL and PostgreSQL, two relational databases, and MongoDB, a NoSQL database. Then, we evaluated the impact of three cloud patterns on the energy consumption of these databases, with the aim to provide some guidance to software engineers about the usage of databases and cloud patterns for cloud-based applications.

We showed that MySQL database is the least energy consuming but is the slowest among the three databases. PostgreSQL is the most energy consuming among the three databases, but is faster than MySQL but slower than MongoDB. MongoDB consumes more energy than MySQL but less than PostgreSQL and is the fastest among the three databases.

In addition, we showed that the implementation of the Local Database Proxy pattern does not impact the behavior of the databases but can significantly improve the energy efficiency of MySQL. Concerning the Local Sharding-Based Router pattern, the Modulo strategy has a strong effect on the energy consumption of PostgreSQL and MongoDB databases but a small one for MySQL. Moreover, the Consistent strategy has a strong effect on the energy consumption of PostgreSQL but improves slightly the energy efficiency of MySQL and MongoDB. The LookUp strategy can significantly improve the energy efficiency of PostgreSQL and MongoDB.

Besides that, we showed that combining Local Database Proxy pattern with the Priority Message Queue pattern has no significant impact neither on the application response time nor on the energy consumed by the application, when it interacts with MySQL. This combination only has a small effect on the energy consumption of PostgreSQL and MongoDB. Interestingly, the implementation of the Local Sharding Based Router pattern with the Priority Message Queue pattern has a strong effect on the response time of the three Databases but without a significant impact on the energy consumption.

6.2 Limitations of the proposed approaches

Our work, like any other work, has limitations. In our study, we have selected the Power-API profiler from the literature, for its high accuracy, to estimate the energy efficiency of the cloud-based applications at the process-level. However, since Power-API is not 100% accurate, more studies should be conducted with possibly more accurate tools to verify our findings.

Also, we cannot generalize our findings, since they may still be specific to our studied applications, which were designed specifically for the experiments. Future works should replicate this study on other cloud based applications.

6.3 Future work

This thesis reports the results of a large empirical study aimed at understanding the impact of databases and their conjunction with cloud patterns on the energy efficiency of cloud applications. The results of our study could provide guidelines for cloud architects and developers.

In the future, we plan to expand our study to different NoSQL databases like HBase, Cassandra, HANA [1], because we believe that the type of database can be an important variable that should be tested for.

In addition, we plan to investigate the energy impact of data modeling strategies like denormalization and data duplication. We also plan examine how a match/mismatch between the selected database and the workload characteristic affects energy efficiency.

[1] https://en.wikipedia.org/wiki/SAP_HANA

REFERENCES

[1] J. Han, M. Song, and J. Song, "A novel solution of distributed memory nosql database for cloud computing," in *Computer and Information Science (ICIS), 2011 IEEE/ACIS 10th International Conference on.* IEEE, 2011, pp. 351–355.

[2] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck, "An architectural pattern language of cloud-based applications," in *Proceedings of the 18th Conference on Pattern Languages of Programs.* ACM, 2011, p. 2.

[3] K. Beck and W. Cunningham, "Using pattern languages for object-oriented programs," 1987.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: elements of," 1994.

[5] D. S. Linthicum, *Cloud computing and SOA convergence in your enterprise: a step-by-step guide.* Pearson Education, 2009.

[6] G. Hecht, B. Jose-Scheidt, C. De Figueiredo, N. Moha, and F. Khomh, "An empirical study of the impact of cloud patterns on quality of service (qos)," in *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on.* IEEE, 2014, pp. 278–283.

[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing.* ACM, 2010, pp. 143–154.

[8] A. Bourdon, A. Noureddine, R. Rouvoy, and L. Seinturier, "Powerapi: A software library to monitor the energy consumed at the processlevel," *ERCIM News*, vol. 2013, no. 92, 2013.

[9] C. Data, *An introduction to database systems.* Addison-Wesley publ., 1975.

[10] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in *Proceedings of the 4th ACM workshop on Scientific cloud computing.* ACM, 2013, pp. 13–20.

[11] S. Kaur and D. Kumar, "The implementation of column-oriented database in postgresql for improving performance of queries," *International Journal of Research*, vol. 3, no. 4, pp. 283–302, 2016.

[12] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High performance MySQL: Optimization, backups, and replication*. " O'Reilly Media, Inc.", 2012.

[13] Z. Wei-ping, L. Ming-Xin, and C. Huan, "Using mongodb to implement textbook management system instead of mysql," in *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. IEEE, 2011, pp. 303–305.

[14] S. W. Dietrich, D. Goelman, C. M. Borror, and S. M. Crook, "An animated introduction to relational databases for many majors," *Education, IEEE Transactions on*, vol. 58, no. 2, pp. 81–89, 2015.

[15] D. Maier, *The theory of relational databases*. Computer science press Rockville, 1983, vol. 11.

[16] H. Garcia-Molina, *Database systems: the complete book*. Pearson Education India, 2008.

[17] A. MySQL, "Mysql," 2001.

[18] B. Momjian, *PostgreSQL: introduction and concepts*. Addison-Wesley New York, 2001, vol. 192.

[19] T. Conrad, "Postgresql vs. mysql vs. commercial databases: It's all about what you need," 2006.

[20] P. DuBois, *MySQL (Developer's Library)*. Sams, 2005.

[21] A. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *arXiv preprint arXiv:1307.0191*, 2013.

[22] S. Gilbert and N. A. Lynch, "Perspectives on the cap theorem." Institute of Electrical and Electronics Engineers, 2012.

[23] P. Membrey, E. Plugge, and D. Hawkins, *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Apress, 2011.

[24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[25] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[26] S. Strauch, V. Andrikopoulos, U. Breitenbuecher, O. Kopp, and F. Leyrnann, "Non-functional data layer patterns for cloud applications," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 2012, pp. 601–605.

[27] C. H. Costa, J. V. B. Filho, P. H. M. Maia, and F. Carlos, "Sharding by hash partitioning."

[28] A. Homer, J. Sharp, L. Brader, M. Narumoto, and T. Swanson, *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Microsoft patterns & practices, 2014.

[29] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "A preliminary study of the impact of software engineering on greenit," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. IEEE, 2012, pp. 21–27.

[30] ——, "Runtime monitoring of software energy hotspots," in *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*. IEEE, 2012, pp. 160–169.

[31] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester, "Overall ict footprint and green communication technologies," in *4th International Symposium on Communications, Control and Signal Processing (ISCCSP 2010)*. IEEE, 2010.

[32] S. A. Abtahizadeh, F. Khomh *et al.*, "How green are cloud patterns?" in *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*. IEEE, 2015, pp. 1–8.

[33] C. Seo, S. Malek, and N. Medvidovic, "An energy consumption framework for distributed java-based systems," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, pp. 421–424.

[34] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 3, pp. 42–49, 2013.

[35] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 2, pp. 26–31, 2008.

[36] A. E. Trefethen and J. Thiyagalingam, "Energy-aware software: Challenges, opportunities and strategies," *Journal of Computational Science*, vol. 4, no. 6, pp. 444–449, 2013.

[37] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 29–42.

[38] N. Amsel, Z. Ibrahim, A. Malik, and B. Tomlinson, "Toward sustainable software engineering (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 976–979.

[39] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*. IEEE, 1999, pp. 2–10.

[40] M. Goraczko, A. Kansal, J. Liu, and F. Zhao, "Joulemeter: Computational energy measurement and optimization," 2011.

[41] J. Reich, M. Goraczko, A. Kansal, and J. Padhye, "Sleepless in seattle no longer." in *USENIX Annual Technical Conference*, 2010.

[42] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 22–31.

[43] I. Manotas, C. Sahin, J. Clause, L. Pollock, and K. Winbladh, "Investigating the impacts of web servers on web application energy usage," in *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*. IEEE, 2013, pp. 16–23.

[44] E. Capra, C. Francalanci, and S. A. Slaughter, "Measuring application software energy efficiency," *IT Professional Magazine*, vol. 14, no. 2, p. 54, 2012.

[45] C. Bunse, H. Höpfner, S. Roychoudhury, and E. Mansour, "Choosing the" best" sorting algorithm for optimal energy consumption." in *ICSOFT (2)*, 2009, pp. 199–206.

[46] S. Arunagiri, V. J. Jordan, P. J. Teller, J. C. Deroba, D. R. Shires, S. J. Park, and L. H. Nguyen, "Stereo matching: Performance study of two global algorithms," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2011, pp. 80 211Z–80 211Z.

[47] C. Sahin, F. Cayci, I. L. M. Gutiérrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. IEEE, 2012, pp. 55–61.

[48] C. Bunse, Z. Schwedenschanze, and S. Stiemer, "On the energy consumption of design patterns," in *Proceedings of the 2nd Workshop EASED@ BUIS Energy Aware Software-Engineering and Development*. Citeseer, 2013, pp. 7–8.

[49] C. Sahin, L. Pollock, and J. Clause, "How do code refactorings affect energy usage?" in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, p. 36.

[50] G. Procaccianti, P. Lago, W. Diesveld *et al.*, "Energy efficiency of orm approaches: an empirical evaluation," 2016.

[51] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th international conference on Mobile systems, applications, and services*. ACM, 2008, pp. 239–252.

[52] A. Merlo, M. Migliardi, and P. Fontanelli, "Measuring and estimating power consumption in android to support energy-based intrusion detection," *Journal of Computer Security*, vol. 23, no. 5, pp. 611–637, 2015.

[53] J. Hoffmann, S. Neumann, and T. Holz, "Mobile malware detection based on energy fingerprints—a dead end?" in *Research in Attacks, Intrusions, and Defenses*. Springer, 2013, pp. 348–368.

[54] A. Merlo, M. Migliardi, and P. Fontanelli, "On energy-based profiling of malware in android," in *High Performance Computing & Simulation (HPCS), 2014 International Conference on*. IEEE, 2014, pp. 535–542.

[55] F. Palmieri, S. Ricciardi, U. Fiore, M. Ficco, and A. Castiglione, "Energy-oriented denial of service attacks: an emerging menace for large cloud infrastructures," *The Journal of Supercomputing*, vol. 71, no. 5, pp. 1620–1641, 2015.

[56] C. A. Ardagna, E. Damiani, F. Frati, D. Rebeccani, and M. Ughetti, "Scalability patterns for platform-as-a-service," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 718–725.

[57] R. Aghi, S. Mehta, R. Chauhan, S. Chaudhary, and N. Bohra, "A comprehensive comparison of sql and mongodb databases," 2015.

[58] D. Hammes, H. Medero, and H. Mitchell, "Comparison of nosql and sql databases in the cloud," *Proceedings of the Southern Association for Information Systems (SAIS), Macon, GA*, pp. 21–22, 2014.

[59] R. Aghi, S. Mehta, R. Chauhan, S. Chaudhary, and N. Bohra, "A comprehensive comparison of sql and mongodb databases," *International Journal of Scientific and Research Publications*, vol. 5, no. 2, 2015.

[60] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, "On the elasticity of nosql databases over cloud management platforms," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2385–2388.

[61] T. Dory, B. Mejías, P. Roy, and N.-L. Tran, "Measuring elasticity for cloud databases," in *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. Citeseer, 2011.

[62] A. MySQL, *MySQL Administrator_s Guide and Language Reference*. Sams Publishing, 2006.

[63] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora, "An industrial case study on the automated detection of performance regressions in heterogeneous environments," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 159–168.

[64] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[65] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[66] J. Cohen, *Statistical power analysis for the behavioral sciences (rev.* Lawrence Erlbaum Associates, Inc, 1977.

[67] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological Bulletin*, vol. 114, no. 3, p. 494, 1993.

[68] J. Cohen, "A power primer." *Psychological bulletin*, vol. 112, no. 1, p. 155, 1992.

[69] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering.* Springer, 2012.