Thibault PERE
Hugo VAUTRIN
Class 2012-2015

Polytechnique Montreal
Quebec
Canada

4th Year Internship Report
from 2nd June to 29th August 2014

An Empirical Study of the Impact of Cloud Patterns on Quality of Service and Energy Consumption

Tutors Polytech
Esther PACITTI
Lysiane BUISSON

Tutors Polytechnique
Naouel MOHA
Foutse KHOMH

SUMMARY

TABLE OF FIGURES

Acknowledgments

We would like to express our sincere gratitude to Naouel Moha and Foutse Khomh -our tutors- for their continuous support during this internship, for their motivation and enthusiasm. Their guidance helped us during these three exciting months.

Besides them, we would like to thank Yann-Gaël Guéhéneuc for giving us the opportunity to come here for our internship.

Our gratitude also goes to Clément De Figueiredo and Benjamin José-Scheidt for their help and time at our arrival in Montreal. Also, we thank our friends who accompanied us to this wonderful city: Fabien Dos Santos, Simon Haioun-Viet and Alexandre Laffaille for all the great moments spent together and all the fun we have had in the last three months.

In particular, we are grateful to Bram Adams for his kindness and his time at our arrival in Polytechnique but also to Francis Gagnon and Jean-Marc Chevalier for their great help during the installation of our architecture.

Last but not least, we would like to thank Tiberiu Stratulat for his help at the beginning of this internship.

Figure 1: Montreal City

Introduction

In recent years, there have been a rapid growth of cloud environments and cloud oriented software. Reducing the energy usage of this kind of solution is becoming more and more important (cf. References: Software solutions). Indeed, the new goal of software engineers is to reduce the energy consumption by making use of good design and implementation decisions. In fact, our intuition is that energy consumption can be reduced by designing more energy efficient software. They can use for example cloud patterns: they are general reusable solution to a commonly occurring problem within a given context in software design for a cloud application. However, engineers have a large number of possible implementations at their disposal but few feedback about their utilization in concrete context of cloud environment.

Figure 2: Green IT, a hot topic

Moreover, Cloud Patterns can have a great impact on Quality of Service (*QoS*) but can be less performant in term of energy consumption: there is a trade-off to make here (cf. Appendix a).

Our fourth year's internship takes place within this problematic, in the *École Polytechnique de Montréal* [1], an engineering school affiliated to the *Université de Montréal*.

Under the supervision of Prof. Foutse Khomh [2] (*École Polytechnique de Montréal*), Prof. Naouel Moha [3] (*Université du Québec à Montréal* [4]) and Prof. Yann-Gaël Guéhéneuc [5] (*École Polytechnique de Montréal*) in the *SWAT laboratory* [6], we have done several different missions during this internship.
During this internship, we targeted three main missions: the first one was about doing research on sharding, replication, reading articles. We also assisted two other French students at *UQAM* for tests and implementations of their study about the impact of cloud patterns on Quality of Service.
The second one was a research of tools in order to measure energy consumption, setting up architecture for our research project at *Polytechnique* with servers, replication and sharding design.
Last but not least, the third part of our work placement consisted in performing tests and collecting data and discussing them.

[1] www.polymtl.ca
[2] www.khomh.net
[3] www.naouelmoha.net
[4] www.uqam.ca
[5] www.yann-gael.gueheneuc.net
[6] www.swat.polymtl.ca

This report will focus first on describing our environment during our 3 months internship in Montreal: context and challenges of our missions, choices we made, methods and tools used, and results with discussions. We will also describe our project management.

At the end of this report, you will find a conclusion about this internship and all the references we used.

PART I - Development

I: Polytechnique Montreal

The *École Polytechnique de Montréal* is an engineering school affiliated with the *Université de Montréal* in Montréal (Quebec, Canada). The *École Polytechnique* is known for its dynamic research split among seven departments such as the *Computer and Software Engineering Department*.

Figure 3 : Polytechnique's logo

This internship was found through the intermediary of Yann-Gaël Guéhéneuc who is Professor at the *Computer and Software Engineering Department*. He leads the *Ptidej Team* (Pattern Trace Identification, Detection, and Enhancement in Java), and aim at developing theories, methods, and tools, to evaluate and to improve the quality of object-oriented programs by promoting the use of idioms, design patterns, and architectural patterns.
With his help, we obtained this work placement under the supervision of Foutse Khomh and Naouel Moha for a period of three months (June 2 to August 29).

Foutse Khomh is an Assistant Professor at the *École Polytechnique de Montréal* where he also leads the *SWAT Team* on software analytics and cloud engineering research.

Naouel Moha is currently associate professor at the *Department of Informatics* at the *Université du Québec à Montréal* (*UQAM*) and adjunct director of the institutional research centre *LATECE*. Her research work focuses on software quality, maintenance and evolution.

II: Context and Challenges

In a cloud context, Cloud Patterns seems to be good solutions when facing design problems in this kind of environment. They are, most of the time, inspired by Design Pattern from Object-Oriented Software or Service Oriented Architecture (SOA) where they are considered to be good solutions to recurrent design problems.

At our arrival at *Polytechnique*, there were already two students working with Foutse Khomh and Naouel Moha at the *UQAM* since January 2014.
With the collaboration of Geoffrey Hecht, PhD student of UQAM and *Université Lille 1*, they performed an empirical study of the Impact of Cloud Patterns on the Quality of Service (Figure 4 for more details) of cloud based software systems. They studied the following 3 Cloud Patterns:

- Local Database Proxy
- Local Sharding-Based Router
- and Priority Message Queue Patterns [7]

Figure 4: Quality of Service?

[7] read Glossary for more information about these three patterns

This is where we started our internship. Clément and Benjamin, the two students already here, were leaving at the end of June and we had to keep up their work.

To fulfill this first goal, we were first asked to:

- read articles to learn more about Cloud Patterns
- meet Clément and Benjamin for having more information about the study
- attend meetings with all the team composed of Foutse Khomh, Naouel Moha, Geoffrey Hecht, Clément De Figueiredo and Benjamin José-Scheidt
- gain knowledge of the source code of the project
- do some research about the Local Sharding-Based Router pattern which was still not implemented by Clément and Benjamin because they were lacking time

After Clément and Benjamin left the team, we were asked to finish all the missing tests for the article that the team intended to submit to the CloudCom conference, under the name:

"**An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS)**
- The Case of Local Database Proxy, Local Sharding-Based Router -
and Priority Queue Patterns"

Conducting these tests consisted of three steps; the first one was about configuring the work environment when then second one was running tests and the last one was collecting and sending results to Geoffrey Hecht. Then he would be able to incorporate them in their paper.
During the configuration stage, we had to set up MySQL to match with our needs. We also needed to have the right architecture to deploy the Cloud application used for the tests.
Concerning the tests, we had three different setups to test: one with only sharding, one with sharding and message queue and one with sharding and proxy.
The aim of these tests was to increase little by little the amount of request to see in which case a setup is better than another (for example, we want to say in which situation the proxy pattern is better than the sharding pattern and which implementation of the pattern is better).

Figure 5 is an example of the result we had after the testing phase. It concerns the Local Sharding-Based Router with the Priority Message Queue with write request. The results show that the combination of these two different patterns affects significantly (and in good terms) the QoS.

Comment: Modulo, Lookup and Consistent are three different algorithms that can be used to implement the Sharding design (cf. Appendix a).

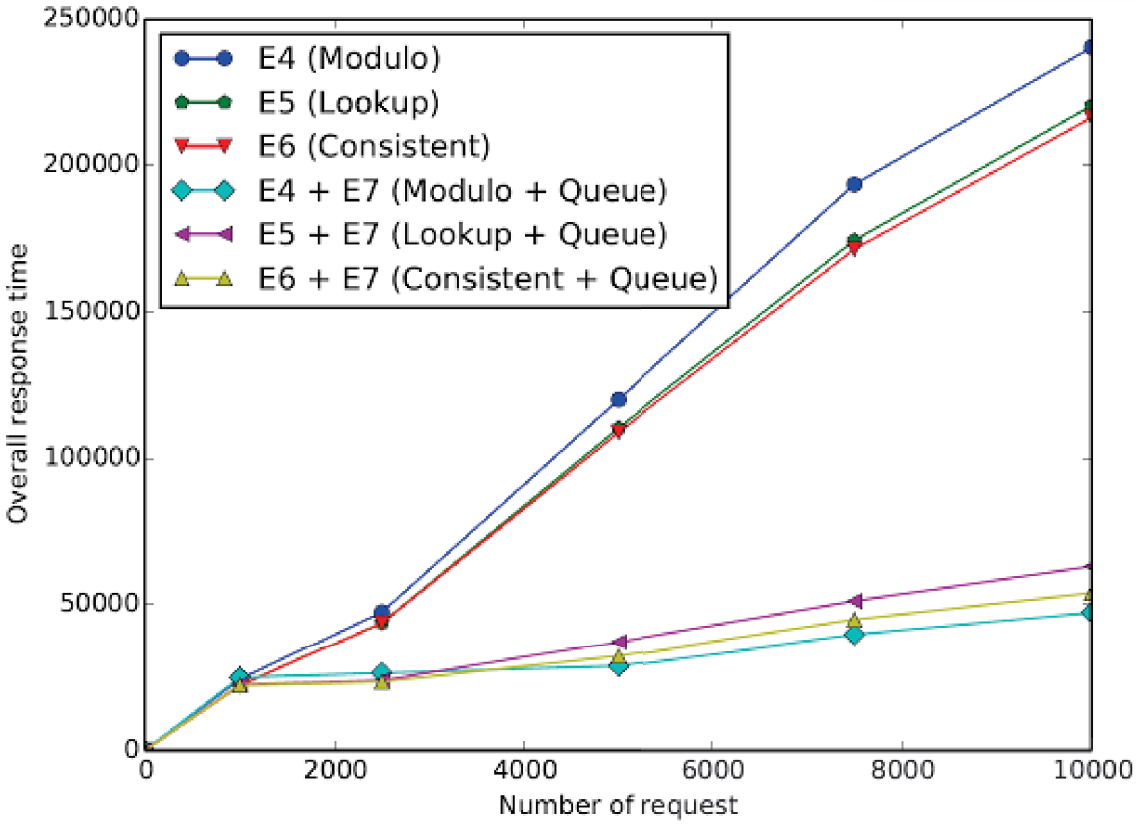

Figure 5: Insert film with Local Sharding-Based Router and Priority Queue

Figure 6: Google Datacenter

This paper with all its experiments aims to show three important results:

- the pattern effect is not always significant and can even be negative in some cases
- the usage of different algorithms to implement patterns affects QoS to a lesser extend compare to the choice of pattern
- the interaction of patterns affects significantly the QoS

At the end of this work about the impact of cloud patterns on QoS, we were asked by Foutse Khomh to continue this project but on **energy consumption**.

III: Choices and Solutions

Energy consumption is nowadays a hot topic, given the widespread of cloud environment. However, there are not a lot of tools available to track the energy consumption of a software or to monitor it (for machine, there are quite some).

Our first challenge was to review the literature to find a tool -hardware or software- that could be used to monitor the energy consumption of a software process.

Solution	Infos	Infrastructure	Method	Measures
PowerAPI	- Free - Developed by INRIA - Window / Linux	- Framework - Scala based - Using Maven	- Based on energy consumption of hardware components - Need components configurations	- Measure of application consumption (Watts)
EnergyChecker	- Free - Developed by Intel	- Software	- Using an hardware tool to calculate	- Energy consumption of code source
PowerTop	- Free - need Intel processor - need 2.6.21+ Linux kernel		- Using battery consumption	- Measure of global consumption
PTop	- Free - Window / Linux		- creation of variable able to measure consumption	- Measure of component consumption (Joule)
JouleMeter	- Free - Developed by Microsoft - Window Only		- Using tools to be configured	- Measure of component consumption - Measure of application consumption
PowerStat	- Free - Linux Only		- Using battery consumption	- Watts / sec - Watts max - Watts min

Figure 7: Comparative Table of Tools

According to this table (Figure 7) and after a meeting with Foutse Khomh, we decided to use PowerAPI because it allows us to monitor the exact process we want and this seemed quite interesting.

IV: Methods and Tools

PowerAPI

PowerAPI is a framework designed to monitor energy consumption of processes. We will now see how it works.

We can see on the top of Figure 8 that we need to know an important thing: the PID (Process Identifier) of the process to monitor. PowerAPI can monitor a list of processes if needed, but next steps are the same.
Then all you have to do is use the framework. This framework permits you different configurations.

Step 1: choose how you want PowerAPI to monitor. Two ways are possible for Linux (using / proc files or using SIGAR library) and one way for Window (SIGAR library). The result will be independent of the chosen way.

Step 2: define the output type. It depends on the future use of your results: tables, console or chart. In our study, we chose to use tables to allow us to process data easily. We will have some examples of this later.

Step 3: choose which hardware component you want to listen, indeed you can monitor the energy consumption of a process on the CPU, the Disk or the memory. You can obviously monitor all three.

PowerAPI is able to know the energy consumption thanks to formulae using information from sensors taken during a chosen period of time, for example each X second. You can define X as an integer.

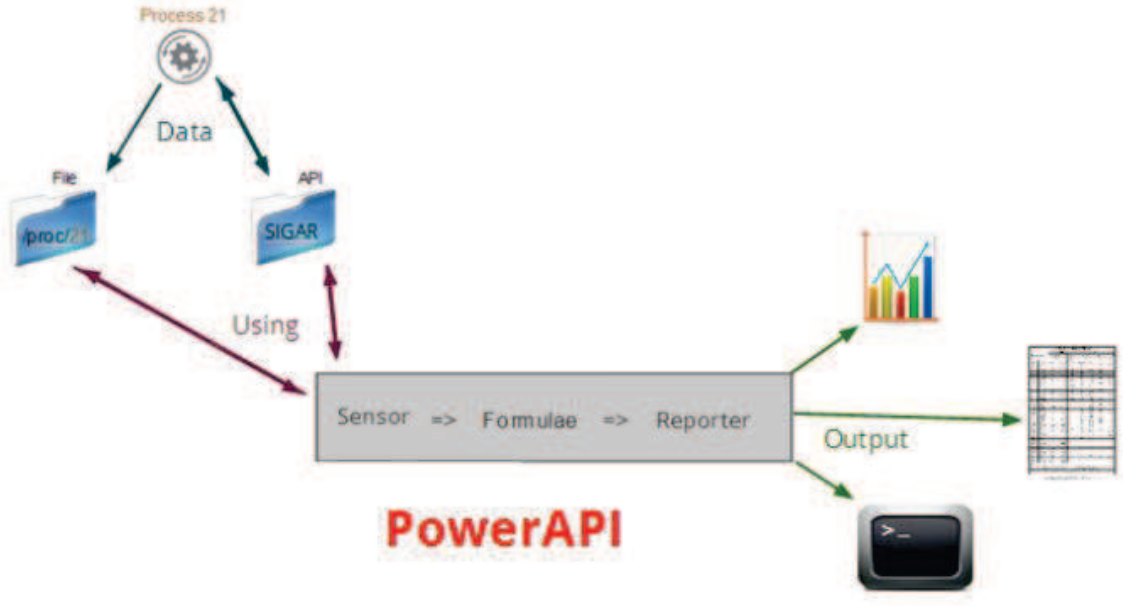

Figure 8: PowerAPI

Architecture

The application

The test application (A Java Web Application developed with the Java Development Kit 1.7) is hosted on a GlassFish 4 Application Server. It is located on an 8 Go RAM, 50 Go DD, Ubuntu Server 14.04.

This application is a distributed application (client-server) which communicates over REST calls.

Databases and virtualization

As database management system, we have chosen MySQL for the following reasons:

- light
- efficient
- give enough utilities and tools to implement our patterns
- one of the most popular database management system for Cloud application

To host our databases, we have three different physical servers. Each server is running through a native hypervisor (in our case we have chosen ESXi, a native hypervisor as explained in the figure 9).
This hypervisor allows us to create virtual machines: each server hosts 4 Virtual Machines (2 Virtual CPU, 1Go RAM, 20 Go DD, MySQL 5.6, OpenSSH Server, Ubuntu Server 14.04).

Figure 10 presents an overview of the architecture that was used during these three months.

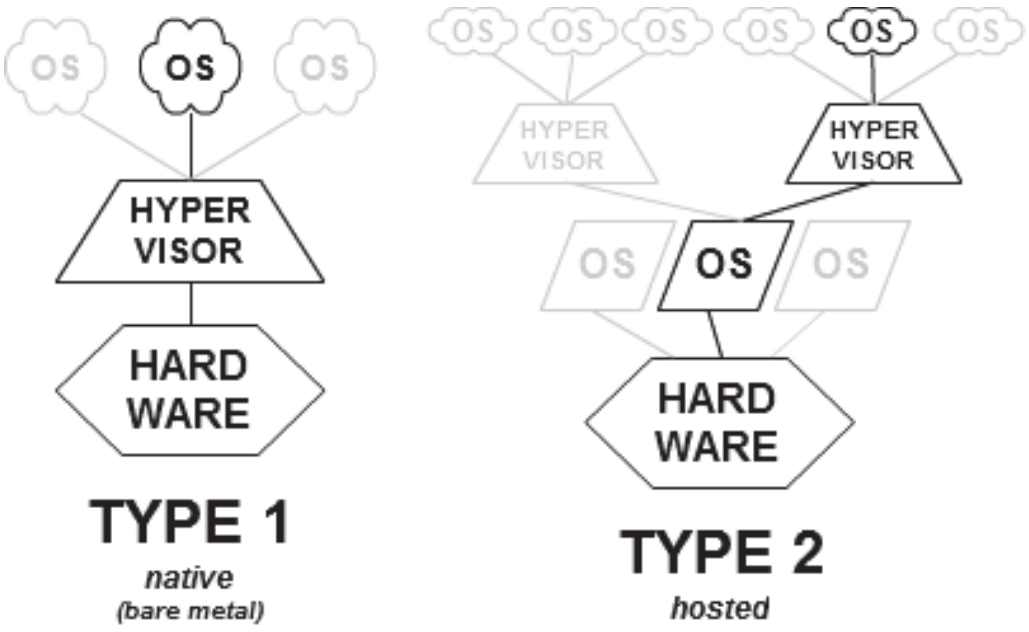

Figure 9: Hypervisor, native versus hosted

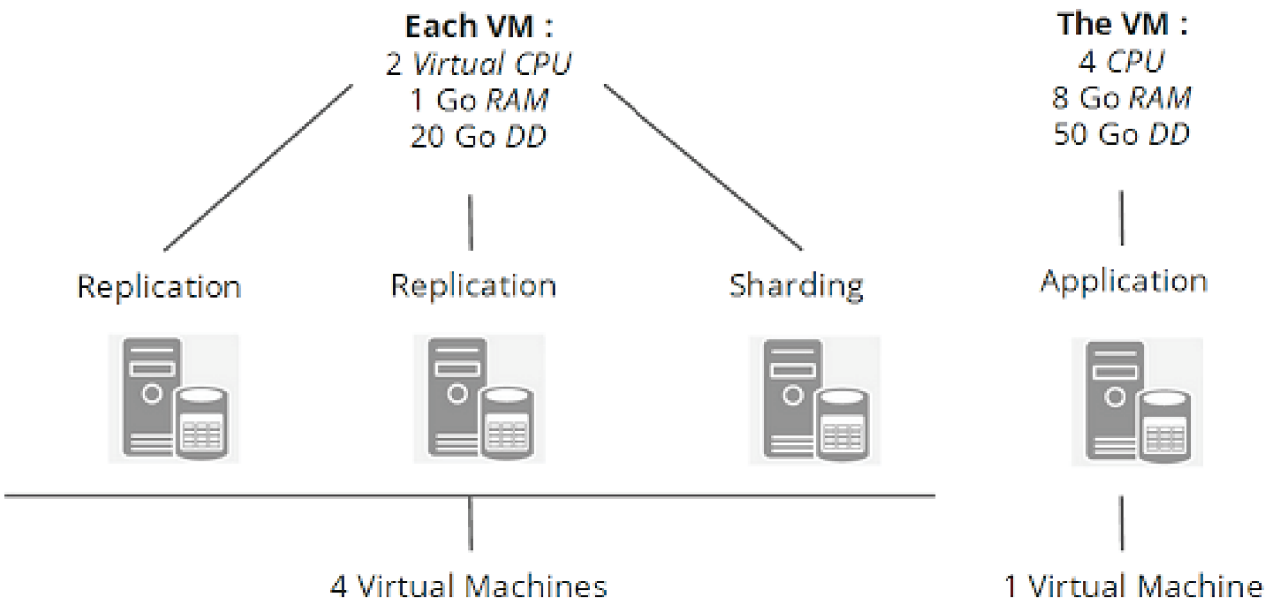

Figure 10: The architecture we used for our tests

Methods

In order to study the impact of the three design patterns:

- Local Database Proxy
- Local Sharding Based Router
- and Priority Message Queue

We decided to implement them in a java web application for specific testing phase: we developed several algorithms to test our patterns and obtain metrics like response time or latency. To do so, we needed different test scenarios (test cases). These test cases will be described later. With these experiments, we have obtained several results. We will also discuss about these results later.

Our null hypothesis was:

H0: Design Patterns doesn't have any effect on Energy Consumption

We planned to refute this hypothesis by proving with several test cases that, design patterns in a cloud environment have indeed effect on energy consumption.

Test Cases

To answer our research question, we did multiple experimentations with the application designed specifically to test our three cloud patterns.

In order to have realistic results, we decided to implement in our application a realistic test case. In the next paragraph, we will describe a test case that was created and used for the Local Database Proxy.

Realistic Scenarios for our tests (Cloud Application and Environment)

Number of Repetition: 3
Number of Clients: {500; 1500; 3000}
Environment: 1 MASTER - 4 SLAVES (2:2)
Databases: 100 movies

What does a client do?

a. 1 READ - CONNEXION
b. SLEEP (5s) - READ THE MENU
c. 10 READ - CONSULTING 10 MOVIES
d. SLEEP (30s) - THINKING, SEARCHING ONLINE, CHATTING...
e. 5 READ - CONSULTING 5 OVER MOVIES
f. SLEEP (10s) - TAKING A DECISION
g. 1 WRITE - RENTING A MOVIE
h. SLEEP (5s) - READING THE VALIDATION OF THE RENT
i. LOGOFF - LOGGING OF THE SERVICE

READ: a read request to the database
SLEEP(x): the client is waiting x seconds before going to the next step
WRITE: a write request to the database
LOGOFF: the client is disconnecting from the service

This test case represents a realistic use of a cloud application by a client in a rental movie cloud application environment.

V: Results and Discussions

In this part, we are going to present you some of the result we have obtained during our study and we will focus on the results of the Proxy pattern implementation and the sharding pattern implementation.
We will now see the result of the test we described previously. As you can see on the figure 11, algorithms and patterns have an effect on energy consumption. With this test we can say that as expected not using the Proxy pattern consume less than any implementation of this one. Indeed, Proxy pattern need the server to do some actions to decide where he will send clients requests. These actions consume energy. Each algorithm has a different way to decide where to send clients request, impacting performance and energy as proved before.

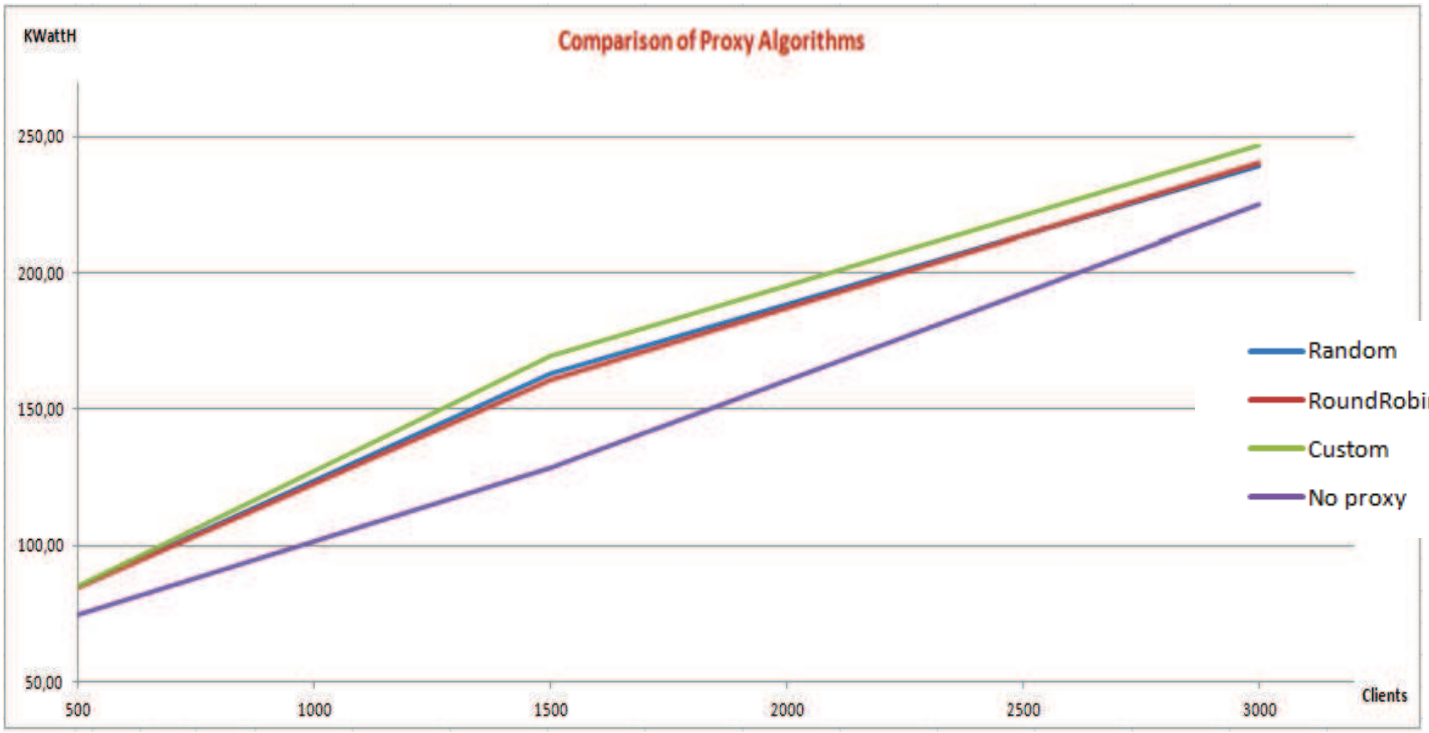

Figure 11: Comparison of Proxy Algorithms

An important aim of our study was to give an idea of the difference between two solutions in terms of energy. We can therefore say that:

- with 500 clients the difference between No Proxy and Custom is about the same energy consumption of a classical use of a hair dryer (48 weeks per year and 30 minutes per day)
- with 1500 clients the difference between No Proxy and Custom is about the same energy consumption of a classical use of a coffee maker (335 days per year and 10 minutes per day)
- with 3000 clients the difference between No Proxy and Custom is about the same energy consumption of a LCD TV in eve mode during a whole year

With these results and the article wrote by Clément and Benjamin (in appendix), software designers are able to take decision: for example, is it better for me to use Custom algorithm (which consume more but is better in term of performance) than No Proxy?

Concerning the Sharding pattern, the test case was the same as the one we used for the proxy pattern because we wanted to have the same scale between this two patterns for comparison. The goal is also the same, we wanted to see if the sharding pattern has an impact on energy consumption and if so, if his different implementations have also an impact. The sharding pattern has three different implementations we studied:

- Modulo
- Consistent
- Look Up

The figure 12 shows our results for the sharding pattern and its three implementations.

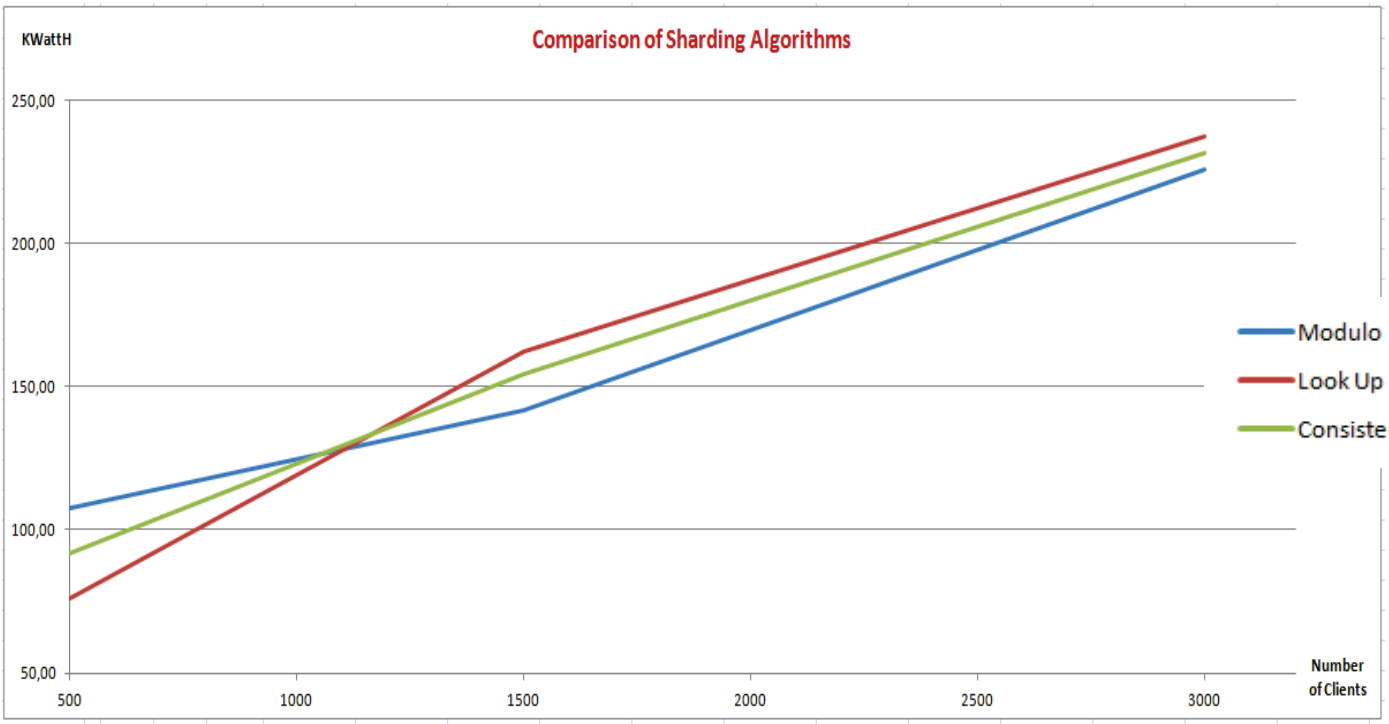

Figure 12: Comparison of Sharding Algorithms

- with 500 clients the difference between Look Up and Modulo is about the same energy consumption of a lamp energy saving bulb (5 hours a day)
- with 1500 clients the difference between Modulo and Look Up is about the same energy consumption of a LCD TV in eve mode during a year
- with 3000 clients the difference between Modulo and Look Up is about the same energy consumption of a classical use of an hair dryer (48 weeks per year and 30 minutes per day)

We can refute our null hypothesis which was "H0: Design Patterns doesn't have any effect on Energy Consumption". Indeed, our first results show us that Design Patterns have an effect on Energy Consumption and that there is a trade-off to make between Energy Consumption and Quality of Service. There is still research to do in this area. Software can be design to consume less energy and we have to because Cloud computing is becoming such a huge thing, we have to reduced its impact on the environment by making Green IT.

PART II- Project Management

To realize this internship as well as it was possible, we needed to organize ourselves. At the beginning of this work placement, we had a documentation phase. We decided to share all the work (researches and readings). Each of us had some articles to read and summarize into a shared document. Then we were able to easily know key information about the work of the other. It worked well and allowed us to save a lot of time.

In a second part, we had two fronts to lead. The first one was about ending tests as we described earlier. The second was about trying to understand how PowerAPI works, be aware of the environment it needs and master the tool. One of us was in charge of carrying out the first one when the other carried out the second. The good thing with this solution was that both of us took time to develop new skills and knowledge. For example Thibault learned a lot about MySQL configurations, architecture stuffs while Hugo learned a lot about the Linux kernel, Scala language and Maven software. All these new skills were useful as we will see in the next step.

In a third part, we had to realize our own tests for our own study. To do so, we decided to work together and to gather our knowledge to configure our architecture, install our tools and start all the work described in this report. This working method was in our opinion good for two reasons: we had only one computer to realize all these tests so splitting the work would have not been faster. We shared all the knowledge learned, which end up saving us a lot of time.

We also did a weekly meeting with Naouel Moha and Foutse Khomh to discuss about the work we did since the last meeting. That was really helpful and allowed us to keep objectives in mind. We also used it to correct little by little some mistakes we did instead of doing it at the end of the internship and have a work overload.

Conclusion

Overall, this internship was a great experience. We gained a lot of experience, especially in design patterns, databases, and hardware architecture. Moreover, *Polytechnique Montreal* has offered us opportunities to set up a good architecture for our study in a safe and private environment.

We are still discussing about the possibility of publishing an article about our study. Also, it is expected that some new students will pick up and continue our work.

In review, this internship has been an excellent and rewarding experience. We are grateful and thankful that we got to experience and learn so many things and also to discover Montreal, this wonderful city.

Figure 13: University of Toronto (August 2014) - Fabien, Simon, Alexandre, Hugo and Thibault

References

Hardware Solutions

DUSTIN MCINTIRE, KEI HO, BERNIE YIP, AMARJEET SINGH, WINSTON WU, AND WILLIAM J. KAISER. *The Low Power Energy Aware Processing (LEAP): Embedded Networked Sensor System*. IPSN'06. 2006

DIGVIJAY SINGH, PETER A. H. PETERSON, PETER L. REIHER AND WILLIAM J. KAISER. *The Atom LEAP Platform For Energy-Efficient Embedded Computing: Architecture, Operation, and System Implementation*. http://tastytronic.net/~pedro/docs/leapwhitepaper.pdf.

ABRAM HINDLE, ALEX WILSON, KENT RASMUSSEN, E. JED BARLOW, JOSHUA CHARLES CAMPBELL, STEPHEN ROMANSKY.
GreenMiner: A Hardware Based Mining Software Repositories Software Energy Consumption Framework. MSR'14. http://dl.acm.org/citation.cfm?doid=2597073.2597097. 2014

Software Solutions

TAO LI, LIZY KURIAN JOHN.
Run-time Modeling and Estimation of Operating System Power Consumption. SIGMETRICS'03. 2003

MARIO LINARES-VÁSQUEZ, ROCCO OLIVETO, GABRIELE BAVOTA, MASSIMILIANO DI PENTA, CARLOS BERNAL-CÁRDENAS, DENYS POSHYVANYK.
Mining Energy-Greedy API Usage Patterns in Android Apps: An Empirical Study. MSR'14. http://dl.acm.org/citation.cfm?doid=2597073.2597085. 2014

NADINE AMSEL, BILL TOMLINSON.
Green Tracker: A Tool for Estimating the Energy Consumption of Software. CHI. 2010

IRENE MANOTAS, LORI POLLOCK, JAMES CLAUSE.
SEEDS: A Software Engineer's Energy-Optimization Decision Support Framework. ICSE 14. http://dl.acm.org/citation.cfm?doid=2568225.2568297. 2014

DING LI, ANGELICA HUYEN TRAN, WILLIAM G. J. HALFOND.
Making Web Applications More Energy Efficient for OLED Smartphones. ICSE 14. http://dl.acm.org/citation.cfm?doid=2568225.2568321. 2014

Middleware Solutions

NIMA NIKZAD, OCTAV CHIPARA, WILLIAM G. GRISWOLD.
APE: An Annotation Language and Middleware for Energy-Efficient Mobile Application Development. ICSE 14. http://dx.doi.org/10.1145/2568225.2568288. 2014

Patterns

STEVE STRAUCH, VASILIOS ANDRIKOPOULOS, UWE BREITENBUCHER, SANTIAGO GOMEZ SAEZ, OLIVER KOPP, FRANK LEYMANN.
Using Patterns to Move the Application Data Layer to the Cloud: The Fifth International Conferences on Pervasive Patterns and Applications. 2013

Glossary

Cloud:
A type of computing that relies on sharing computing resources rather than having local servers or personal devices to handle applications

Cloud Pattern:
A design pattern for a cloud application

Database Proxy:
See Replication

Design Pattern:
A general reusable solution to a commonly occurring problem within a given context in software design

Framework:
A reusable set of libraries or classes for a software system

Hypervisor:
A piece of computer software, firmware or hardware that creates and runs virtual machines

Maven:
A software project management and comprehension tool. It is able to create links and dependencies between projects

MySQL:
An open-source relational database management system owned by Oracle Corporation

Priority Message Queue:
A software engineering component used for inter-process communication or inter-thread communication

Ptidej Team:
Pattern Trace Identification, Detection, and Enhancements in Java

QoS (Quality of Service):
The overall performance of a service, particularly seen by the users

Read request:
A request to a database in order to consult data

Replication (Database Proxy):
A technique that involves sharing information to ensure consistency between redundant resources to improve reliability, fault-tolerance and accessibility

REST (Representational State Transfer):
An architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system

Scala:
A programming language considered as an evolution of Java

Sharding:
A type of database partitioning that is used to separate very large databases into smaller, faster data shards.

SOA (Service Oriented Architecture):
A software design and software architecture design pattern based on distinct pieces of software providing application functionality as services to other applications

SWAT Team:
The SoftWare Analytics and Technologies Lab

Virtual machine:
An emulation of a particular computer system and functions of a real or hypothetical computer

Write request:
A request to a database in order to create data

Appendices

a - An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS) - The Case of Local Database Proxy, Local Sharding-Based Router - and Priority Queue Patterns (8 pages)

An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS)

— The Case of Local Database Proxy, Local Sharding-Based Router — and Priority Queue Patterns

Geoffrey Hecht[1,2], Benjamin Jose-Scheidt[1], Clément De Figueiredo[1], Naouel Moha[1], Foutse Khomh[3]
[1]Université du Québec à Montréal, Canada, [2]Université Lille 1, France, [3]SWAT, École Polytechnique de Montréal, Canada
geoffrey.hecht@courrier.uqam.ca, {benjamin.josescheidt, clement.defigueiredo}@viacesi.fr
moha.naouel@uqam.ca, foutse.khomh@polymtl.ca

***Abstract*—Cloud patterns are described as good solutions to recurring design problems in a cloud context. These patterns are often inherited from Service Oriented Architectures or Object-Oriented Architectures where they are considered good practices. However, there is a lack of studies that assess the benefits of these patterns for cloud applications. In this paper, we conduct an empirical study on a RESTful application deployed in the cloud, to investigate the individual and the combined impact of three cloud patterns (*i.e.,* Local Database proxy, Local Sharding-Based Router and Priority Queue Patterns) on Quality of Service (QoS). We measure the QoS using the application's response time, average, and maximum number of requests processed per seconds. Results show that cloud patterns doesn't always improve the response time of an application. In the case of the Local Database proxy pattern, the choice of algorithm used to route requests has an impact on response time, as well as the average and maximum number of requests processed per second. Combinations of patterns can significantly affect the QoS of applications. Developers and software architects can make use of these results to guide their design decisions.**

***Keywords*—Cloud Patterns, Replication, Sharding, Priority Queue, QoS.**

I. Introduction

Design Patterns are general and reusable solutions to recurring design problems. They were first introduced in software engineering by Beck and Cunningham [1] in 1987 but really gained popularity only after the publication of the book of Gamma *et al.* [2] in 1994. Since then, design patterns have been applied to all field of software engineering, including Software Architecture [3], Service Oriented Architecture (SOA) [4] and Cloud Computing [5]–[10].

Most cloud patterns like *Proxy Service*, *message queue* or *Composed Service* were adopted from SOA or parallel computing [9]. These patterns were refined to take into account the specificities and requirements of the cloud. For example, the *message queue* pattern is usually used to allow asynchronous messaging between two components. In the cloud context, this pattern is used to reduce coupling between components and thus allowing a better scalability and availability of the overall application [6]. Therefore the *message queue pattern* is supposed to improve architecture quality as well as Quality of Service (QoS).

Despite several benchmarks and studies [11]–[15] comparing cloud solutions and technologies that use patterns (*e.g., NoSQL databases* that use database sharding and message queue patterns or *message-oriented middleware*), to the best of our knowledge, there is a lack of studies that empirically investigate the impact of multiple cloud patterns on the QoS of applications. Consequently the benefits and tradeoffs of cloud patterns are mostly intuitively discovered and not properly validated. Moreover, the available benchmarks evaluated patterns in isolation and did not considered possible interactions between multiple patterns.

In this paper, we evaluate the impact on QoS of three cloud patterns : the Local Database Proxy, Local Sharding-Based Router and Priority Queue Patterns. The study is performed using a RESTful cloud-based, data-centered and service based application implemented with different combinations of the aforementioned patterns. To measure the QoS we rely on the following three metrics : response time, average and maximum queries processed per second. Our objective is to provide evidence to confirm or refute the claimed efficiency of these patterns and comprehend the interplay between them.

The rest of the paper is organized as follows. We provide some background information describing the studied patterns in Section II. Section III presents related works on the impact of design patterns. Section IV presents the design of our experiments and section V discusses the obtained results. Section VI concludes our study and outlines some avenues for future works.

II. Background

In this section, we briefly present the three patterns under study in this paper and outline their benefits for cloud applications as identified in the literature.

Local Database Proxy : The Local Database proxy pattern provides a read scalability on a relational database by using data replication between master/slave databases and a proxy to route requests [7]. All write requests are handled by the master and replicated on its slaves while read requests are processed by the slaves. Unlike usual replication mechanisms where application components access a predetermined replica

[16], with this pattern, 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. It should be noted that this pattern is not suitable when there is a need to scale with write request since there is a risk of bottleneck on the master database. This pattern is described by Strauch *et al.* [7] in their work about Non-functional Data Layer Patterns for Cloud Applications. Using the Local Database Proxy pattern, Microsoft [16] provided guidelines for the replication in a cloud application. These two works recommend implementing the Local Database Proxy pattern to improve the scalability for data reads, as well as the availability and resiliency of applications. The risk of bottlenecks on the master database and the lack of strategy for write requests are listed among the limitations of this pattern. To the best of our knowledge, no work has empirically investigated the impact of the Local Database Proxy pattern on the QoS of applications.

Local Sharding-Based Router : The Local Sharding-Based Router is recommended when the need for scalability concerns read and write operations [7]. Data are split among multiple databases into functional groups called shards, requests are processed 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. Multiple strategies can be used to determine the sharding logic, a range of value, a specific shard key or hashing can be used to distribute data among the databases [17]. In addition, this pattern can used a replication mechanism for each shard to ensure a strong resilience. This pattern is also described by [7] and [17]. The Local Sharding-Based Router pattern is recommended to improve the overall scalability of the storage when data can be split into independent shards. The impact of this pattern on the QoS of applications is yet to be investigated, which is the purpose of the study presented in this paper.

Priority Message Queue : Message Queues are First In First Out (FIFO) queues typically used to delegate tasks to background processing or to allow asynchronous communications between components. When different types of messages exist, a Priority Message Queue can be used to gain flexibility. Priority values are set by the sending component. Messages with high priority values are received and processed more quickly than those with lower priority values [5]. Multiple simple message queues can be used to implement a priority message queue, considering a different priority for each queue. Message Queues are considered good practices for cloud applications, to design loosely coupled components and to improve scalability [6].

III. Related Work

In this section, we discuss the relevant literature about the impact of patterns on software quality.

Impact of Object-oriented design patterns : Several works exists in the literature to assess the impact of design patterns on software quality [18], [19], software maintainability [20], [21] and code complexity [22]. Overall, these studies found that design patterns do not always improve the quality of applications. Khomh and Guéhéneuc [18] claim that design patterns should be used with caution during software development because they may actually impede software quality. Object Oriented design patterns are usually not supposed to increase performance, nevertheless, Aras *et. al* [23] have found that design patterns can have a positive effect on the performance of scientific applications despite the overhead that they add. Of course the results of these studies cannot be directly generalize to cloud patterns which usually focus on scalability, however they provide hints about the possible benefits–or–downside of cloud patterns. Clearing up the impact of cloud patterns on QoS is important to help software development teams make good design decisions.

Evaluation of Cloud Patterns : Ardagna *et al.* [24] 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 also explored the effects of the addition and the removal of virtual resources. Tudorica *et al.* [12] and Burtica *et al.* [13] provide a comprehensive comparison and evaluation of no-SQL databases which make use of multiple sharding and replication strategies to increase performance. However they did not considered the impact of these solutions on the QoS of the overall application and the association with others patterns. Similarly, Cattel [11] examined no-SQL and SQL data stores designed to scale by using replication and sharding. His work highlighted the lack of studies and benchmarks on these solutions. Message oriented middlewares have been benchmarked by Sachs *et al.* [14] and the performance of priority queues has been evaluated by Alwakeel *et al.* [15]. In these works, the message queue is evaluated as a technical solution in isolation, without considering any application context. The message queue is just considered as a solution to allow asynchronous messaging.

IV. Study Design

This section presents the design of our study, which aims to understand the impact of cloud patterns on the QoS of applications and investigate potential interactions between these patterns. We select three cloud patterns (*i.e.,* Local Database proxy, Local Sharding-Based Router and Priority Queue Patterns) which are described as good design practices by both academic and practitioners and address the following research questions:

1) Does the implementation of Local Database proxy, Local Sharding-Based Router or Priority Message Queue Patterns affect the QoS of cloud applications?
2) Do interactions among Local Database proxy, Local Sharding-Based Router and Priority Message Queue Patterns affect the QoS of cloud applications?

To answer these research questions, we perform a series of experimentations with multiple versions of an application designed specifically to test the aforementioned cloud patterns. The Local Database proxy and Local Sharding-based router patterns were implemented with different algorithms which are explained in section IV-C. In total we analyze eight versions of the application which are summarized in Table I. The Priority Message Queue was combined with the two others patterns in some experiments. The application was built around an SQL Database. The results were collected by performing a series of stress tests on the application (varying the number of requests) and tracing their executions. The same test sets were used for all the experimentations in order ensure comparable results. The remainder of this section elaborates more on the details of our experimentations.

A. Objects

The application used in this study is hosted on a GlassFish 4 application server. It is a distributed application (client-server), which communicates through REST calls. We choose MySQL as database management system because it's light, efficient and provides enough utilities and tools to perform and tune the implementation of every pattern. Moreover, MySQL is one of the most popular database for Cloud applications [25]. We use the Sakila sample database [26] provided by MySQL. It's a good sample for experiments because it contains a large number of records and it is consistent with existing databases. Sakila is composed of 16 tables, 7 views and procedures and about 50,000 records. The test application was fully developed with the Java Development Kit 1.7 and is composed of about 3,500 lines of code and its size is 6 MB. It has a class library project which is shared between the client and the server project.

The master node has the following characteristics : 2 virtual processors (CPU : Intel Xeon X5650) with 4GB 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 having each one virtual processor (CPU : Intel QuadCore i5) with 256 MB RAM and 10 GB disk space. The 4 others on a second server having other characteristics : each Virtual Machine has one virtual processor (CPU : Intel Core 2 Duo), 256 MB RAM and 10 GB disk space. All the hardware is connected on a private network behind a switch. All the servers are running Ubuntu 14.04 LTS as operating system.

B. Design

In order to assess the benefits and the trade-offs of 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 IV-A and test them through the scenarios described in Section IV-C. In total we obtained 8 versions of the application as presented in Table I. The basic version E0 don't 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 implements the Local Sharding-Based Router with three sharding algorithms : Modulo, Lookup and consistent hashing. Version E7 implements the Priority Message Queue.

TABLE I
EXPERIMENTAL DESIGNS

Pattern	Algorithm	Code Version
Basic Version		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

Experimentations were orchestrated using the different types of requests described in Table II. For each type of request, we simulated a client sending the request to a server 1000, 2500, 5000, 7500 and 10000 times. Each experimentation was performed five times in order to obtain an average and with different amount of transactions (from 1 to 100 000). It should be noted that the variation between two instances of an experiment never exceeded a few hundreds milliseconds under heavy loads. In the following, we describe the specific experiments that were performed for each pattern.

TABLE II
TYPES OF REQUEST

Read 1	Select a single film
Read 2	Select customer inventory
Random read	Random between simple read 1 and 2
Aggregation	Select overdue DVD's for a customer
Write	Insert a film

Local Database Proxy Pattern : We performed two implementations of this pattern using respectively, the Random Allocation Strategy and the Round-Robin Allocation Strategy [27]. We also implemented a Custom Load Balancing Strategy to test a more reactive strategy.

The proxy is located between server and clients. A first REST web service exposes a set of methods which are hitting the database regarding different algorithms. These methods are used in order to test the local database proxy pattern. The queries are built using parameters such as the ID of a select passed over the REST call. Once the query is built, it is sent to the proxy.

The first work of the proxy is to identify if it is a read or a write query. To do this, it analyses the first word of the query : if it starts with "SELECT" then it is a read query, otherwise it is a write. The next step is to route the query to a slave node. The random algorithm chooses 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 customised algorithm uses two metrics to evaluate the best slave node to

choose : the ping response time between the server and each slave, and the number of active connections on the slaves. To monitor these metrics, a thread is started every 500ms as long as there are queries that has to be executed. Finally, once the slave is chosen, the query is executed and the result is sent back to the function that was called. In order to simplify the tests, we chose to only send back IDs (number identifier), so we don't need to serialize any data. If the result sent from the slave node is null, the query is executed on the master node in order to be sure that the replication did not failed. At last, if the result is null, the response sent to the client has the http *no content* status. If not, the result is sent back to the client using the http *ok response* status.

Local Sharding-based Router Pattern : To test this pattern we used multiple shards hosted separately. Each shard has the same database structure in order to fit with the requirements of sharding algorithms [28]. The first work of the local sharding-based router is to correctly identify which part of the database should be sharded. According to Maxym Kharchenko's Art of Database Sharding [29], we chose two tables of a modified version of the Sakila database [26]. To facilitate the tests, we removed all of the relationships in both the rental and film tables since the sharding is adapted only for independent data.

We chose three commonly used sharding algorithms : 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 server number who will handle the request.

The second sharding algorithm used is the Look-up strategy. This algorithm consists in 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 Pattern : 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 are then processed by 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 performance of the patterns in term of response time and amount of queries executed per second. The result is a tri-dimensional comparison between response time, average number of queries and maximum number of queries executed per second. These measures were taken by the test application itself during every experimentation.

The response time measured in these experiments is the overall response time of the application when executing all the queries. This metric is measured in milliseconds. We choose these metrics because it reflects the capacity of the application to scale with the number of requests. We are only considering results where all the request are processed successfully.

The other metrics are the average and maximum number of queries executed by the application during one second. As we are studying database-related patterns, these metrics are useful to compare the effectiveness of these patterns for database load balancing.

F. Hypotheses

To answer our two 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:

- HR_x^1 : There is no difference between the response time of design Ex and design E0.
- HR_x^2 : There is no difference between the average number of queries processed per second by design Ex and design E0.
- HR_x^3 : There is no difference between the maximum number of queries processed per second by design Ex and design E0.
- HR_{x7}^1 : The response time of the combination of designs Ex and E7 is not different from the response time of each design taken separately.
- HR_{x7}^2 : The average number of queries processed per second by the combination of designs Ex and E7 is not different from the average number of queries processed per second by each design taken separately.
- HR_{x7}^3 : The maximum number of queries processed per second by the combination of designs Ex and E7 is not different from the maximum number of queries processed per second by each design taken separately.

G. Analysis Method

We performed the Mann-Whitney U test [30] to test HR_x^1, HR_x^2, HR_x^3, HR_{x7}^1, HR_{x7}^2, HR_{x7}^3. We also computed the Cliff's δ effect size [31] to quantify the importance of the difference between metrics values. We selected the Cliff's δ effect size because it is reported to be more robust and reliable than the Cohen's d effect size [32]. All the tests are performed using a 95% confidence level (i.e., p-value < 0.05).

Mann-Whitney U test is a non-parametric statistical test that assesses whether two independent distributions are the same or if one distribution 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 [31]. It ranges from -1 (if all selected values in

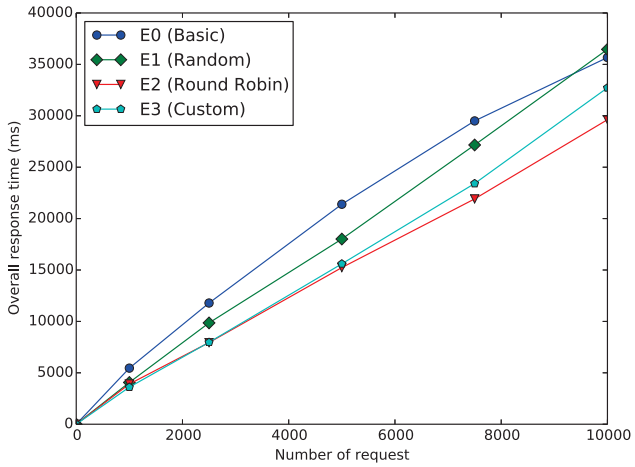

Fig. 1. Select a film with Local Database Proxy

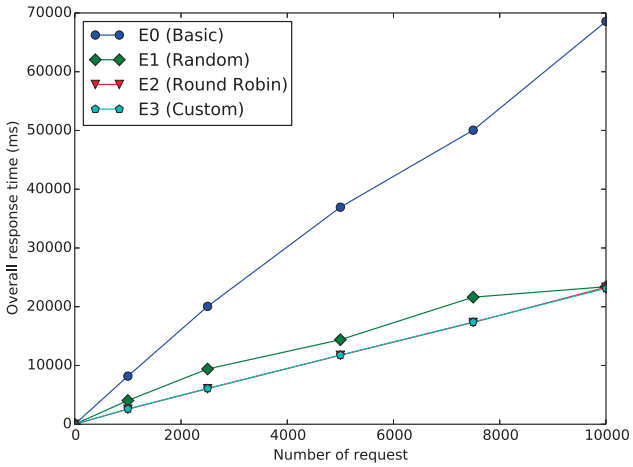

Fig. 2. Select customer inventory with Local Database Proxy

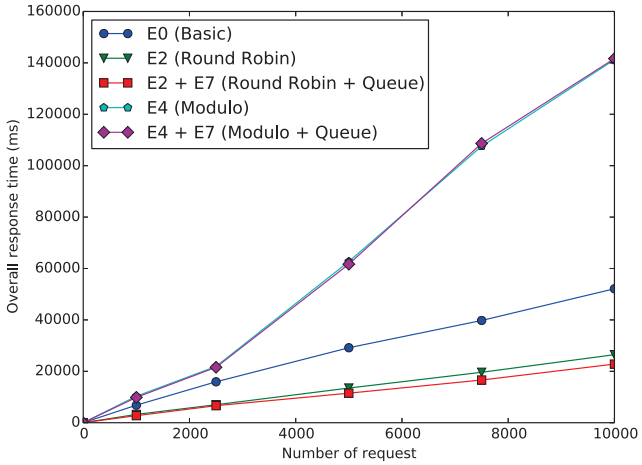

Fig. 3. Random select between film and customer inventory

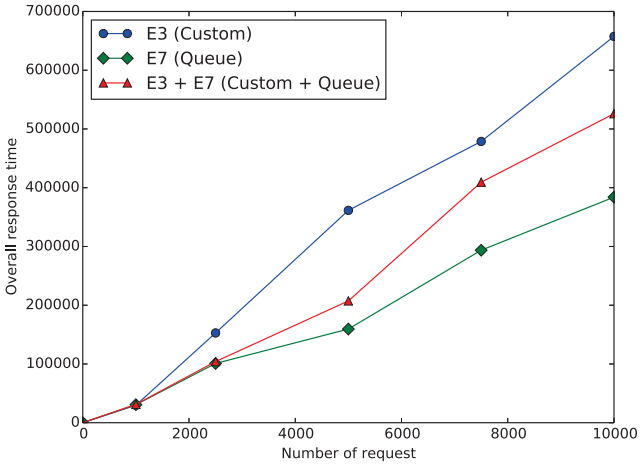

Fig. 4. Insert film with Local Database Proxy and Priority Queue

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). It is zero when two sample distributions are identical [33]. **Interpreting the Effect Sizes** : Cohen's d is mapped to Cliff's δ via the percentage of non-overlap as shown in Table III [31]. Cohen [34] states that a medium effect size represents a difference likely to be visible to a careful observer, while a large effect is noticeably larger than medium.

TABLE III
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

Replication Package All the data collected in our study are publicly available at http://goo.gl/B9upx8.

V. CASE STUDY RESULTS

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

A. Does the implementation of Local Database proxy, Local Sharding-Based Router or Priority Message Queue Patterns affect the QoS of cloud applications?

Table IV and V summarises the results of Mann–Whitney U test and Cliff's δ effect sizes for each metrics. Significant results are marked in bold.

Response time : Results of Table IV and V show that there is no statistically significant difference between the overall response time of applications implementing the studied patterns and the application not implementing any of the three patterns, hence we cannot reject HR_x^1 for all Ex ($x \in \{1 \dots 6\}$). However, Figures 1 to 6, as well as effect size values show that all three patterns have a slightly positive impact on the response time of the applications (*i.e.,* the response time is lower), in all the scenarios with the exception of Local

TABLE IV
p-VALUE OF MANN–WHITNEY U TEST AND CLIFF'S δ EFFECT SIZE FOR RANDOM SELECTS BETWEEN FILM AND CUSTOMER INVENTORIES

	Overall Response Time		Average Query per Second		Maximum Query per Second	
	p-value	Effect Size	p-value	Effect Size	p-value	Effect Size
E0, E1	1.68×10^{-1}	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E2	1.68×10^{-1}	-0.68	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E3	1.68×10^{-1}	-0.68	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E4	2.11×10^{-1}	0.48	**<0.05**	**-1**	**<0.05**	**-1**
E0, E5	2.11×10^{-1}	0.48	**<0.05**	**-1**	**<0.05**	**-1**
E0, E6	2.11×10^{-1}	0.48	**<0.05**	**-1**	**<0.05**	**-1**
E0, E7	2.61×10^{-1}	-0.44	**<0.05**	**0.72**	7.18×10^{-2}	0.48
E1, E1 + E7	3.15×10^{-1}	-0.28	7.44×10^{-2}	0.60	**<0.05**	**0.80**
E2, E2 + E7	3.74×10^{-1}	-0.20	7.12×10^{-2}	0.72	**<0.05**	**0.84**
E3, E3 + E7	3.74×10^{-1}	-0.20	1.05×10^{-1}	0.64	**<0.05**	**0.84**
E4, E4 + E7	5×10^{-1}	-0.00	3.37×10^{-1}	-0.20	4.58×10^{-1}	0.08
E5, E5 + E7	4.36×10^{-1}	-0.12	4.16×10^{-1}	-0.04	2.28×10^{-1}	0.32
E6, E6 + E7	5×10^{-1}	-0.08	5×10^{-1}	0.12	7.06×10^{-2}	0.72

TABLE V
p-VALUE OF MANN–WHITNEY U TEST AND CLIFF'S δ EFFECT SIZE FOR INSERT A FILM

	Overall Response Time		Average Query per Second		Maximum Query per Second	
	p-value	Effect Size	p-value	Effect Size	p-value	Effect Size
E0, E1	4.36×10^{-1}	0.16	2.33×10^{-1}	-0.40	2.05×10^{-1}	-0.44
E0, E2	4.36×10^{-1}	0.16	2.83×10^{-1}	-0.32	1.94×10^{-1}	-0.44
E0, E3	4.36×10^{-1}	0.16	2.33×10^{-1}	-0.40	2.84×10^{-1}	-0.32
E0, E4	1.68×10^{-1}	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E5	1.68×10^{-1}	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E6	1.68×10^{-1}	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E7	3.15×10^{-1}	-0.08	3.15×10^{-1}	0.28	1.57×10^{-1}	0.52
E1, E1 + E7	4.36×10^{-1}	-0.12	1.30×10^{-1}	0.60	**<0.05**	**-1**
E2, E2 + E7	4.36×10^{-1}	-0.12	1.30×10^{-1}	0.60	3.71×10^{-1}	0.32
E3, E3 + E7	4.36×10^{-1}	-0.12	1.30×10^{-1}	0.60	3.14×10^{-1}	-0.12
E4, E1 + E4	1.31×10^{-1}	-0.6	1.22×10^{-1}	0.76	1.31×10^{-1}	0.76
E5, E1 + E5	2.11×10^{-1}	-0.44	9.81×10^{-2}	0.84	1.30×10^{-1}	0.76
E6, E1 + E6	2.11×10^{-1}	-0.44	8.22×10^{-2}	0.88	**<0.05**	**0.84**

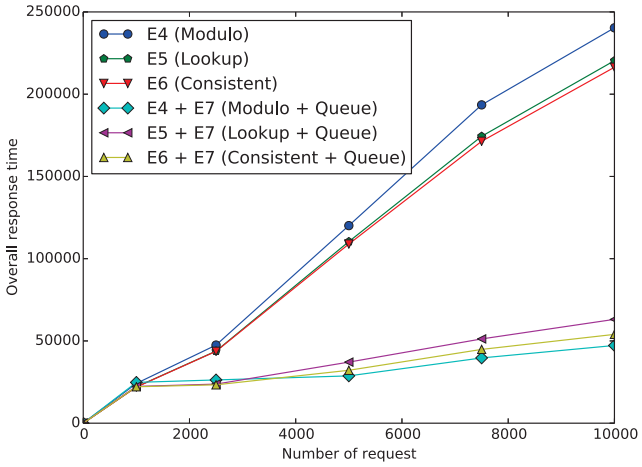

Fig. 5. Insert film with Local Sharding-Based Router and Priority Queue

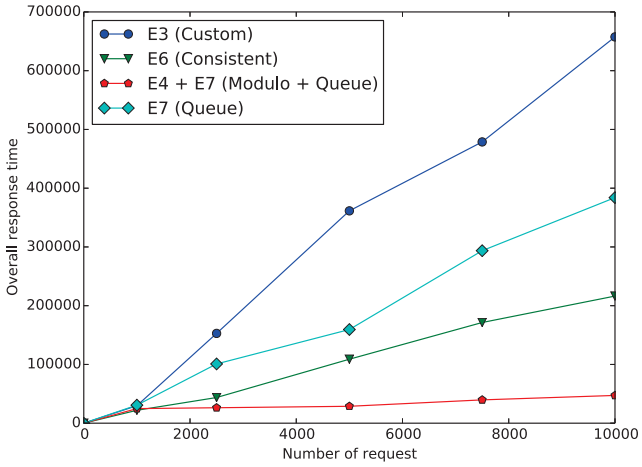

Fig. 6. Insert film

Sharding-Based Router on read requests (see E4 on Figure 3). Also, Figures 1 to 6 show that the impact of these patterns increases with the number of requests, suggesting that:

When the number of requests is very large, Local Database proxy and Priority Message Queue Patterns can have a positive impact on the response time of an application.

Average number of query processed per second : Results of Table IV and V show that for *random selects between film and customer inventories*, there is a statistically significant difference between the average number of query processed per second by applications implementing the studied patterns and the application not implementing any of the three patterns, and the effect size is large. Hence we reject HR^2_x for all Ex ($x \in \{1 \dots 6\}$). We also obtained statistically significant results with *read requests*, for all implementations of Local Database proxy and Priority Message Queue. By contrast, we obtained lower numbers of requests processed per second with the Local Sharding-Based Router. We explain this result by the overhead induced by the sharding algorithms. For *write requests*, we obtained statistically significant results only for designs E4, E5 and E6 (the effect size is large); hence we reject HR^2_4, HR^2_5, and HR^2_6.

> ***Overall, results show that Local Database proxy and Priority Message Queue can increase the average number of query processed per second by an application. This increase is statistically significant in most cases.***

Maximum number of query processed per second : Results for the maximum number of query processed per second are similar to the results obtained for the average number of query processed per second, except for the Priority Message Queue (see IV and V).

In general, we can conclude that Local Database proxy, Local Sharding-Based Router and Priority Message Queue Patterns have a positive impact on the ability of applications to handle heavy loads of *read and write queries*, as suggested by the literature [5], [7]. More specifically, the Local-Database Proxy is a good design solution for applications experiencing heavy loads of *read requests*, while the Local Sharding-Based Router is more adequate for applications handling huge *write requests* loads. The Priority Message Queue pattern has only a moderate effect on both types of requests.

The results of our study also show that the load balancing and sharding algorithms implementing the patterns also affect the QoS of the applications. Round Robin and Consistent Hashing algorithms produced the best results in all our experimentations. However, given the small differences in effect sizes observed among the different variants of the patterns (*i.e.,* with different algorithms), it appears that:

> ***The choice of pattern is more important than the choice of a particular algorithm (for the implementation of the pattern) since it has a bigger impact on the QoS.***

B. Do interactions among Local Database proxy, Local Sharding-Based Router and Priority Message Queue Patterns affect the QoS of cloud applications?

Regarding response time, results from Figures 3 and 5 and Table IV and V show that the addition of the Message Queue pattern to an application implementing Local Database proxy or Local Sharding-Based Router patterns does improve the overall response time of the application, but this improvement is not statistically significant. Therefore, we cannot reject HR^1_{x7} for all Ex ($x \in \{1 \dots 6\}$).

Regarding the number of queries processed per second, we obtained significant differences between the maximum number of queries processed per second by designs E1 + E7, E2 + E7, and E3 + E7, when performing *random selects between film and customer inventories*. We reject HR^3_{17}, HR^3_{27}, and HR^3_{37} in this case.

> ***A combination of the Priority Message Queue pattern with Local Database proxy or Local Sharding-Based Router patterns can improve the QoS of an application experiencing heavy loads of read and write requests. More analysis are desirable to better understand the interplay between these patterns.***

C. Threats to Validity

In this section, we discuss the threats to validity of our study based on the guidelines provided by Wohlin *et al.* [35].

Construct validity threats concern the relation between theory and observations. In this study, they could be due to measurement errors. We instrumented the different versions of the application described in Section IV-A, to generate execution logs from which we computed response time, average, and maximum numbers of queries processed per second. 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.

Internal validity concern our selection of subject systems and analysis methods. Despite the usage of a well known benchmark (the Sakila sample database [26]) 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 applications.

External validity threats concern the possibility to generalise our findings. Further validation should be done on different cloud applications and with different cloud patterns to broaden our understanding of the impact of cloud patterns on the QoS of applications. One major challenge however is the difficulty to find open-source applications running on the cloud, and in which the studied patterns are implemented. It is because of this limitation that we implemented a complete cloud based application for the purpose of our study.

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[1].

Finally, the *conclusion validity* threats refer to the relation between the treatment and the outcome. We paid attention not

[1] http://goo.gl/B9upx8

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 and future work

Cloud patterns are always described in the literature as good practices, without considering applications contexts and interactions with other patterns. In this paper, we performed 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 these patterns on the QoS of the application through measurements of the overall response time, the average and maximum number of requests processed by the application per second. Results show that these patterns and their combinations can increase the QoS of applications. The Local Database proxy is more adapted for applications experiencing heavy loads of *read requests*, while the Local Sharding-Based Router is more appropriate for applications handling huge *write requests* loads. The Priority Message Queue pattern has only a moderate effect on both types of requests. The impact of Priority Message Queue is larger under heavy loads, especially on the average number of requests processed per second. We also found that Round Robin and Consistent Hashing algorithms are good implementation choices for Local Database proxy and Local Sharding-Based Router patterns, respectively. However, the choice of a pattern seems to have a bigger effect on the QoS than the choice of a particular algorithm. These results provide important guidelines for software organisations developing and deploying cloud based applications with MSQL databases. In the future, we plan to expand our study to investigate a broader variety of cloud applications and more cloud patterns. We also plan to investigate other aspects of the sustainability of cloud applications, such as the energy consumption.

References

[1] K. Beck and W. Cunningham, "Using Pattern Languages for Object-Oriented Programs," Tech. Rep., Sep. 1987.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[3] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 2013, vol. 2.

[4] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, and T. Newling, *Patterns: service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization, 2004.

[5] 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.

[6] J. Varia, "Architecting for the cloud: Best practices," *Amazon Web Services*, 2010.

[7] S. Strauch, V. Andrikopoulos, U. Breitenbuecher, O. Kopp, and F. Leymann, "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.

[8] S. Strauch, V. Andrikopoulos, U. Breitenbücher, S. Gómez Sáez, O. Kopp, and F. Leymann, "Using patterns to move the application data layer to the cloud," in *PATTERNS 2013, The Fifth International Conferences on Pervasive Patterns and Applications*, 2013, pp. 26–33.

[9] D. Petcu, "Identifying cloud computing usage patterns," in *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–8.

[10] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Publishing Company, Incorporated, 2014.

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

[12] 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.

[13] 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.

[14] 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.

[15] S. S. Alwakeel and H. Almansour, "Modeling and performance evaluation of message-oriented middleware with priority queuing," *Information Technology Journal*, vol. 10, no. 1, pp. 61–70, 2011.

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

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

[18] F. Khomh and Y.-G. Guéhéneuc, "Do design patterns impact software quality positively?" in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*. IEEE, 2008, pp. 274–278.

[19] P. Wendorff, "Assessment of design patterns during software reengineering: Lessons learned from a large commercial project," in *Software Maintenance and Reengineering, 2001. Fifth European Conference on*. IEEE, 2001, pp. 77–84.

[20] M. Vokáč, W. Tichy, D. I. Sjøberg, E. Arisholm, and M. Aldrin, "A controlled experiment comparing the maintainability of programs designed with and without design patterns—a replication in a real programming environment," *Empirical Software Engineering*, vol. 9, no. 3, pp. 149–195, 2004.

[21] A. Ampatzoglou and A. Chatzigeorgiou, "Evaluation of object-oriented design patterns in game development," *Information and Software Technology*, vol. 49, no. 5, pp. 445–454, 2007.

[22] L. Prechelt, B. Unger, W. F. Tichy, P. Brossler, and L. G. Votta, "A controlled experiment in maintenance: comparing design patterns to simpler solutions," *Software Engineering, IEEE Transactions on*, vol. 27, no. 12, pp. 1134–1144, 2001.

[23] K. Aras, T. Cickovski, and J. A. Izaguirre, "Empirical evaluation of design patterns in scientific application," 2005.

[24] 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.

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

[26] "Mysql sakila sample database," http://dev.mysql.com/doc/sakila/en/, 2014.

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

[28] "Sharding algorithms," http://kennethxu.blogspot.fr/2012/11/sharding-algorithm.html, November 2012.

[29] M. Kharchenko, "The art of database sharding," 2012.

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

[31] 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.

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

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

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

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

Title

An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS) and Energy Consumption (Green IT)

Key-words

Cloud Patterns, Replication, Sharding, Priority Queue, Energy Consumption

Abstract

Cloud patterns are described as good solutions to recurring problems in a cloud context: lowering the response time, having a low latency, being scalable and having a high availability. However, there is a lack of studies around their impact on energy consumption which is nowadays a hot topic. During this internship, we conducted a study on a RESTful application deployed in the cloud to investigate the individual and the combined impact of three cloud patterns (i.e. Local Database Proxy, Local Sharding Based Router and Priority Queue Patterns) on Energy Consumption. This study follows a recent study about the impact of these three patterns on Quality of Service (QoS). We measure the Energy Consumption using PowerAPI, a framework designed in Scala by the INRIA to monitor energy consumption of processes such as MySQL. Results show that design patterns and their different implementations have an impact on the energy consumption of the cloud application.
Developers and software architects can make use of these results to guide their design decisions: between QoS and energy consumption, there is a trade-off to make.

Titre

Une étude empirique de l'impact des patrons de conception Cloud sur la qualité de service (QoS) et la consommation d'énergie (Green IT)

Mots-clés

Patron de Conception Cloud, Réplication, Sharding, Queue de Priorité, Consommation énergétique

Résumé

Les patrons de conception de génie logiciel pour le Cloud sont décrits comme étant de bonne solution pour répondre à des problèmes récurant liés au Cloud comme la latence, le temps de réponse, la scalabilité mais aussi la disponibilité. Cependant, on note un manque de recherche en ce qui concerne l'impact de ces patrons sur la consommation énergétique qui est un sujet très discuté actuellement. Pendant ce stage, nous avons mené une étude sur une application REST déployée dans le Cloud pour pouvoir étudier l'impact de trois patrons (i.e. Local Database Proxy, Local Sharding Based Router and Priority Queue Patterns) sur la consommation énergétique de façon individuel mais aussi en les combinant. Cette étude fait suite à une précédente étude s'intéressant à l'impact de ces trois patrons sur la Qualité de Service. Nous avons mesuré la consommation énergétique de notre application en utilisant PowerAPI, un framework développé en Scala par l'INRIA et permettant de suivre la consommation énergétique de processus comme MySQL. Les résultats ont montré que les patrons de conceptions et leurs différentes implémentations ont un impact sur la consommation énergétique de l'application cloud.
Les développeurs et ingénieurs logiciels peuvent utiliser ces résultats pour prendre des décisions dans leurs choix de conception: entre la performance et la consommation énergétique, il y a un compromis à faire.