A Study of the Energy Consumption of Databases and Cloud Patterns

Béchir Bani[1,2], Foutse Khomh[1], and Yann-Gaël Guéhéneuc[2]

[1] SWAT Lab., Polytechnique Montréal, Québec, Canada
{bechir.bani,foutse.khomh}@polymtl.ca
[2] Ptidej Team, Polytechnique Montréal, Québec, Canada
yann-gael.gueheneuc@polymtl.ca

Abstract. Nowadays, databases have become the backbone of cloud-based applications. Cloud-based applications are used in about every industry today; from financial, retail, education, and communications, 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. Two families of databases are currently used in cloud-based applications: relational and non-relational databases. 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. In this paper, we study the energy consumption of three databases used by cloud-based applications: MySQL, PostgreSQL, and MongoDB, through 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, or 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 report that the choice of the databases can reduce the energy consumption of a cloud-based application regardless of the cloud patterns that are implemented.

Keywords: Energy Consumption, Relational Databases, NoSQL Databases, Performance, Sharding, Priority Message Queue.

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. These companies demand high performance databases when reading and writing data. In addition, they want to benefit from best practices encoded in the form of cloud patterns [?], which are general and reusable "good" solutions to recurring design problems for cloud-based applications.

Design Patterns were introduced by Beck and Cunningham [?] and Gamma et al. [?]. Since then, design patterns have been applied to all fields of software engineering, including cloud computing. These patterns were refined to take into account the specificities and requirements of the cloud.

In this paper, we evaluate the impact on energy consumption of three 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 goal, 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 these applications.

We measure energy consumption using the Power-API [?], which estimates the energy consumed by an application at the process-level.

Our results show that MySQL database is the least energy consuming among the three databases and PostgreSQL is the most energy consuming among them. MongoDB consumes more energy than MySQL but less than PostgreSQL. We also show that various combinations of patterns impact energy consumption.

The rest of the paper is structured as follows. Section **??** provides some background information describing the studied patterns and databases along with some related works. Section **??** presents the cloud-based applications used by our study and the design of our experiments. Section **??** discusses the results of our experiments. Section **??** discusses our results and possible threats to their validity. Section **??** concludes with some future works.

2 Background and Related Work

In this section, we introduce a brief description of the three cloud patterns that have been used in this study. Also, we present a description of SQL and NoSQL databases and the differences between them. We also include a discussion about the Power-API software tool.

2.1 Cloud Patterns

Local Database Proxy: The Local Database proxy pattern is characterized by data replication between master/slave databases and a proxy to route requests [?]. All write requests are handled by the master and they are replicated on its slaves while read requests are processed by the slaves. Components must use a local proxy whenever they need to retrieve or write data. The proxy has the responsibility to distribute requests between master and slaves depending of their type and workload. Slaves may be added or removed during the execution to gain elasticity.

Local Sharding Based Router: This pattern is useful when an application needs scalability both for read and write operations [?]. Sharding is a technique that consists in splitting data between multiple databases into functional groups called shards. Requests are processes by a local router to determine the suitable databases. The Sharding pattern is applicable through multiple strategies: hashing, a range of value or a specific shard key can be used to distribute data among the databases [?]. It is possible to scale the system out by adding further shards running on redundant storage nodes. Sharding reduces contentions and improves the performance of applications by balancing the workload across shards [?].

Priority Message Queue: This pattern which implements a First In First Out (FIFO) queue is especially used to allow asynchronous communications between components. Priority Message Queue is recommended when there are different types of messages. Message Queues enable designing loosely coupled components and improve the scalability of applications [?]. Messages with high priority values are received and processed more quickly than those with lower priority values [?].

2.2 Relational Databases

Relational Databases as a basic definition, these are databases that have dynamic relationships between tables. Contrary to NoSQL databases, relational databases follow the ACID Transaction support [?]. In this paper, we use MySQL and PostgreSQL as two relational databases. We choose these two databases because they are the most popular relational databases in the last few years [?]. Previous works have studied the impact of relational databases with cloud applications [?], but their impact on the energy consumption of cloud applications is still unknown.

2.3 NoSQL Databases

NoSQL databases [?] are non-relational and distributed databases that enable rapid, ad-hoc organization, and analysis of high-volume, disparate data types. NoSQL databases follow the famous CAP theorem [?] that is Consistency, Availability, tolerance of Partition. NoSQL databases are categorized based on the way they store data such as document store (e.g MongoDB) and key-value stores (e.g., BigTable, Dynamo). In our study, we use MongoDB database as a NoSQL database because it's the most popular NoSQL databases available today which is widely used by eBay, IBM, Expedia, and The New York Times. Previous works have compared SQL and MongoDB databases [?], but only for the response time of these databases. 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.

2.4 Power-API

PowerAPI is a profiler that provides power information (in watts which we converted to joules in order to measure the energy) per PID (*Process Identifier*) for each system component (e.g., CPU, memory, etc.) [?]. PowerAPI uses sensors and analytical models for its energy estimation. Noureddine et al. [?] performed a test to evaluate the accuracy of Power-API profiler using `PowerSpy` [?]. The results of this experiment showed that there is only minor variations between the energy consumption measured by `PowerSpy` and the energy estimations of Power-API profiler [?], and for this reason (i.e., high accuracy), we selected Power-API profiler for our study. Besides that, according to an experiment performed by Abtahizadeh et al. [?], PowerAPI profiler does not introduce noise in its measurements.

3 Study Design

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 Queue) in these three different databases on the energy consumption. We select these databases because they were used in previous studies [?], [?], [?] and they are the most popular databases available today. We select also these three cloud patterns because they are used in previous studies [?]. We now introduce our research questions, describe the objects of our study, as well as our experimental design and analysis method.

Our research questions are:

- 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 or 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?

3.1 Objects

We choose three systems for each experiment, two applications developed in Java and one application developed by a combination between PHP and Microsoft .NET. We performed each experiment on three different systems, because one system could be intrinsically more complex to understand.

At first, for Experiment 1, we implement and deploy a multi-threaded distributed application that communicates through REST calls. We use GlassFish 4 as an application server. The application interacts with one of the three chosen databases management system. *Sakila sample database* [?] provided by MySQL is used as it contains a large number of records, making it interesting for experiments. We adapted The schema of the *Sakila database* (provided by MySQL) to PostgreSQL and MongoDB databases.

For Experiment 2 and 3, we use DVDStore and JPetStore systems. DVDStore [3] is an open source simulation of an E-commerce site, 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 with a MongoDB database. Similarly to DVD Store application, we also modified the code of JPetStore[4], Commerce web application, to implement connections to MySQL, PostgreSQL and MongoDB databases.

3.2 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 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 will execute the request. The Lookup strategy, a table with a number of slots bigger than the number of servers is used to select the instance. The consistent hashing algorithm uses hashes to select the server.

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.

On the other hand, we extend our experiments by using DVDStore and JPetStore cloud applications (but without applying any pattern). By default, those applications only support MySQL and PostreSQL databases. So, we decided to extend them to support also MongoDB database so that we can have a complete comparison.

We also perform 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, 1500). Each execution is done using different databases and different cloud patterns. We measured the

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

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

energy consumption and the response time of the database in each scenario. To get precise results, we repeated each scenario five times and we computed the average for each performance metric.

3.3 Independent Variables

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

3.4 Dependent Variables

In this study, the application response time (corresponding to select and insert requests) where it is measured in *nanoseconds* and then converted to *milliseconds* and the energy consumption measured by Power-API profiler (provided in watts, which is converted to joules (J)). are considered as dependent variables.

3.5 Hypotheses

To answer our research questions, we formulate the following null hypotheses, where P0 is the experiment consisting in comparing 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 clear comprehension regarding the trade-offs between the energy consumption and the performance of a cloud-based application measured in terms of response time, 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.

3.6 Analysis Method

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

We also computed the Cliff's δ effect size [?] because effect sizes are very important to understand the magnitude of the difference between 2 distributions. In addition, it represents the degree of overlap between two sample distributions [?]. We should mention that the selection that was not arbitrary: Cliff's δ is more reliable and robust than the Cohen's d effect size [?]. The Cliff's δ effect size value expanses from -1 to +1, and it is zero when two sample distributions are the same [?]. In all our tests, we reject the corresponding null hypothesis (i.e., there is a significant difference between the the 2 distributions) when its p-value < 0.05.

Table 1. Energy Consumption p-value and Cliff's δ

max width=

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

4 Study Results

This section presents and discusses the results of our research questions.

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

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

Table 2. Response Time p-value and Cliff's δ

max width=

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

Average Amount of Consumed Energy: Results presented in **??** show that, without using any pattern (in other words, by applying the design P0), there is a statistically significant difference between the average amount of energy consumed by application using MySQL database 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 database and application using MongoDB database. 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 **??** 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). In fact, 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 cannot reject H^2_{0yz} for D_y, D_z ((y=1, z=3), (y=2, z=3)).

4.2 RQ2: Does the implementation of Local Database Proxy, Local Sharding Based Router or 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: These results show that by applying the Local Database Proxy pattern, 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 PostgreSQL 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).

Further results, by applying the Local Sharding Based Router, 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).

Average Response Time: 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)). In fact, 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, by applying the Local Sharding Based Router, 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)). In fact, 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)).

4.3 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?

The study is extended to conduct a series of experimentation in regard to the combination of Local Database Proxy pattern with the priority Message Queue pattern and also the combination of 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, 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). Similarly also by application using PostgreSQL database and application using MongoDB database (where the effect size is large), But 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).

Also, when we combine the Local Sharding Based Router pattern with the priority Message Queue pattern, 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). In fact, there is no a 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 $\in$ {4,5,6}, (y=1, z=2), (y=1, z=3)), but we cannot reject H^1_{xyz7} for P_x, D_y, D_z (x $\in$ {4,5,6}, y=2, z=3).

Average Response Time: By applying the Local Database Proxy pattern with the priority Message Queue 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_{xyz7} for P_x, D_y, D_z (x $\in$ {1,2,3}, (y=1, z=2)). In fact, 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, 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, but except applying the combination of the Lookup strategy with the Priority Message Queue pattern, there is not a significant difference. Similarly, there is a statistically significant difference between the average response time of application using PostgreSQL database and application using MongoDB database, but except applying the combination of the Lookup strategy with 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 Discussion

This section discusses some of the key aspects and findings of our study. 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. First, we can explain this result that made the PostgreSQL database generates multiple parallel process 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. According to our interpretation, it seems that for the reason of using these multiple processes, PostgreSQL database is the most energy consuming among the three studied databases. As mentioned in **??**, the two relational databases MySQL and PostgreSQL follow the ACID theorem while the MongoDB database follow the CAP theorem. Based on this aspect, we believe that the NoSQL database studied 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.

6 Threats to validity

Our experiments, as any other experiment, are subject to threats to their validity. We now discuss these threats based on the guidelines provided by Wohlin et al. [1].

Construct validity threats concern the relation between theory and observations. In this study, they could be due to measurement errors. These measurements are subject to variation depending of hardware and network. For this reason, we did several experiments, we conducted each experiment (i.e for each number of clients) five times, and computed average values of these measurements, in order to mitigate the potential biases that could be induced by perturbations on the network or the hardware, and our tracing. Before each measurement, *Internal validity* threats concern our selection of subject systems and analysis methods. Despite of using the well known benchmark (the Sakila sample database [2]) for the implemented RESTful multi-threaded cloud-based application interacted each time with one of the three databases, the well-know patterns and strategies, and the two standard cloud applications (DVD Store, JPetStore) some of our findings may still be specific to our studied application which was designed specifically for the experiments. Future studies should consider using different RDBMS and NoSQL databases, and also other cloud standard applications implementing the cloud patterns.

External validity threats concern the possibility to generalize our findings. Further validation should be done on different cloud applications and with different relational and NoSQL databases and 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 guideline to about the usage of relational and NoSQL databases when developing cloud-based applications.

Reliability validity threats concern the possibility of replicating this study. We attempt to provide all the necessary details to replicate our study.

Finally, the *conclusion validity* threats refer to the relation between the treatment and the outcome. We paid attention not to violate the assumptions of the performed statistical tests. We mainly used non-parametric tests that do not require making assumptions about the distribution of the metrics.

7 Conclusion

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, with the aim to provide some guidance to software engineers about the usage of databases and cloud patterns for 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) to validate our results.

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.

References

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

2. “Mysql sakila sample database,” http://dev.mysql.com/doc/sakila/en/, 2014, [Online; accessed August-2015].