Quality of Service in Cloud Computing Environments with Multitenant DBMS

Manuel I. Capel[1][a], Oscar I. Aporta[1] and María C. Pegalajar-Jiménez[2][b]

[1]*Department of Software Engineering, ETSI Informática and Telecomunicación, University of Granada, Periodista Daniel Saucedo Aranda s/n, 18015 Granada, Spain*

[2]*Department of Computer Science and AI, ETSI Informática and Telecomunicación, University of Granada Periodista Daniel Saucedo Aranda s/n, 18015 Granada, Spain*

Keywords: Multitenancy, Multitenant DBMS, Quality of Service, DBMS Located in the Cloud, IaaS, OpenNebula.

Abstract: This article proposes a new study of Quality of Service (QoS) in Database Management Systems with multitenancy in which it is experimentally verified that tenants follow interference patterns between them when they concurrently access the DBMS. The interference degree depends on characteristics of the database used by each tenant. A testing architecture with virtual machines (VM), managed with OpenNebula, has been designed. In each VM one DBMS is loaded managing many databases, one for each tenant. Five experiments were designed and numerous measurements performed using benchmarks of reference, such as TPCC, in a Cloud computing-based system. The results of the experiments are presented here, for which the latency and performance were measured with respect to different workloads and tenant configurations. To carry out the experiments, a multitenant environment model known as shared database/separate schema or *shared instance* was deployed, which is widely used at moment and presents the best ratio between resource use, performance and response.

1 INTRODUCTION

The requirements of today's society and the new ITC paradigms that dominate in a general way explain that software development tends to be increasingly flexible, dynamic and personalized, accessible off-premise through the Internet, without the need to be installed and managed on-premise. Virtualization is one of the fundamental technologies to make this new approach possible for the development and provision of software services, since it allows a variety of applications, which function as dedicated software, to be grouped into a set of shared resources that help to improve the use of physical resources, to simplify management and reduce costs for companies.

A tenant is defined according to the context in which it is inserted, for example, a tenant can be a user of the application or a particular database in relation to a DBMS.

A multitenant approach can help us consolidate applications composed of multiple simultaneous versions into a single functional system, thus avoiding the inefficiency of having a separate system for each tenant (Benjamin et al., 2011). DBMS are potential candidates for implementation in a multitenant Cloud Computing environment, thus promoting scalability, cost reduction, ease of configuration, availability of on-demand services, etc. Currently, multitenant DBMS have been used to host multiple tenants within a single system, and thus allowing the efficient sharing of resources at different levels of abstraction and isolation (Agrawal et al., 2011a).

We propose here a software architecture and a performance evaluation methodology to carry out a Quality of Service (QoS) study in Database Management Systems (DBMS) for multitenant environments in the Cloud.

Cloud service providers have to solve several challenges, such as availability, performance, scalability and elasticity, to meet the quality of service required by customers of multitenancy systems in the Cloud. A possible solution to achieve this is to automatically manage the available resources and the workload of the system to obtain elasticity and improve the use of these resources (Sousa et al., 2011), but this solution usually causes throughput gets worse due to the in-

[a] https://orcid.org/0000-0003-2449-4394
[b] https://orcid.org/0000-0001-9408-6770

Capel, M., Aporta, O. and Pegalajar-Jiménez, M.
Quality of Service in Cloud Computing Environments with Multitenant DBMS.
DOI: 10.5220/0009794605060514
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 506-514
ISBN: 978-989-758-424-4

crease in the response time of tenants' requests. In particular, to solve the drawback caused by the high degree of concurrency among the tenants of a DBMS, there is a strategy that consists of distributing the location of tenants in virtual machines according to their individual interference pattern. To apply that means performing the following tasks: (a) analysis of the tenant's profile to determine the level of interaction with other tenants, (b) dynamic assignment of tenants to different virtual machines without degrading the system response and (c) application of techniques to efficiently migrate tenants which show a lot of interference.

This article, therefore, has focused on conducting a systematic study to determine interference profile of each tenant by using metrics such as performance, latency and response time. The study has to be considered a preliminary stage in the development of a method and software tool that allow automatic migration of tenants and DBMS. The experiments carried out mix different workload/tenant configurations and measure their latency and throughput by assuming one DBMS per VM, many databases (one database per tenant) managed by one DBMS and VM, which are managed using OpenNebula.

The article structure is as follows. First the quality of service (QoS) model used to comply with the SLA (client requirements) of a multitenant DBMS system is presented. In the third section, the software architecture and the implementation of the execution environment to conduct the study are discussed. In the fourth section the design of the testing experiments carried out is detailed. The fifth section is dedicated to the measures obtained in the study and discussion of results. Finally, a section of conlusions and future work is included.

2 A PROPOSAL FOR OBTAINING QoS OF A MULTITENANCY DBMS IN THE CLOUD

In multitenant models where each tenant only needs a fraction of the resources, the degree of concurrency of multiple tenants is quite high and makes difficult to guarantee the quality of service (QoS), which is usually defined by means of an SLA with the user. Cloud service providers have to solve several challenges to meet their required availability to customers of the service.

Scalability is a static property of the system that specifies its behavior with respect to a given configuration, and elasticity is the dynamic property that allows to scale the system when there is a variation in demand, while the system is in operation (Agrawal et al., 2011b). Therefore, the property of elasticity is the most important to achieve quality of service by Cloud service providers, who try to obtain an estimate of the workloads expected to be reached, in order to be able to perform a proactive management of the resources.

We select a multitenancy model that presents the best relationship between the maximum degree of resource sharing and the least number of interferences between tenants that may concur in the DBMS. The selected multitenant model is that of shared database/separate schema or *shared instance*, since it is the most widely used and present the best relationship between resource utilization, performance and security (Barker et al., 2012). According to this model, each virtual machine has an instance of the DBMS, and each DBMS contains a variable number of tenants, depending on the capacity of the virtual machine resources and the workload. A tenant is represented by a database in the system. Despite requiring less infrastructure resources, this model increases the interference between tenants, since there will be a greater number of tenants in the same DBMS.

2.1 Proposal of DBMS Model with Multitenancy

Our DBMS model in the Cloud with QoS was built according to the requirements of elasticity, scalability and efficient use of shared resources.

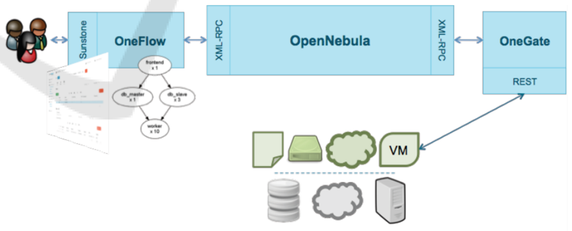


Figure 1: OpenNebula-based Cloud architecture.

To meet the above requirements, the multitenant system proposal is structured in the following components,

1. *Autonomic Manager*: Highly scalable monitoring system that can interpret a computation to collect the relevant parameters, regarding migration of tenants.
2. *Predictive Models*: allow to anticipate software and hardware failures.
3. *Managed Resources*: Cloud management platform for autonomous services that provides self-

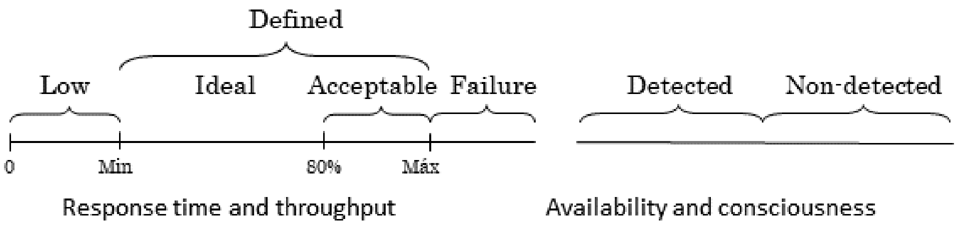


Figure 2: Determination of limits of acceptance for Q-attributes in a DBMS w.r.t. SLA during monitoring of services execution, from (2011:Sousa).

awareness and self-configuration through sensors and actuators to take proactive measures and decisions of reconfiguration (replication, migration).

4. *Task Assignment System based on QoS*: distributes the workload based on the current state of available resources. The allocation system is a load balancer based on QoS.

We choose OpenNebula middleware (see Figure 1), which manages physical and virtual resources (nodes, networks, virtual machines, images, etc.), to perform the functions of a stand-alone administrator ("Autonomic Manager"), while OpenNebula's *OneFlow* service is the service administrator ("Managed Resources") that manages services automatically, including elasticity. Self-awareness and self-configuration are achieved by extending the *OneGate* component provided by OpenNebula.

Users and administrators use *OneGate* to collect metrics, detect problems in applications, and trigger elasticity rules in *OneFlow*. OpenNebula interacts with *OneGate*, through its XML-RPC interface, to send monitoring metrics to virtual machines. *OneGate* is the principal mechanism used to exchange information between virtual machines and *OneFlow*.

2.2 Quality Model of the DBMS According to the Service Level Agreement

The service level contract (SLA) contains information related to the functional and non-functional requirements that the service provider must guarantee and the penalties in case of non-compliance. The definition of an SLA is not a trivial task and should consists of information about stakeholders, the SLA's parameters, an algorithm to calculate these and the metrics used, service level objective (SLO) and the actions to be performed in case of violation of the agreement (Schnjakin et al., 2010). That information must be specified as characteristics and attributes of quality in a structured and guided way, by means of a SLA specification language, such as one of the languages: WSLA, WSOL or SLAng.

According to (Chi et al., 2011), SLA metrics for database in Cloud Computing should optimize the system, address relevant aspects for data management and contemplate the characteristics of the Cloud Computing model, such as *elasticity*, *scalability* and *multitenancy*. For (Schoroeder et al., 2010) it is important to establish more general criteria to evaluate QoS, such as the percentile *x*% within which the response time is less than a given value *y*. In (Sousa et al., 2011) (Moreira et al., 2013) we find SLADB, an example of a quality model, which includes a definition of SLA, monitoring techniques and penalties. Each metric has a quality of service level objective (SLO) associated with it, as indicated below,

- *Response Time*: percentile x% of response times less than a certain value and for a period of time *t*
- *Throughput*: percentile z% of throughput values greater than a *k* value and for a period of time *t*
- *Availability*: function detected/non-detected for indicating the existence of queries rejected over a period of time
- *Consciousness*: function detected/non-detected for indicating whether updated data are accessed according to the consistency type (strong or weak)

In a QoS model for DBMS based on states, to comply with the SLA, the acceptable limits of the above parameters must be defined to perform a correct monitoring, as shown in Figure 2.

2.3 QoS Metrics for a Multitenant DBMS

The proposed multitenant DBMS infrastructure presents a set of objectives included in the SLA, which have associated metrics that allow measuring the quality of the service. The System Level Objective (SLO) contains the predetermined limits for the parameter to be measured and, for each parameter, a way of calculating how it is defined, e.g., by computing the *mean time*. We define a Function of Aptitude (FA) that will help us in the decision process regarding the distribution of workloads, thus allowing us to verify the allocation of a load to a virtual machine (VM). FA will give us a criterion to determine whether the instance of the DBMS loaded in the VM is suitable to receive a new workload or not.

We calculate the FA(i) for each VM(i) that will reflect the percentage of work capacity of each VM (i) with respect to the rest of the VMs. The function must be defined according to the capacity of the resource and the efficiency,

$$\begin{aligned} FA(i) &= a*cp(i)+b*cm(i)+c*tr(i) \ \ where: \\ cp(i) &= \ CPU\,capacity\,available\,for\,the\,VM(i) \\ cm(i) &= \ Available\,memory\,capacity\,of\,the\,VM(i) \\ tr(i) &= \ Response\,time\,of\,the\,VM(i) \\ \{a, b, c\} &\ are\,the\,weigths \end{aligned}$$

Where each of the three parameters of FA will be multiplied by a weight depending on the importance of this factor in the multitenant system.

3 ARCHITECTURE OF THE SOFTWARE AND IMPLEMENTATION

We will introduce in this section the OpenNebula middleware (Fontan et al., 2008), which is the management platform of Cloud computing selected for this study. We will show how physical resources are orchestrated and virtualized. We use OpenNebula in this study because it makes possible the construction of any type of cloud computing system: private, public or hybrid, which is fundamental for the administration of heterogeneous infrastructures of distributed data centers. It includes features for integration, administration, planning, management, scalability, security and data center accounting. Its core is very efficient and is fully developed in C ++ with a backend for highly scalable database management systems, including support for MySQL and SQLite.

The software architecture of OpenNebula provides interfaces that allow interaction with physical resources as well as virtual resources, such as the following: i) Interfaces for consumers and administrators of Cloud Computing, with several available APIs: AWS EC2, EBS and OGF OCCI. In hybrid configurations, it has adapters for Amazon EC2 Cloud services and ElasticHosts. ii) Management interfaces for advanced users and operators of Cloud Computing, such as the UNIX-based command line interface (ONE CLI), and its own graphical GUI from Sunstone, which serves as a multi-user portal and manager of resources for advanced users. iii) Low level extensible APIs for Cloud integrators for Java, Ruby, XMLRPC API. iv) OpenNebula Marketplace is an online catalog that offers a wide variety of applications capable of running in OpenNebula environments.

In our study we chose to use a multitenant model of *shared instance*, so that, according to this model, a correspondence between a database and a tenant in the system can be defined; each DBMS contains a variable number of tenants and each virtual machine maintains a single instance of the DBMS (see Figure 3).

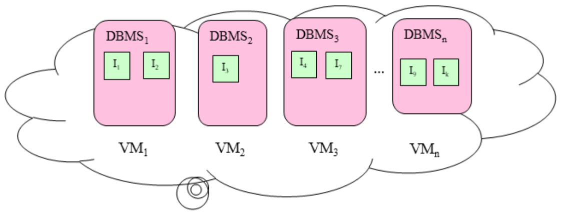


Figure 3: ‘Shared instance’ multitenant DBMS model used in the study.

For the evaluation of the model, we used a private cloud implemented with the OpenNebula middleware. This allows us to focus the study’s attention on analyzing the interference between tenants after getting rid of external factors (network latency, unavailability of a public cloud, etc.) that could affect the results of our measurements. In relation to the evaluation tool, we opted for MuTeBench, since it is the first specific tool for this type of systems that currently exists, at least, until our best knowledge at the time of publication. MuTeBench allows you to simulate a complete multitenant environment. To carry out the measurements, we created 2 virtual machines (VM-MuTeBench, VM-mySQL) each one running a Ubuntu 16.04 LTS distro. VM-MuTeBench is deployed in 2 CPUs, 8 GB of RAM, 30 GB of storage, and contains the MuTeBench framework. In its turn, the VM-mySQL virtual machine is deployed in 2 CPUs, 4 GB of RAM, 30 GB of storage and runs MySQL 5.7 DBMS, with the InnoDB engine, and 128 MB of buffer memory. The databases provided by MuTeBench for the realization of the TPCC, YCSB and Wikipedia benchmarks, were located in the VM-mySQL virtual machine.

4 DESIGN OF TESTING EXPERIMENTS

In a multitenant architecture the data layer is essential and an important issue is how a tenant’s workload

Table 1: Set of benchmarks accepted in OLTP-Bench.

Class	Benchmark	Application domain
Transactional	AuctionMark	Auctions on line
	CH-benCHmark	OLTP and OLAP mix
	SEATS	On-line airline ticketing
	SmallBank	Banking system
	TATP	Call location application
	TPCC	Processing order
	Voter	Talent sample voting
Web oriented	Epinions	Social networks
	Twitter	Social networks
	Wikipedia	Online encyclopedia
Functional test	ResourceStresser	Isolate resources stress-test
	YCSB	Scalable store of key-value pairs
	JPAB	Relational assignment of objects
	SIBench	Transactional isolation

Table 2: Databases used in the experiments.

Multiple tenants running the benchmatk	Size (MB)
TPCC	500 MB
YCSB	800 MB
Wikipedia	600 MB

interferes with the rest of tenants sharing a resource. The evaluation of multiple tenants in the Cloud differs completely from the methods used in a traditional evaluation of DBMSs, being necessary to use specific benchmarks for these environments that may have the ability to execute in parallel and changing workloads of several tenants. Until very recently there was no standard benchmark designed to conduct the evaluation of database services with multitenancy in cloud computing systems. MuTeBench (MuTeBench, 2014) allowed us to make different measurements on these systems by creating different workload scenarios. Like OLTP-Bench, the execution of a benchmark in MuTeBench consists of three phases, creation of the database, data loading and execution; the type of execution must be indicated with the appropriate parameter. OLTP-Bench is an open-source framework for benchmarking, useful for relational databases, which supports data generation and execution of workloads. To carry it out, OLTP-Bench uses 14 specific benchmarks applicable to online transaction processing (OLTP) (Difallah et al., 2013), as Table 1 shows. *Transactional benchmarks* include intensive writing transactions and complex relationships. *Benchmarks oriented to the Web* address characteristics of social networks, with operations based on many-to-many graphs relationships. These benchmarks take into account public data available to simulate a real application. *Benchmarks oriented to functional tests* are focused on testing individual functionalities of certain DBMS, such as multitenancy ones.

The objective of this study is to verify through experiments the *availability*. Therefore, we use benchmarks to perform this analysis with different characteristics, capable of validating a well differentiated range of applications belonging to the three types of benchmarks accepted in OLTP-Bench,

1. TPCC, a transactional benchmark well known in the specific literature (TPCC, 2008),
2. Wikipedia, understood as a web-oriented benchmark
3. YSCB (Yahoo Cloud Serving Benchmark) as a functional test.

Next, we design a set of experiments to perform different types of tests, first by analyzing the behavior of tenants in an isolated environment, which assumes the absence of interference with other tenants. In a second phase, response times are analyzed by assuming that several tenants could start interfering if the number of these is progressively increased.

5 MEASUREMENTS AND RESULTS

The main objective of the tests carried out was to verify the performance of a multitenant DBMS under synthetic workloads that simulate operations of different applications. In order to do that, the benchmarks TPCC, YCSB (Cooper et al., 2010) and Wikipedia, were used throughout the different experiments carried out in the study.

To evaluate a multitenant database system we used the aforementioned *shared instance* model. Consequently, we create the databases shown in Table 2. Each experiment explores a different feature of a multitenant DBMS in the Cloud, such as quality of service or elasticity, for this reason we need to vary the transaction rate.

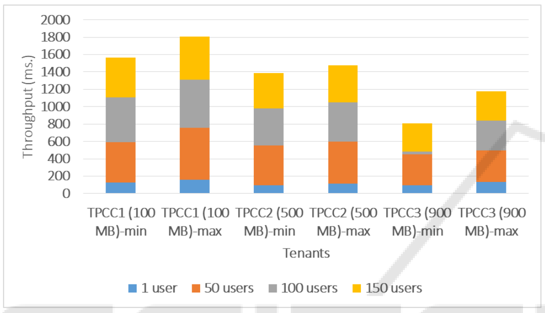
Firstly, we executed 3 tenants of different sizes and individually, without competition of resources among them. For each one, we made four measures modifying the number of users (in each measure we doubled the number of the previous one), as Table 3 shows.

5.1 Individual Execution

In this first experiment we want to evaluate how some characteristics influence the performance of the tenants that run in an instance of the DBMS: size of

Table 3: QoS temporal properties (ms.) of each tenant during a 30-minute execution of the model.

Throughput				
Tenant#	**1 user**	**50 users**	**100 users**	**150 users**
TPCC$_1$ (100 MB)	128.5 – 157.05	460.26 – 598.05	515.51 – 553.35	458.32 – 495.40
TPCC$_2$ (500 MB)	95.5 – 113.56	456.41 – 484.56	426.8 – 452.61	405.78 – 425.37
TPCC$_3$ (900 MB)	96.06 – 130.91	352.25 – 365.80	333.9 – 343.3	327.05 – 338.47
Latency				
TPCC$_1$ (100 MB)	6.36 – 10.30	83.60 – 108.72	180.61 – 193.94	279.01 – 328.2
TPCC$_2$ (500 MB)	8.67 – 10.45	97.09 – 109.62	221.0 – 305.9	344.82 – 367.58
TPCC$_3$ (900 MB)	7.63 – 10.68	134.05 – 142.0	287.71 – 299.2	440.25 – 511.08
99th percentile latency				
TPCC$_1$ (100 MB)	27.20 – 39.96	298.55 – 369.53	558.15 – 636.05	825.65 – 1020.99
TPCC$_2$ (500 MB)	30.72 – 36.30	352.30 – 381.09	570.14 – 1073.75	822.53 – 887.69
TPCC$_3$ (900 MB)	34.30 – 36.36	446.38 – 513.97	776.76 – 916.06	1086.47 – 1196.64



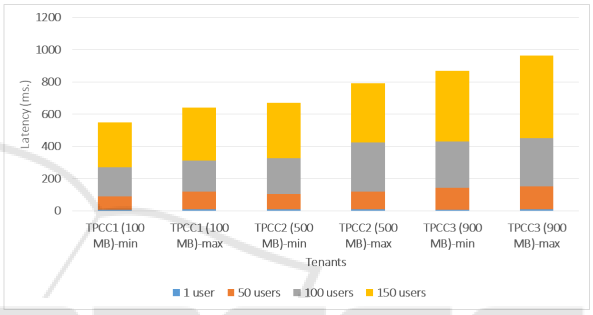


Figure 4: Minimum and maximum: (a)throughput measured for 1, 50, 100 and 150 users, (b)average latency measured for 1, 50, 100 and 150 users.

the tenant, number of users executing parallel connections to the DB, number of transactions per second that each user executes. We show in Table 3 the measurements obtained for the Throughput, Average Latency and Average Latency of the 99th Percentile response time or *tail lantency*.

From the data obtained, both the size of the database (different for each tenant) and the number of users influence the performance of the DBMS, when these values increase, it worsens throughput and increases latency.

This experiment served to observe the behavior of the DBMS in a simulated environment of a single tenant and serves as a comparison reference for multitenancy DBMS assessment.

Table 4: Configuration of tenants in the second experiment.

Tenant#	**Users**	**Start** (min.)	**Finish** (min.)	**Rate** (tps)
$TPCC_1$ $(Tenant\,1)$	25	2	7	1000
$TPCC_2$ $(Tenant\,2)$	25	4	9	1000
$TPCC_3$ $(Tenant\,3)$	25	6	11	1000
$TPCC_4$ $(Tenant\,4)$	25	8	13	1000
$TPCC_5$ $(Tenant\,5)$	25	10	15	1000

5.2 Constant Load Concurrent Execution

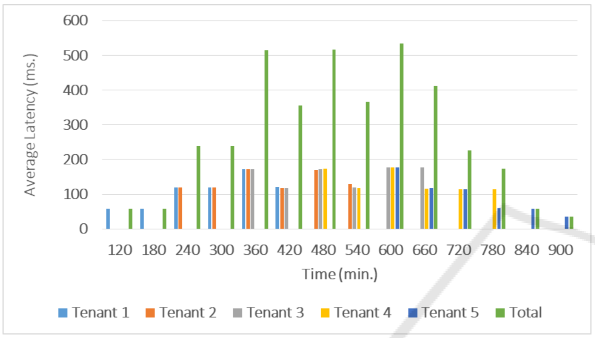
In the second experiment (see tables 5 and 6), we used TPCC tenants of 500 MB each, at a rate of 1000 transactions per second and running for 5 min. The first tenant ran after an initial phase of 2 min to reach 7 minutes, and then every 2 minutes a new tenant was included in the DBMS until including 4 more. Throughput of tenants gets worse (see Table 5) as new tenants are incorporated into the DBMS and increases when a tenant ends, and thus the total number of tenants sharing the DBMS instance decreases. The results of the total throughput show that MySQL adequately manages tenant concurrency by providing a good isolation among them. As shown in table 6, the latency values increase with the inclusion of new tenants in the DBMS and decrease when a tenant ends, therefore no anomalies in measured latency, which might lead to interference anomaly, were observed in the experiments that were carried out. The results of the average latency of 99th percentile for the 5 tenants are depicted in Figure 5. These results reflect that MySQL suitably manages the overload that suffers trying to cope with concurrency of tenants through

Table 5: Throughput of each tenant during the time of the second experiment.

	Time (m.)													
Tenant#	120	180	240	300	360	420	480	540	600	660	720	780	840	900
Tenant 1	58.265	57.661	118.604	119.005	171.604	121.511	0	0	0	0	0	0	0	0
Tenant 2	0	0	119.216	118.626	171.617	117.315	170.663	129.952	0	0	0	0	0	0
Tenant 3	0	0	0	0	172.51	117.472	171.976	118.744	177.909	177.536	0	0	0	0
Tenant 4	0	0	0	0	0	0	173.469	118.2	177.725	116.629	113.46	113.956	0	0
Tenant 5	0	0	0	0	0	0	0	0	177.907	117.526	113.471	59.116	58.545	34.908

Table 6: 99th percentile latency during the time of the second experiment.

	Time (m.)													
Tenant#	120	180	240	300	360	420	480	540	600	660	720	780	840	900
Tenant 1	243.749	251.211	388.414	395.643	508.144	280.246	0	0	0	0	0	0	0	0
Tenant 2	0	0	393.486	399.541	513.79	390.997	511.26	322.939	0	0	0	0	0	0
Tenant 3	0	0	0	0	506.217	382.018	499.573	397.456	517.671	305.902	0	0	0	0
Tenant 4	0	0	0	0	0	0	510.845	393.934	503.831	381.745	390.311	251.671	0	0
Tenant 5	0	0	0	0	0	0	510.845	0	505.716	377.593	382.607	247.873	246.484	95.263



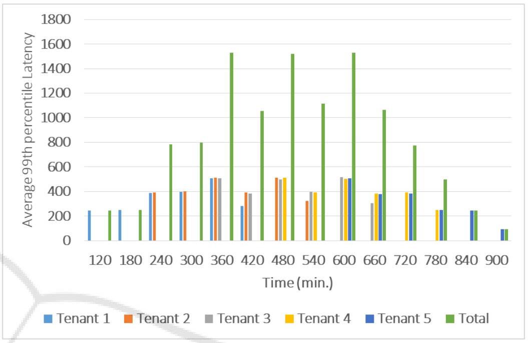


Figure 5: (a) System's average latency ; (b)99th percentile average latency (TPCC$_1$, TPCC$_2$, TPCC$_3$, TPCC$_3$, TPCC$_4$, TPCC$_5$).

preserving a good level of isolation between them.

5.3 Measurement of the QoS

According to the graphs plots shown in Figure 5, the value of average latency of p99 increases with the inclusion of tenants into the DBMS and decreases when a tenant ends its work and exits. At time equal to 640 s. the average latency reaches its maximum value, which is when all the maxima are reached in the system, i.e., the critical instant when tenants' concurrency is the highest in the DBMS for this sample, as well connections to MySQL and transactions per second. Latency evolution reflects how the increase in transactions produces an overload in the DBMS, and also how MySQL presents a good level of isolation among the different tenants. In our third experiment we used TPCC tenants of 500 MB in size, a variable transaction rate per second according to the sequence: 500, 1000, 1500, 2000, 2500, which increased every 2 min. We started by adding 2 tenants to the DBMS, with an interval of 3 min; and thus 2 new tenants were added until reaching 19 min of time when stopped incorporating new tenants. With this experiment we wanted to evaluate how the variation of workload influences the quality of service of the multitenant DBMS. We designed several experiments that represented different scenarios, combining different types of tenants and varying the workload over time. It is also observed that for time win-

Table 7: Throughput variability depending on the workload of tenants.

	(s.)		
Tenant#	**300**	**360**	**420**
TPCC 1	115.067	107.233	108.783
TPCC 2	116.667	107.617	106.867
TPCC 3	111.7	107.25	108.45
TPCC 4	112.083	109.3	106.483
TPS	3000	4000	5000

dows where the number of tenants is constant and are concurrently executed, the throughput only suffers a small decrease due to the increase in the number of transactions executed by each tenant. The data in table 7 shows that at 300 seconds 3000 tps is executed, at 360 seconds 4000 tps and at 420 seconds 5000 tps, a small decrease in throughput occurs when the number of transactions per second increases and there are no new tenants incoming.

In Figure 6 we can observe how the througput of TPCC tenants gets worse when the tenants YCSB2 and YCSB3 increase their workload. Therefore, the increase in the workload of YCSB tenants interferes

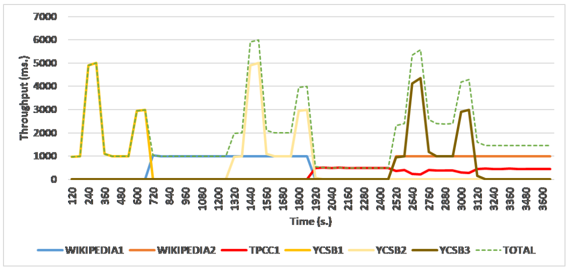
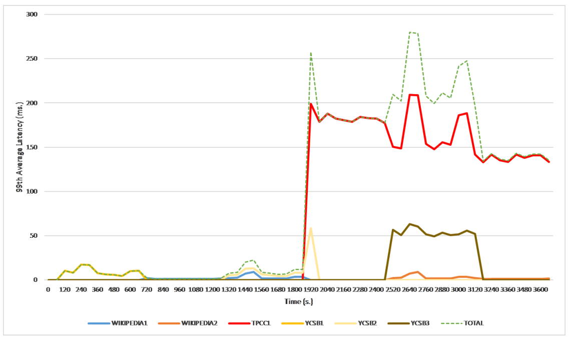



Figure 6: (a)Throughput (tps) (TPCC$_1$, Wikipedia$_1$, Wikipedia$_2$, YCSB$_1$, YCSB$_2$, YCSB$_3$) (b)99th percentile average latency (ms.) (TPCC$_1$, Wikipedia$_1$, Wikipedia$_2$, YCSB$_1$, YCSB$_2$, YCSB$_3$).

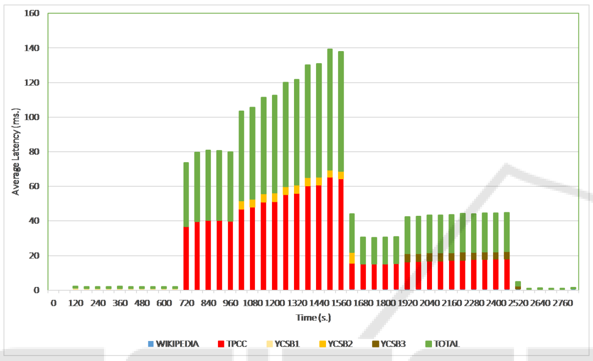
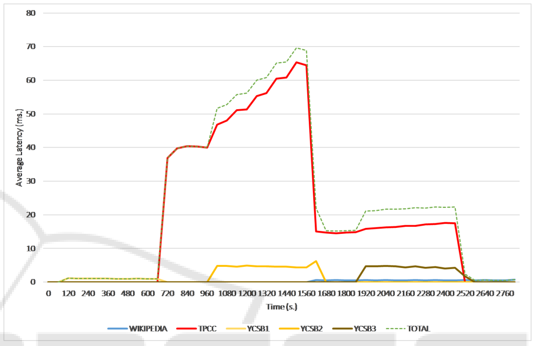



Figure 7: (a)Aggregated average latency (ms.) (TPCC$_1$, Wikipedia$_1$, YCSB$_1$, YCSB$_2$, YCSB$_3$) (b)99th percentile average latency (ms.) (TPCC$_1$, Wikipedia$_1$, YCSB$_1$, YCSB$_2$, YCSB$_3$).

with the performance of the TPCC tenant. However, it is noticed that the Wikipedia-tenant does not present interference with the rest of the tenants.

As we can observe in Figure 7, in the two latency plots, we can see that augmenting the workload of the YCSB tenant produces an increase in the TPCC tenant's latency (minutes from 17 to 26 and from 32 to 41) due to the interference that occurs between the two. It must be also pointed out that MySQL works well even with an increase in the workload, as it can keep acceptable latency values when the concurrent execution of the three tenants takes place.

5.4 Elasticity

In this section we will evaluate the elasticity when the workloads of tenants increase and decrease during a certain period of time. We designed several scenarios by using different types of tenants and combinations of heterogeneous workloads. Elasticity was analized, so that we used the same tenants of prior experiments: functional test (YCSB), transactional test (TPCC) and web-oriented test (Wikipedia). The evolution in the workload of the system varied by increasing and decreasing the workload of the YCSB tenant while the workloads of TPCC and Wikipedia tenants kept fixed. Thus, the workload of the YCSB tenant changed every 5 minutes, according to the sequence: 1000, 5000, 1000, 1000, 3000. For the benchmarks that we named: TPCC and Wikipedia, the transaction rate was set to 1000 tps. All the tenants were configured with 20 users.

The plots in Figure 7 show how MySQL works well with the elasticity of the workload, given that the troughput values respond to the evolution of the workload of the tenants, presenting little interference. At minute 44 and 55 we see how the growth of the workload of YCSB3 causes a drop in the throughput value of TPCC1, which denotes an interference between both tenants that causes a reduction in the performance of TPCC1. In Figure 6 we observe that increasing the work load of YCSB increases its latency, and when it decreases it shows a better result. After 32 minutes, the TPCC tenant starts executing, coinciding with the completion of YCSB2, which explains the alteration of the latency at that point due to the interference between the two. After 42 minutes, the YCSB3 tenant begins to run, and the interference that occurs with TPCC at minutes 44 and 51 is clearly visible.

6 CONCLUSIONS

The paper presents experimental analyses of DBMSs running on virtual machines with different benchmarks (mainly TPCC). The experiments mix different workload / tenant configurations and measure their latency and throughput. We have verified that a multi-tenant DBMS model of *shared instance* prevents from the anomaly of throughput degradation, which usually occurs due to interference between tenants of a DBMS (MySQL) located in the same VM.

Getting the tenant's time profile (workload evolution) allows detecting of changes in DBMS performance and overload and, therefore, allows to identify the level of interference between tenants. Five experiments were designed and numerous measurements have been performed using benchmarks, such as TPCC, in a cloud computing based system. The work done presents the results of the experiments in which different workloads and tenant configurations for which their latency and performance were measured. Patterns of mutual interference between tenants have been identified depending on the three types: YCSB, TPCC and Wikipedia considered in this study.

Our objective is to obtain individualized techniques for assigning tenants to VMs, relying on the monitoring of quality features in addition to the ones studied in this article, which will allow us in the future to obtain a variation of the tenant's workload through proactive models and machine learning. Likewise, it is important to detect tenant usage patterns that help to classify tenants with little interference. Several levels of QoS could be defined for different types of tenants.

REFERENCES

Agrawal, D., Das, S., and El-Abbadi, A. (2011a). Big data and cloud computing: Current state and future opportunities. In *In: Proceedings of the 14th International Conference on Extending Database Technology (EDBT'11)*. ACM.

Agrawal, D., Das, S., and El-Abbadi, A. (2011b). Database scalability, elasticity, and autonomy in the cloud. In *In: Database Systems for Advanced Applications - 16th International Conference (DASFAA 2011), v.1, 2:15*. ACM.

Barker, S., Yun, C., Hyun, M., Hacigumus, H., and Shenoy, P. (2012). Cut me some slack:latency-aware live migration for databases. In *In Proceedings of the 15th International Conference on Extending Database Technology (EDBT12), 432:443*. ACM.

Benjamin, S., Andreas, B., and Bernhard, M. (2011). Native support of multi-tenancy in rdbms for software as a service. In *In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT '11, 117:128*. ACM.

Chi, Y., Moon, H., Hacimugus, H., and Tatemura, J. (2011). Sla-tree: A framework for efficiently supporting sla-based decisions in cloud computing. In *In Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11), 129:140*. ACM.

Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *In: Proceedings SoCC. DOI: 10.1145/1807128.1807152*.

Difallah, D., Pavlo, A., Curino, C., and Cudre-Mauroux, P. (2013). Oltp-bench: An extensible testbed for benchmarking relational databases. In *In: PVLDB 7 Proceedings, 277:288*. PVLDB.

Fontan, J., Vazquez, T., Gonzalez, L., Montero, R., and Llorente, I. (2008). Open nebula: the open source virtual machine manager for cluster computing. In *In: Proceedings of Open Source Grid and Cluster Software Conference*. OSGCSC 2008, San Francisco (USA).

Moreira, L., Sousa, F., Maia, J., Farias, V., Santos, G., and Machado, J. (2013). A live migration approach for multi-tenant rdbms in the cloud. In *In: 28th Brazilian Symposium on Databases (SBBD '13), 73:78*. SBBD.

MuTeBench (2014). *Colaborative development platform in GitHub*. GitHub.

Schnjakin, M., Alnemr, R., and Meinel, C. (2010). Contract-based cloud architecture. In *In: Proceedings of the Second International Workshop on Cloud Data Management, CloudDB '10, 33:40*. ACM. DOI: 10.1145/1871929.1871936.

Schoroeder, B., Harchol-Balter, M., Iyengar, A., and Nahum, E. (2010). Achieving class-based qos for transactional workloads. In *In: Proceedings of the 22nd International Conference on Data Engineering (ICDE '06), 153:155*. IEEE-CS. DOI: 10.1109/ICDE.2006.11.

Sousa, F., Moreira, L., and Machado, J. (2011). Sladb: Acordo de nível de serviço para banco de dados em nuvem. In *In: 26th Brazilian Symposium on Databases (SBBD '11), 155:162*. SBBD.