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

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 Cloud Computing.

Most cloud patterns like *Proxy Service*, *message queue* or *Composed Service* were adopted from SOA or parallel computing [3]. 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 [4].

Despite several benchmarks and studies [5]–[7] 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. Section II presents background information and related works on the impact of design patterns. Section III presents the design of our experiments and section IV discusses the obtained results. Section V concludes our study and outlines some avenues for future works.

II. Related Work

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. We also discuss the relevant literature about cloud patterns evaluation.

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 [8]. All write requests are handled by the master and replicated on its slaves while read requests are processed by slaves. Unlike usual replication mechanisms where application components access a predetermined replica [9], with this pattern, components must use a local proxy whenever they need to retrieve or write data. Using the Local Database Proxy pattern, Microsoft [9] 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.

Local Sharding-Based Router : The Local Sharding-Based Router is recommended when the need for scalability concerns read and write operations [8]. Data are split among multiple databases into functional groups called shards, requests are

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.141

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. 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 [9].

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. Messages with high priority values are received and processed more quickly than those with lower priority values [9].Message Queues are considered good practices for cloud applications, to design loosely coupled components and to improve scalability [4].

Evaluation of Cloud Patterns : Ardagna *et al.* [10] 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. Burtica *et al.* [6] 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 [5] 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. The performance of priority queues has been evaluated by Alwakeel *et al.* [7]. In this work, the message queue is evaluated as a technical solution in isolation, without considering application's context.

III. 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 patterns were implemented with different algorithms which are explained in section III-C. We analyzed eight versions of the application, 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 because it's one of the most popular database for Cloud applications [11]. We use the Sakila sample database [12] 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.The test application was fully developed using Java and is composed of about 3,500 lines of code and its size is 6 MB.

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 III-A and test them through the scenarios described in Section III-C. In total we obtained 8 versions of the application as presented in Table I. The basic version E0 don't use any pattern.

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 (read, write and aggregation). 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.

Local Database Proxy Pattern : We performed two implementations of this pattern using respectively, the Random Allocation Strategy and the Round-Robin Allocation Strategy [13]. 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 by analyzing 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. A thread monitors these metrics 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 [14]. 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 [15], we chose two tables of a modified version of the Sakila database [12]. 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.

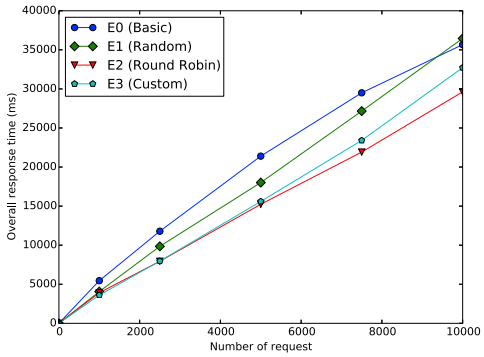

Fig. 1. Select a film with Local Database Proxy

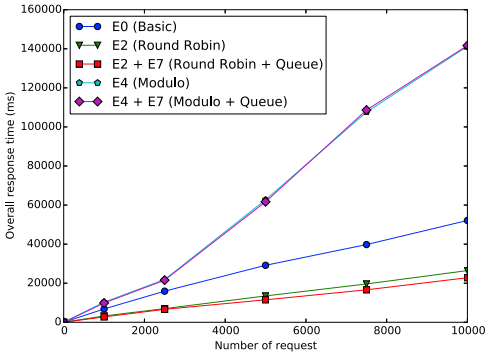

Fig. 2. Random select between film and customer inventory

- HR^1_{x7} : The response time of the combination of designs Ex and E7 is not different from the response time of each design taken separately.
- HR^2_{x7} : 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^3_{x7} : 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 [16] to test HR^1_x, HR^2_x, HR^3_x, HR^1_{x7}, HR^2_{x7}, HR^3_{x7}. We also computed the Cliff's δ effect size [17] to quantify the importance of the difference between metrics values. 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. Cliff's δ is a non-parametric effect size measure which represents the degree of overlap between two sample distributions [17]. It 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). It is zero when two sample distributions are identical [18].

IV. 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 II 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 II 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^1_x for all Ex ($x \in \{1 \dots 6\}$). However, Figures 1 to 4, 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 Sharding-Based Router on read requests (see E4 on Figure 2). Also, Figures 1 to 4 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.***

TABLE II
p-VALUE OF MANN–WHITNEY U TEST AND CLIFF'S δ EFFECT SIZE (ES)

	Overall Response Time		Average Query/s		Maximum Query/s	
	p-value	ES	p-value	ES	p-value	ES
Random select between film and customer inventory						
E0, E1	0.17	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E2	0.17	-0.68	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E3	0.17	-0.68	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E4	0.21	0.48	**<0.05**	**-1**	**<0.05**	**-1**
E0, E5	0.21	0.48	**<0.05**	**-1**	**<0.05**	**-1**
E0, E6	0.21	0.48	**<0.05**	**-1**	**<0.05**	**-1**
E0, E7	0.26	-0.44	**<0.05**	**0.72**	0.07	0.48
E1, E1+E7	0.32	-0.28	0.07	0.60	**<0.05**	**0.80**
E2, E2+E7	0.37	-0.20	0.07	0.72	**<0.05**	**0.84**
E3, E3+E7	0.37	-0.20	0.11	0.64	**<0.05**	**0.84**
E4, E4+E7	0.50	-0.00	0.34	-0.20	0.46	0.08
E5, E5+E7	0.44	-0.12	0.42	-0.04	0.23	0.32
E6, E6+E7	0.50	-0.08	0.50	0.12	0.07	0.72
Insert a film						
E0, E1	0.44	0.16	0.23	-0.40	0.20	-0.44
E0, E2	0.44	0.16	0.28	-0.32	0.19	-0.44
E0, E3	0.44	0.16	0.23	-0.40	0.28	-0.32
E0, E4	0.17	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E5	0.17	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E6	0.17	-0.52	**<0.05**	**0.80**	**<0.05**	**0.80**
E0, E7	0.32	-0.08	0.31	0.28	0.16	0.52
E1, E1+E7	0.44	-0.12	0.13	0.60	**<0.05**	**-1**
E2, E2+E7	0.44	-0.12	0.13	0.60	0.37	0.32
E3, E3+E7	0.44	-0.12	0.13	0.60	0.31	-0.12
E4, E1+E4	0.13	-0.6	0.12	0.76	0.13	0.76
E5, E1+E5	0.21	-0.44	0.10	0.84	0.13	0.76
E6, E1+E6	0.21	-0.44	0.08	0.88	**<0.05**	**0.84**

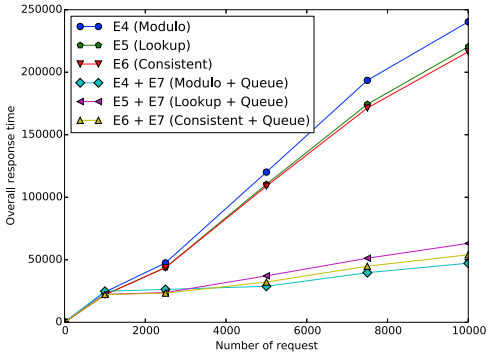

Fig. 3. Insert film with Local Sharding-Based Router and Priority Queue

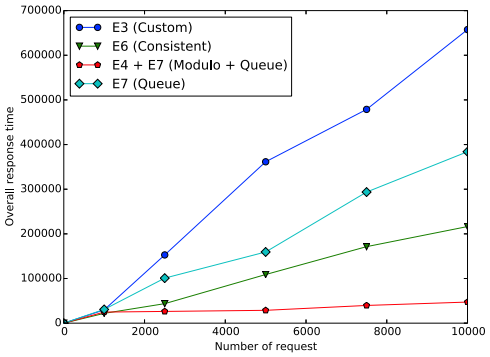

Fig. 4. Insert film

Average number of query processed per second : Results of Table II 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_x^2 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_4^2, HR_5^2, and HR_6^2.

> ***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 II).

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 [8], [9]. 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 2 and 3 and Table II shows 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_{x7}^1 for all Ex ($x \in \{1 \dots 6\}$). Figures 2 and 3 show a positive impact when Priority Message Queue is combined with others designs.

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_{17}^3, HR_{27}^3, and HR_{37}^3 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.* [19].

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 III-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 [12]) 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 (http://goo.gl/B9upx8).

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.

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

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

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

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

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

[17] 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.

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

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