How Green Are Cloud Patterns?

S. Amirhossein Abtahizadeh, Foutse Khomh, Yann-Gaël Guéhéneuc
SWAT–PTIDEJ, École Polytechnique de Montréal, Canada
{a.abtahizadeh, foutse.khomh, yann-gael.gueheneuc}@polymtl.ca

***Abstract*—Cloud Patterns are abstract solutions to recurrent design problems in the cloud. Previous work has shown that these patterns can improve the Quality of Service (QoS) of cloud applications but their impact on energy consumption is still unknown. Yet, energy consumption is the biggest challenge that cloud computing systems (the backbone of today's high-tech economy) face today. In fact, 10% of the world's electricity is now being consumed by servers, laptops, tablets and smartphones. Energy consumption has complex dependencies on the hardware platform, and the multiple software layers. The hardware, its firmware, the operating system, and the various software components used by a cloud application, all contribute to determining the energy footprint. Hence, even though increasing a data center efficiency will eventually improve energy efficiency, the internal design of cloud-based applications can be improved to lower energy consumption. In this paper, we conduct an empirical study on a RESTful multi-threaded application deployed in the cloud, to investigate the individual and the combined impact of three cloud patterns (*e.g.*, Local Database proxy, Local Sharding Based Router and Priority Queue) on the energy consumption of cloud based applications. We measure the energy consumption using Power-API; an application programming interface (API) written in Java to monitor the energy consumed at the process-level. Results show that cloud patterns can effectively reduce the energy consumption of a cloud application, but not in all cases. In general, there appear to be a trade-off between an improved response time of the application and the energy consumption. Developers and software architects can make use of these results to guide their design decisions.**

***keywords*—Cloud Patterns, Energy Consumption, Energy Efficiency, Sharding, Priority Message Queue**

I. Introduction

Cloud computing systems are now pervasive in our society. As a consequence, the energy consumption of data centres and cloud based applications, has become an emerging topic in the software engineering research communities [1], [2]. Energy consumption has complex dependencies on the hardware platform, and the multiple software layers. The hardware, its firmware, the operating system, and the various software components used by a cloud application, all contribute to determining the energy footprint; making energy optimisation a very challenging problem. The role of the software components and coding practices has been recently investigated by works such as [3], [4] and researchers have proposed energy-aware algorithms [5] and sensor relocation techniques [6] to help to optimize software energy consumption. When developing an energy efficient cloud based application, developers must seek a compromise between the application's Quality of Service (QoS) and energy efficiency. Manually finding such a compromise is a daunting task, and developers should be supported by guidelines and–or tailored recommendations in the form of best practices. Cloud patterns, which are general and reusable solutions to recurring design problems, have been proposed as best practices to guide developers during the development of cloud based applications. However, although previous work [7] has shown that these cloud patterns can improve the QoS of cloud based applications, their impact on energy consumption is still unknown.

978-1-4799-7575-4/14/$31.00 ©2015 IEEE

In this paper, we evaluate the impact on energy consumption of three cloud patterns: Local Database Proxy, Local Sharding-Based Router and Priority Queue. The study is performed using a RESTful multi-threaded application deployed in the cloud. The application is implemented with different combinations of the aforementioned patterns. The energy consumption is measured using Power-API; an application programming interface (API) written in Java that can monitor the energy consumed by an application, at the process-level [8].

The rest of the paper is structured as follows. In Section II we provide some background information describing the studied patterns along with related works. Section III presents the cloud-based application scenarios and the design of our experiments and section IV discusses the obtained results. Section V explains possible threats to the validity of our research. Section VI wraps up our research and Section VII suggests some future works that might extend our study.

II. Background and Related Work

This section includes a brief presentation of the three patterns under study in this research and outlines some of their benefits for cloud applications. Energy measurements with the Power-API software library is discussed.

A. Cloud Patterns

Local Database Proxy: This pattern uses data replication between master/slave databases and a proxy to route requests [9] and provides a read scalability on a relational database. Write requests are handled by the master and replicated on its slaves, while Read requests are processed by slaves. When applying this pattern, components must use a local proxy whenever they need to retrieve or write data. The proxy distributes requests between master and slaves depending on their type and workload. Slaves may be added or removed during the execution to obtain elasticity. There could be a risk of bottleneck on the master database when there is a need to scale with write requests. This issue together with the lack of strategy for write requests are listed among the limitations of this pattern in Microsoft guidelines [10]. The

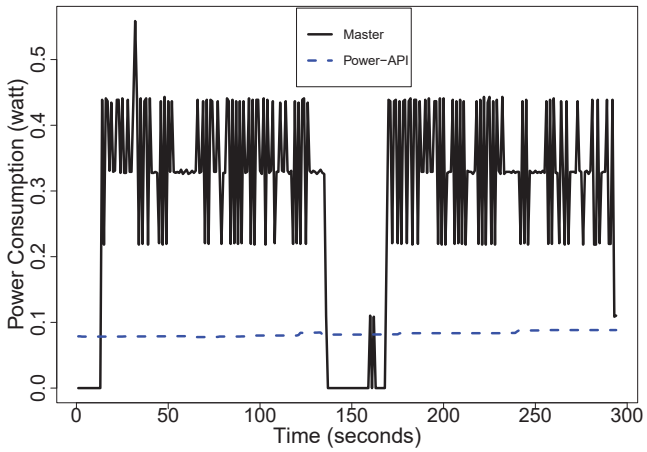


Fig. 1: Power-API distortion test

impact of this pattern on the QoS of applications has been examined by Hecht et al. [7]; however, to the best of our knowledge, no work has empirically investigated the impact of the Local Database Proxy pattern on the energy consumption of applications.

Local Sharding-Based Router: This pattern is useful when an application needs scalability both for read and write operations [9]. 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. Data are split horizontally *i.e.*, on rows, and each split must be independent as much as possible to avoid joins and to benefit from the sharding. The sharding logic is applicable through multiple strategies; a range of value, a specific shard key or hashing can be used to distribute data among the databases [11]. 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 [11]. Shards can be located close to specified clients to improve data accessibility. When combined with other patterns, Sharding can have a positive impact on the QoS of applications (specifically when experiencing heavy loads) [7]. However, the impact of Sharding on energy consumption is still unclear.

Priority Message Queue: This pattern which implements a First In First Out (FIFO) queue is typically used to delegate tasks to background processing or to allow asynchronous communications between components. Priority Message Queue is recommended when there are different types of messages. Messages with high priority values are received and processed more quickly than those with lower priority values [12]. Message Queues enable designing loosely coupled components and improve the scalability of applications [7].

Evaluation of Cloud Patterns: Ardagna *et al.* [13] empirically evaluated the performance of five scalability patterns for Platform as a service (PaaS): Single, Shared, Clustered, Multiple Shared and Multiple Clustered Platform Patterns. To compare the performance of these patterns they measured the response time and the number of transactions per second. They explored the effects of the addition and the removal of virtual resources, but did not examine the impact of the patterns on energy consumption. Tudorica *et al.* [14] and Burtica *et al.* [15] performed a comprehensive comparison and evaluation of no-SQL databases (which make use of multiple sharding and replication strategies to increase performance), but did not examine energy consumption aspects. Along the same line of work, Cattel [16] examined no-SQL and SQL data stores designed to scale by using replication and sharding. They did not perform energy consumption evaluations. However, these works highlighted the lack of studies and benchmarks on cloud patterns solutions. Message oriented middlewares have been benchmarked by Sachs *et al.* [17]; nevertheless, the energy consumption aspect was ignored.

B. Power-API

PowerAPI is a system-level library that provides power information per PID for each system component (*e.g.*, CPU, network card, etc.) [8]. The energy estimation is performed using analytical models that characterize the consumption of the components (*i.e.*, CPU, memory, disk). The accuracy of PowerAPI was evaluated using the bluetooth powermeter (PowerSpy) [18], and results revealed only minor variations between the energy estimations of Power-API, and the energy consumption measured by PowerSpy [8].
PowerAPI was selected for this work because of its reported high accuracy [8], and the aggregate energy consumption data for CPU, Memory and Disk were collected by auditing multiple corresponding VMs. Power-API measures power in watts, which was converted to the unit of energy (Kilojoules). However, prior to our study, we performed a pilot experiment to examine the risk of Power-API introducing noise in its own measurements. More specifically, we conducted a test in which 10,000 records were inserted twice with a short gap of time into our database and both the process of Power-API and the process of the database were measured. The result illustrated in Figure 1 shows that Power-API does not introduce noise in its measurements; it did not alter the energy measurements of the master server during the period of insertion.

III. Study Design

In this paper, we set out to empirically evaluate the impact of three cloud patterns (*i.e.*, Local Database Proxy, Local Sharding-Based Router and Priority Queue) on the energy consumption of cloud based applications. We select these cloud patterns because they are used in previous studies [7], [14] (allowing us to compare our results with these previous works), and because they are described as good design practices by both academic and practitioners. In this section, we introduce our research questions, describe the objects of our study, as well as our experimental design and analysis method.

(RQ1) Does the implementation of Local Database Proxy, Local Sharding-Based Router or Priority Message Queue patterns affect the energy consumption of

cloud applications?

(RQ2) Do interactions among Local Database Proxy, Local Sharding-Based Router and Priority Message Queue patterns affect the energy consumption of cloud applications?

To answer these research questions, a multi-threaded version of the application used in [7] was implemented and a series of experimentations were performed by using multiple versions of the aforementioned application. The Local Database Proxy and Local Sharding-Based router patterns are implemented using different algorithms which are explained in section III-C. In total, eight versions of the application were analyzed and summarized in Table I. To collect the energy measurements required to answer our research questions a series of scenarios were executed, designed specifically to simulate the characteristics of a real-world cloud-based application. The remainder of this section elaborates more on the details of our experimentations.

A. Objects

A multi-threaded distributed application, which communicates through REST calls was implemented and deployed on GlassFish 4 application server. We chose MySQL as the database management system because it is one of the most popular databases for Cloud applications [19]. Sakila sample database [20] provided by MySQL is used as it contains a large number of records, making it interesting for experimentations. Sakila is consistent with existing databases. The test application was fully developed with the Java Development Kit 1.7 and it is composed of about 3,800 lines of code and the size is 6 MB.

The master node has the following characteristics: 2 virtual processors (CPU: Intel Xeon X5650) with 8GB RAM and 40GB disk space. This node is a virtual machine of a server located on a separate network. We have 8 slave database nodes: 4 on one server, each one has a virtual processor (CPU: Intel QuadCore i5) with 1 GB RAM and 24 GB disk space. The other four database nodes are on a second server with the following characteristics: each Virtual Machine has one virtual processor (CPU: Intel Core 2 Duo), 1 GB RAM and 24 GB disk space. All the hardware is connected on a private network behind a switch. All the virtual machines are running on VMware ESXi and all the servers are running Ubuntu 14.04 LTS 64-bit as operating system.

B. Design

In order to evaluate the benefits and the trade-offs between the Local Database Proxy, the Local Sharding-Based Router and the Priority Message Queue design patterns, we implemented these patterns in the application described in Section III-A and examined them through the scenarios described in Section III-C. In total, 8 versions of the application were obtained and presented in Table I. The NoProxy/NoSharding version E0 does not use any pattern. Versions E1 to E3 implement Local Database Proxy with Random Allocation, Round-Robin and a Custom load balancing algorithm. Versions E4 to E6 correspond to Local Sharding-Based Router with three sharding algorithms: Modulo, Lookup and Consistent Hashing. Version E7 implements the Priority Message Queue pattern. Since there is only one implementation of Priority Message Queue pattern in this study, we investigate its impact on energy consumption only in combination with the other patterns.

TABLE I: Experimental Design

Pattern	Algorithm	Code Version
NoProxy/NoSharding	-	E0
Local Database Proxy	Random Allocation	E1
	Round-Robin	E2
	Custom Load Balancing	E3
Local Sharding-Based Router	Modulo Algorithm	E4
	Lookup Algorithm	E5
	Consistent Hashing	E6
Priority Message Queue	-	E7

C. Procedure

A scenario was designed in which the client is a thread generated on the client side of our cloud-based application. This client establishes a connection to the server then performs a series of actions in a certain amount of time (see Figure 2). Each client sends *100 select requests* at the peak of the scenario workload, and we measure the response time of the application at this point by taking the average of the response time to all clients. This will represent the performance of the application.

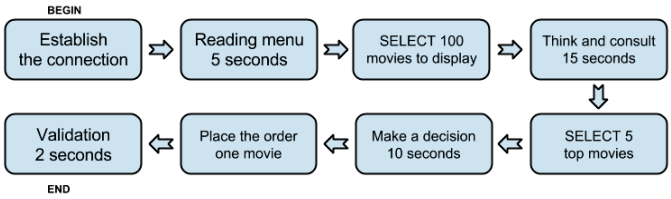


Fig. 2: Test Scenario

We repeat this scenario using different number of clients. The requests performed by the clients are: *select* or *write* to *display* or *place the order* respectively. The application generates concurrent threads to simulate clients propagating these requests. The number of clients used in our experiments are: 100, 250, 500, 1000 and 1500. Power-API is located on the database node and measured the amount of energy consumed by MySQL process. Because of the variability observed during multiple executions of an application (caused for example by optimization mechanisms like caching), we repeated each experiment five times and took the average. To ensure lower variance between maximum and average values and hence increase the precision of our energy measurements, we eliminated the values of the first and last executions. In total, our application was able to simulate a maximum of 150,000 concurrent requests, enabling us to establish a cloud-based application for our experiment capable of responding to thousands of coincidental requests from clients. This level of load is reflective of real-world cloud applications.

In our study, each experiment is independent with regard to others, and simulations were terminated at a fixed time. Even large requests and heavy loads never lasted more than this amount of time. In the following, we explain in details the specific algorithms that were implemented in each pattern.

Local Database Proxy: Three implementations of this pattern were considered in our research; Random allocation strategy, Round-Robin allocation strategy [21], and Custom Load balancing strategy. The proxy is placed between the server and the clients. The basic approach, *NoProxy* REST web service, exposes a set of methods which are hitting the database regarding different algorithms. These methods are used to test the local database proxy pattern. It is the baseline to which we compare the results of our proxy implementations. The queries are built using parameters such as the ID of a select passed over the REST call from each client (thread) concurrently during each scenario.

The random approach is implemented by choosing randomly an instance of the pool. The round-robin chooses the next instance that has not yet been used in the "round", *i.e.*, the first, then the second, then the third,..., finally the first and so on. The custom algorithm is more reactive, and it uses two metrics to evaluate the best slave node to pick: the ping response time between the server and slave, and the number of active connections on the slaves. A thread is started every 500 ms with the purpose of monitoring such metrics. After choosing the corresponding slave, the query is executed and the result is sent back to the function that was called. To simplify the tests, only IDs (number identifier) were sent back, so there was no need to serialize any data. The query is executed consequently whenever the result is null on the master node in order to make sure that the replication did not fail. Eventually, if the result is null, the response sent to the client has the http *no content* status. Otherwise, the result is sent back to the client using the http *ok response* status.

Local Sharding-Based Router: To test this pattern, we used multiple shards hosted separately. Each shard had the same database schema and structure as suggested by sharding algorithms [22]. Two tables of a modified version of the Sakila database [20] were used. All the relationships in both the "rental" and "film" tables were removed since the sharding is adapted only for independent data.

Three commonly known sharding algorithms were studied in our research: Modulo algorithm, Look-up algorithm and the Consistent Hashing algorithm. The modulo algorithm divides the request primary key by the number of running shards, the remainder is the number of the server which will handle the request. The second sharding algorithm used is the Look-up strategy. This algorithm implements an array with a larger amount of elements than the number of server nodes available. References to the server node are randomly placed in this array such that every node receives the same share of slots. To determine which node should be used, the key is divided by the number of slots and the remainder is used as index in the array. The third sharding algorithm used is the Consistent Hashing. For each request, a value is computed for each node. This value is composed of the hash of the key and the node. Then, the server with the longest hash value processes the request. The hash algorithms recommended for this sharding algorithm are MD5 and SHA-1.

Priority Message Queue: Requests are annotated with different priority numbers and sent in the priority message queue of our test application. All requests are ordered according to their priority and processed by the database services in this order.

D. Independent Variables

Local Database Proxy, Local Sharding-Based Router, and Priority Message Queue patterns, as well as the algorithms presented in Table I are the independent variables of our study.

E. Dependent Variables

The dependant variables measure the quality of service in terms of response time to select queries dispatched by the clients and the energy consumption measured by Power-API during each scenario. The result is a two-dimensional comparison between response time and the amount of energy consumed. The response time is measured in nanoseconds and then converted to milliseconds. We choose this metric because it reflects the capacity of the application to scale with the number of clients at peak with maximum number of requests.

The other metric is the power consumption provided by Power-API in watts, which is converted to kilojoules(kJ) using the equation E_{kJ} = (P_{watt} x t_s)/ 1000.

F. Hypotheses

To answer our research questions, we formulate the following null hypotheses, where E0, Ex ($x \in \{1 \dots 6\}$), and E7 are the different versions of the application described in Table I:

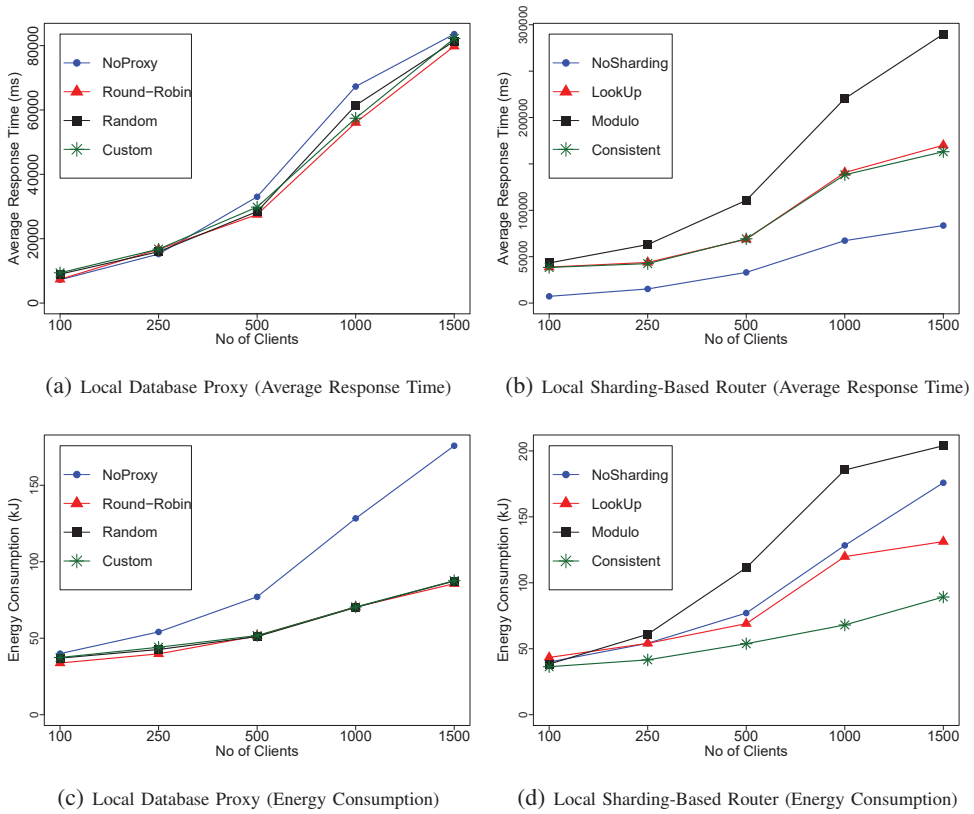
- H_x^1 : There is no difference between the average amount of energy consumed by design Ex and design E0.
- H_{x7}^1 : The average amount of energy consumed by the combination of designs Ex and E7 is not different from the average amount of energy consumed by each design taken separately.

To better understand 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_x^2 : There is no difference between the average response time by design Ex and design E0.
- H_{x7}^2 : The average response time of the combination of designs Ex and E7 is not different from the average response time of each design taken separately.

G. Analysis Method

Since the observations of each scenario were independent of those of the other scenarios, we perform the Mann-Whitney U test [23] to test H_x^1, H_x^2, H_{x7}^1, H_{x7}^2. Moreover, we computed the Cliff's δ effect size [24] to quantify the importance of the difference between the metric values. Cliff's δ effect size is reported to be more robust and reliable than the Cohen's



(a) Local Database Proxy (Average Response Time)

(b) Local Sharding-Based Router (Average Response Time)

(c) Local Database Proxy (Energy Consumption)

(d) Local Sharding-Based Router (Energy Consumption)

Fig. 3: Results obtained for individual patterns

TABLE II: p-value of Mann–Whitney U test and Cliff's δ effect size for application scenarios

	Average Response Time		Average Energy Consumption	
	p-value	Effect Size	p-value	Effect Size
E0, E1	0.87	0.04	**<0.05**	0.44
E0, E2	0.77	0.06	**<0.05**	0.49
E0, E3	0.87	0.04	**<0.05**	0.44
E0, E4	**<0.05**	-0.68	0.36	-0.2
E0, E5	**<0.05**	-0.6	0.74	0.07
E0, E6	**<0.05**	-0.6	0.05	0.42
E1, E2	0.59	0.12	0.71	0.08
E1, E3	0.59	-0.12	0.48	-0.15
E1, E4	**<0.05**	-0.76	**<0.05**	-0.52
E1, E5	**<0.05**	-0.6	**<0.05**	-0.44
E1, E6	**<0.05**	-0.6	0.87	0.04
E2, E3	0.46	-0.16	0.56	-0.12
E2, E4	**<0.05**	-0.76	**<0.05**	-0.52
E2, E5	**<0.05**	-0.6	**<0.05**	-0.44
E2, E6	**<0.05**	-0.6	0.59	-0.12
E3, E4	**<0.05**	-0.76	**<0.05**	-0.51
E3, E5	**<0.05**	-0.6	0.08	-0.36
E3, E6	**<0.05**	-0.6	0.87	0.04
E4, E5	0.18	0.28	0.36	0.2
E4, E6	0.09	0.36	**<0.05**	0.52
E5, E6	0.59	0.12	**<0.05**	0.46
E1, E1 + E7	0.38	0.19	0.53	-0.13
E2, E2 + E7	0.43	0.17	0.20	-0.28
E3, E3 + E7	0.51	0.14	0.53	-0.13
E4, E4 + E7	**<0.05**	0.68	0.2	-0.28
E5, E5 + E7	**<0.05**	0.52	0.2	-0.28
E6, E6 + E7	**<0.05**	0.49	**<0.05**	-0.6
E4, E3 + E4	**<0.05**	0.76	0.36	-0.2
E6, E2 + E6	**<0.05**	0.6	0.36	-0.2

d effect size [25]. We perform all our tests using a 95% confidence level (*i.e.*, p-value < 0.05). Mann-Whitney U test is a non-parametric statistical test that examines whether two independent distributions are the same or if one tends to have higher values. Non-parametric statistical tests make no assumptions about the distributions of the metrics. Cliff's δ is a non-parametric effect size measure which represents the degree of overlap between two sample distributions [24]. Cliff's δ effect size ranges from -1 (if all selected values in the first group are larger than the second group) to +1 (if all selected values in the first group are smaller than the second group), and it is zero when two sample distributions are identical [26]. A Cliff's δ effect size is considered negligible if it is < 0.147, small if it is < 0.33, medium if it is < 0.474, and large if it is $>= 0.474$.

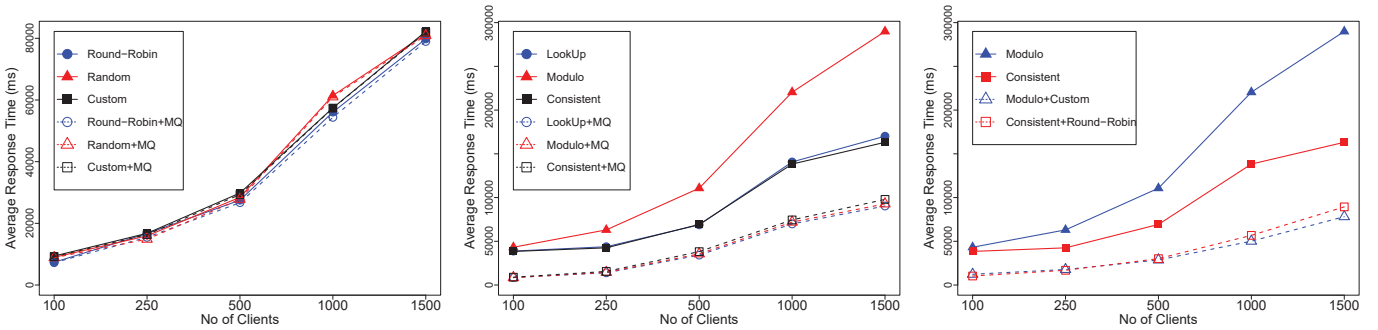
IV. Results

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

RQ1: Does the implementation of Local Database proxy, Local Sharding-Based Router or Priority Message Queue patterns affect the energy consumption of cloud applications?

Table II summarizes the results of Mann–Whitney U test and Cliff's δ effect sizes for the energy consumption and the response time. We marked significant results in bold.

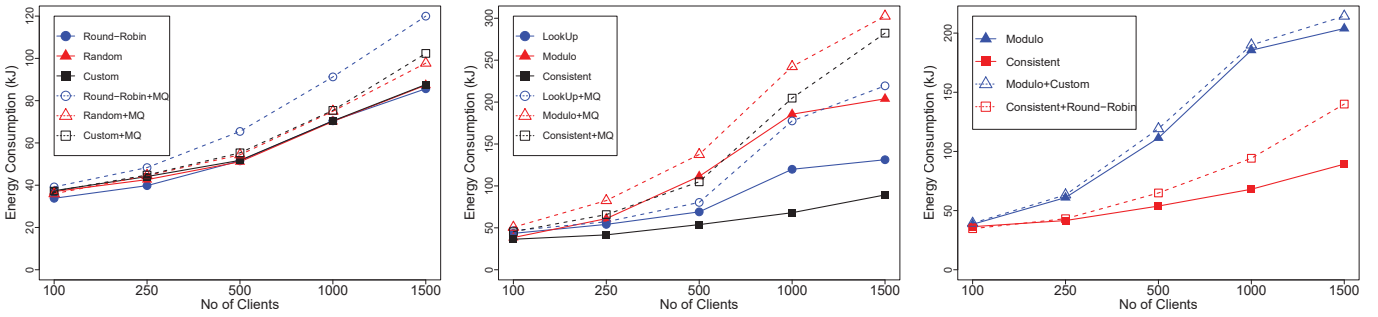
Average amount of consumed energy: Results presented in Table II show that, there is a statistically significant dif-



(a) Database Proxy and Priority Message Queue (Average Response Time)

(b) Local Sharding-Based Router and Priority Message Queue (Average Response Time)

(c) Local Sharding-Based Router and Local Database Proxy (Average Response Time)



(d) Local Database Proxy and Priority Message Queue (Energy Consumption)

(e) Local Sharding-Based Router and Priority Message Queue (Energy Consumption)

(f) Local Sharding-Based Router and Local Database Proxy (Energy Consumption)

Fig. 4: Results obtained for the combination of patterns

ference between the average amount of energy consumed by application implementing the Local Database Proxy pattern and application not implementing this pattern. The effect size is almost large in all cases. Therefore, we reject H_x^1 for all Ex ($x \in \{1 \dots 3\}$). Regarding the Local Sharding-Based Router pattern, except for the case where the pattern is implemented using the consistent hashing algorithm, the difference between the amount of energy consumed by an application implementing the pattern and another application that did not implement the pattern is not significantly different. In other words, only consistent hashing tends to consume (to some extent) less energy than no sharding strategy. However, the effect size is low. Therefore, we cannot reject H_x^1 for all Ex ($x \in \{4 \dots 6\}$).

These results show that any implementation of the Local Database Proxy pattern can significantly improve the energy efficiency of an application, while the Local Sharding-Based Router pattern has little effect on energy consumption. Figure 3 summarizes the results obtained for all the implementations of the two patterns.

Average response time: Results from Table II show that there is a statistically significant difference between the response time of applications implementing the Local Sharding-Based Router pattern and application not implementing this pattern. Hence, we reject H_x^2 for all Ex ($x \in \{4 \dots 6\}$). In fact, as shown on Figure 3b, all the implementations of the Local Sharding-Based pattern have a negative impact on the response time of the applications (*i.e.*, the average response time is increased). Among the different implementations of the Local Sharding-Based Router pattern, the Modulo algorithm has most negative impact on the response time.

Regarding the Local Database Proxy pattern, there is no statistically significant difference between the response time of application using any version of the pattern and application not using the pattern. However, Figure 3a, as well as effect size values show that Local Database Proxy pattern has a (small) positive impact on the response time of the applications. Yet, we cannot reject H_x^2 for ($x \in \{1 \dots 3\}$).

In summary, we conclude that **although the Local Database Proxy pattern only has a small positive impact on the ability of applications to handle large number of requests of *read queries*, it can significantly improve the energy energy efficiency of a cloud based application.**

Regarding the Local Sharding-Based Router pattern, **it is only when the pattern is implemented using the consistent hashing strategy that the energy efficiency is slightly improved. Moreover, all the implementations of this pattern increase the average response time of applications for heavy *read requests*. This result is consistent with previous works [9] that found the pattern to be more adequate for applications handling huge *write requests* loads.**

RQ2: Do interactions among Local Database Proxy, Local Sharding-Based Router and Priority Message Queue patterns affect the energy consumption of cloud applications?

Regarding the response time, Table II and Figure 4a show that there is no statistically significant difference between the Local Database Proxy pattern and the combination of this pattern with Priority Message Queue. It did not have a pronounced impact on the average response time of the application. Consequently, we cannot reject H^2_{x7} for all Ex ($x \in \{1 \dots 3\}$). Besides, our results have shown that Local Sharding-Based Router pattern when combined with the Priority Message Queue pattern can remarkably reduce an application's response time. Statistical tests show that there is a significant difference, regardless of the type of algorithm, between the individual sharding pattern and combined sharding with the message queue pattern (see Figure 4b). The effect size is large in all three cases in Table II. Consequently, we reject H^2_{x7} for all Ex ($x \in \{4 \dots 6\}$).

On the other hand, represented values in Table II depict that Local Database Proxy combined with Priority Message Queue does not significantly increase the amount of energy consumed in the application (see Figure 4d). Therefore, we cannot reject H^1_{x7} for all Ex ($x \in \{1 \dots 3\}$).

Further results, surprisingly, indicate that when combining Local Sharding-Based Router with the Priority Message Queue pattern, choosing only the consistent hashing algorithm can make a significant difference on the average amount of energy consumed by the application. If we combine the Priority Message Queue pattern with the Local Sharding-Based Router pattern implemented using the consistent hashing algorithm, it appears that there is more energy being consumed in our application (see Figure 4e). Hence, we reject H^1_{x7} for E_6, likewise we cannot reject H^1_{x7} for E_4 and E_5.

Combining Priority Message Queue pattern with Local Database Proxy has no significant impact neither on application response time, nor on the average amount of energy consumed by the application. On the contrary, the combination of Priority Message Queue pattern and Sharding-Based Router pattern can improve the response time of an application experiencing heavy loads of read requests. Besides, only the implementation of consistent hashing in Local Sharding-Based Router pattern can increase the energy consumption of such application dramatically.

The study is extended to conduct a series of experimentation in regard to the combination of Local Sharding-Based Router pattern with Local Database Proxy pattern. As it can be seen in Figure 4c, such combination considerably improves the applications QoS by reducing the average response time. Table II confirms our experiments results in a way that there is a significant difference between using Local Sharding-Based Router pattern individually and combining this pattern with Local Database Proxy. Conversely, statistical test in Table II shows that Local Sharding-Based Router pattern combined with Local Database Proxy pattern has no significant impact on the energy consumption of the application (Figure 4f).

V. Threats to Validity

This section briefly discusses the threats to validity of our study taking into account the guidelines provided by Wohlin *et al.* [27].

Construct validity threats concern the relation between theory and observations such as measurement errors in this study. We repeated each experimentation five times and computed average values, in order to mitigate the potential biases that could be induced by perturbations on the network or the hardware, and our tracing. Power-API tool was carefully compiled and calibrated before each run of the application. This redundant accuracy we believe increased the quality of our measurements. The first and last values were eliminated to obtain lower variance between the average and maximum.

Internal validity threats concern our selection of subject systems and analysis methods. Although we have used a well known benchmark (the Sakila sample database [20]) and well-know patterns and algorithms, 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 cloud based applications and different tools to measure the energy consumption.

External validity threats concern the possibility to generalise our findings. Further validation with different cloud patterns can extend our understanding of the impact of cloud patterns on the energy consumption of applications, providing developers with guideline about the usage of cloud patterns 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. All the data used in this study are available online [28].

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.

VI. Conclusion

Energy consumption is the biggest challenge that cloud computing systems face today. Pinto et al. [29] who analyzed more than 300 questions and 550 answers on the popular StackOverflow question-and-answer site for developers reported that the number of questions on energy consumption increased by 183% from 2012 to 2013. The majority of those questions were related to software design, showing that developers need guidance for designing green software systems. Similarly, Pang et al. [30] found that developers lack knowledge on how to develop energy-efficient software. In this paper, we examine the impact on energy consumption of three cloud patterns, with the aim to provide some guidance to developers, about the usage of cloud patterns. More specifically, we conducted a series of experiments with different versions of a cloud based RESTful application implementing the Local Database Proxy, the Local Sharding-Based Router and the Priority Queue patterns. We assessed the impact of the

patterns both on the response time and the energy consumption of the application. Results show that any implementation of the Local Database Proxy pattern can significantly improve the energy efficiency of cloud based application, while the Local Sharding-Based Router pattern only has a small effect on the energy consumption. In fact, only the consistent hashing algorithm seems to have a positive effect on the energy efficiency of applications using the Local Sharding-Based Router pattern. Overall, the Local Database proxy appears to be more adapted for applications experiencing heavy loads of *read requests*, while the Local Sharding-Based Router is not suitable for such applications, but seems more appropriate for applications handling huge *write requests* loads.

Our results have shown that combining Priority Message Queue pattern with the Local Database Proxy has no significant impact neither on the application's response time, nor on the average amount of energy consumed by the application. Local Sharding Based Router when combined with Local Database Proxy tends to improve response time significantly. Interestingly, the implementation of the custom proxy algorithm in Local Database Proxy combined with the modulo algorithm in Local Sharding-Based Router can improve the response time of an application without consuming significantly more energy.

VII. Future Work

The study presented in this paper can be extended to different relational databases and NoSQL databases [31], where multiple fine grained optimizations are performed to improve service availability. NoSQL databases are not using the relational model, and they are increasingly used in big data applications, which are consuming more and more energy these days, and hence would significantly benefit from energy-aware designs.

References

[1] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: An empirical study," in *MSR*, 2014, pp. 2–11.

[2] N. Nikzad, O. Chipara, and W. G. Griswold, "Ape: An annotation language and middleware for energy-efficient mobile application development," in *ICSE*, 2014, pp. 515–526.

[3] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, *Power aware computing*. Kluwer Academic Publishers, 2002, ch. The case for power management in web servers, pp. 261–289.

[4] 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*, ser. ESEM '14, 2014, pp. 36:1–36:10. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652538

[5] F. Prosperi, M. Bambagini, G. Buttazzo, M. Marinoni, and G. Franchino, "Energy-aware algorithms for tasks and bandwidth co-allocation under real-time and redundancy constraints," in *Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2012, pp. 1–8.

[6] I. El Korbia and S. Zeadallyb, "Energy-aware sensor node relocation in mobile sensor networks," *Ad Hoc Networks*, vol. 16, pp. 247—-265, 2014.

[7] 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 *6th International Conference on Cloud Computing Technology and Science*. IEEE, 2014, pp. 278–283.

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

[9] 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.

[10] "Data Replication and Synchronization Guidance," http://msdn.microsoft.com/en-us/library/dn589787.aspx/, 2014, [Online; accessed August-2015].

[11] "Sharding Pattern," http://msdn.microsoft.com/en-us/library/dn589797.aspx, 2014, [Online; accessed August-2015].

[12] A. Homer, J. Sharp, L. Brader, M. N. Narumoto, and T. Swanson, *Cloud Design Patterns Prescriptive Architecture Guidance for Cloud Applications (Microsoft patterns practices)*. Microsoft patterns practices, February 2014.

[13] 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.

[14] B. G. Tudorica and C. Bucur, "A comparison between several nosql databases with comments and notes," in *Roedunet International Conference (RoEduNet), 2011 10th*. IEEE, 2011, pp. 1–5.

[15] R. Burtica, E. M. Mocanu, M. I. Andreica, and N. Tapus, "Practical application and evaluation of no-sql databases in cloud computing," in *Systems Conference (SysCon), 2012 IEEE International*. IEEE, 2012, pp. 1–6.

[16] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[17] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann, "Performance evaluation of message-oriented middleware using the specjms2007 benchmark," *Performance Evaluation*, vol. 66, no. 8, pp. 410–434, 2009.

[18] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester, "Overall ict footprint and green communication technologies," *ISCCSP*, vol. 10, pp. 1–6, 2010.

[19] "MySQL in the Cloud," http://www.mysql.com/why-mysql/cloud/, 2014, [Online; accessed August-2015].

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

[21] D. Haney and K. S. Madsen, "Load-balancing for mysql," *Kobenhavns Universitet*, 2003.

[22] "Sharding algorithms," http://kennethxu.blogspot.fr/2012/11/sharding-algorithm.html, [Online; accessed August-2015].

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

[24] 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'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

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

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

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

[28] "Data for our experimental results," https://goo.gl/Cczot4, 2015, [Online; accessed August-2015].

[29] G. Pinto, F. Castor, and Y. D. Liu, "Mining questions about software energy consumption," in *MSR*, 2014, pp. 22–31.

[30] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?" *IEEE Software*, 2015, to appear.

[31] "NoSQL Databases," http://nosql-database.org, 2015, [Online; accessed August-2015].