PEESOS-Cloud: a workload-aware architecture for performance evaluation in service-oriented systems

Carlos H. G. Ferreira, Luiz H. Nunes,
Lourenço A. Pereira Jr, Luis H. V. Nakamura, Julio C. Estrella
University of São Paulo
E-mail: chgferreira@usp.br,
{lhnunes, ljr, nakamura, jcezar}@icmc.usp.br

Stephan Reiff-Marganiec
University of Leicester
E-mail: srm13@le.ac.uk

***Abstract*—It is a challenging task to ensure quality in service-oriented systems deployed in cloud computing owing to the dynamicity of its environment. Many approaches have been adopted to identify and evaluate bottlenecks and problems in performance. The most common scenario consists of distributed systems that use a workload capable of enabling clients to exploit the target system in different operational conditions. However, one requirement that tends to be overlooked is to determine how the workload is executed, as software and hardware faults can lead to its mischaracterization. In this paper, a number of problems in the workload generation have been identified and summarized. A new architecture, called PEESOS-Cloud, is proposed which allows these services to be evaluated as well as to improve the ability of the workload so that it conforms with its described characteristics. Experiments in a cloud environment were conducted to show how PEESOS-Cloud works and validate its capabilities. Our experiment also showed that the mischaracterization of the workload leads to poor results, whereas an workload-aware implementation leads to a better performance evaluation.**

***Index Terms*—Web Service, Service-oriented Systems, Performance Evaluation, Workload Characterisation**

I. Introduction

Cloud Computing is a model based on the idea of sharing computational resource (eg, networks, servers, storage and applications) on the Internet as a service [1]. When applications are available in the cloud, these services can be called *Web Services*, since they enable the features of a given application to be re-used by external entities. The services are provided on demand in a transparent way and the clients usually follow ”pay-as-you-go” scheme [2].

The attributes of Quality of Service (QoS) are used to measure the quality of services offered and thus ensure their efficiency and effectiveness. Thus, the provider is able to ensure the functionality of these applications for the clients, which is defined in a Service Level Agreement (SLA). Moreover, the QoS attributes are used for service selection and composition, which is necessary to avoid any violation of the SLA [3], [4].

Several conditions can cause a violation of the established SLA such as unexpected traffic. Examples of this were found in: a) the portal of the American health care system called *Obamacare* [5], b) technical failures and security as recorded by *Xbox Live* [6], c) a lack of capacity planning applications in abnormal periods of traffic, such as *Black Friday* [7]. This violation was caused by the inability of these systems to support many simultaneous requests and adapt to the demands of the users [8]. These problems can be avoided or mitigated by making a comprehensive evaluation of their performance, before a deployment and final delivery.

In the literature, there are studies with various approaches that seek to ensure the correct levels of QoS in different contexts, (such as capacity planning) through load testing, stress testing, performance testing and setting benchmarks. These approaches require the application of workloads in the target system that can measure variables related to resources and performance. In carrying out these studies, the experimental environment generally consists of three key components: the orchestrator of the experiments, the clients responsible for generating the workload and the target system. These components are combined with a set of computational resources, such as servers and service providers, that can be allocated in the cloud owing to their ability to provide resources on demand.

However, these environments are subject to failures with regard to hardware, software, communication and synchronization between the parties involved. In particular, there may be unpredictable noises that are added to the workload imposed on the target system. Thus, the workload might be uncharacteristic and have unexpected results in scenarios that lead to inconsistencies in the analysis. It is a complex task to detect the occurrence of failures during an experiment and quantify the effects these cause. The reason for this is that the losses incurred go beyond the question of defining the QoS levels of the service measured, or the extent of the wasted resources [9].

In this context, this paper proposes an architecture for conducting experiments in services that are able to take into account the features of the workload used. We investigated the literature and identified which problems are linked to the workload generation used to evaluate service-oriented systems (SOS). Thus, we describe a generic and extensible architecture that makes it possible to evaluate the features of the resulting workload in a target system. Finally, our proposal is applied in a case study that demonstrates how it can be used to evaluate a cloud-based SOS. The results show the importance of the workload characteristics employed in the evaluation. The main contributions made by this paper are: 1) to give a summary and analysis of current problems in the literature about how to carry out experiments with prototypes in service-oriented systems; 2) to propose a new architecture, called **P**lanning and **E**xecution for **E**xperiments in **S**ervice **O**riented **S**ystems based on **Cloud** (PEESOS-Cloud), for conducting experiments in hosted cloud applications. In addition, we describe a model of a distributed workload, with extensible and generic modules that comprise the proposed architecture. This architecture is based on the problems identified in the literature where there are solutions to generate the workload for evaluating the service-oriented system.

The remainder of this paper is structured as follows: Section II provides a literature review of approaches that can be adopted to assess service-oriented systems. Section III defines the research problem being addressed. Section IV outlines the PEESOS-Cloud architecture that was developed to evaluate the service-oriented systems in cloud, The experiment configuration and results are shown in section V, as well the benefits obtained from PEESOS-Cloud. Finally, Section VI summarizes the conclusions and makes suggestions for future work.

II. Related works

Recently research has been conducted on a number of fronts to explore the main features of the workload. In this section we will examine several studies that provide workloads for system evaluation in local and cloud environments.

Dillenseger (2009) [10] propose a framework called CLIF to generate a distributed workload in heterogeneous computational systems. The framework consists of three main parts: 1) The workload injector which handles the requests for the target system; 2) The probes which monitor the metrics in environment; 3) other components which store and analyse the metrics derived from the probes. The work adopts a flexible approach which can be extended to new proposals as discussed by Tchana et al. (2013) [11] who set a benchmark with the aid of this architecture.

Tchana et al. (2015) draw on the work carried out by Dillenseger (2009) [10] and Tchana et al. (2013) [11] to propose a Benchmark as a Service (BaaS) model, which is capable of generating self-scalable distributed workloads. A protocol is added to manage the workload injector in the CLIF framework. The results show that the proposed policies to manage the workload injector improved the original CLIF framework.

Oberle and Szabo (2015) [12] summarize the available load tests that can be performed by cloud. Thus, an architecture was designed to specify the kind of test scenario and the distributed nodes that should be used in accordance with the Test as a Service (TaaS) model. A case study was carried out to validate the prototype of the architecture. The results showed that the prototype is functional and able to measure the metrics of the target environment.

Cunha et al. (2013) [13] create a programmable environment to evaluate the performance of IaaS.The environment allows several performance test parameters to be set up, such as the number of providers and their features. A performance evaluation for two levels of workload was conducted using an environment consisting of IaaS EC2 and Rackspace machines. The collated results were drawn on to analyse the relationship between the cost and performance of the providers within the described scenario. Chhetri et al. (2013) 14, examine the Smart cloud Bench, which is a benchmark for IaaS platforms. The virtual machines are deployed with pre-designed images which have enough packages to handle the requests. Test agents are responsible for handling the requests and stating what are the desired metrics. A case study was performed and the results made it possible to determine which virtual machines are suitable according to specifics workloads.

Minzhi et al. (2012) [15] establish a cloud distributed load test platform called WSTaaS, which has the ability to deploy, set up and scale the components used to conduct the experiments. The experiments compare the approach with others and used a single node to represent several clients. The results gave rise to overload problems when only one node was used, owing to the bandwidth and number of threads required to meet the requests and this had an adverse effect on the final analysis. Budai et al. (2014) [16] employ a distributed framework for workload generation to evaluate the services hosted in the cloud. The architecture of this framework comprises several agents and a controller module. The agents are virtual machines that are responsible for handling the requests in the target environment. The workload is operated through messages that are exchanged between the experiment controller and the agents. A case study was conducted to evaluate the MySQL cluster and the performance metrics and to expose scalability problems.

Nunes et al. (2014) [17] enploy the PEESOS tool, a mechanism designed to study the performance and QoS attributes of a service-oriented architecture called WSARCH. A case study shows the performance of the WSARCH architecture in different scenarios. However, this mechanism is restricted to the WSARCH architecture and has problems in generating the workload.

TABLE I
Summary of Related Wor

	Objective	**Experiments Environment**	**Workload-aware**
Dillenseger (2009) [10]	Load Test	Local	✗
Tchana et al. (2015) [9]	Stress Test	Local	✗
Oberle and Szabo (2015) [12]	TaaS	Cloud	✗
Cunha et al. (2013) [13]	Performance Test	Cloud	✗
Chhetri et al. (2013) [14]	Bechmark	Cloud	✗
Nunes et al. (2014) [17]	Load Test	Local	✗
Minzhi et al. (2012) [15]	Load Test	Cloud	✗
Budai et al. (2014) [16]	Load Test	Cloud	✗
PEESOS-Cloud	Flexible	Local Cloud	✓

Table I summarizes the main features of the related works. It ishould be noted that all the related works set out a range of choices to generate and apply their systems in different contexts. Dillenseger (2009) [10], Tchana et al. (2015) [9] and Nunes et al. (2014) [17] rely on the local network to generate workloads and do not take account of the level of noise in this environment, which can influence the expected workload. Other works such as those of Cunha et al. (2013) [13], Oberle and Szabo (2015) [12], Chhetri et al. (2013) [14], Minzhi et al. (2012) [15], Budai et al. (2014) [16] make use of the cloud environment to generate the workload. However they are not able to ensure the quality of the generated work or in some cases, fail to make use of distributed hosts.

In summary, the related works seek to introduce an assessment system that automates the evaluation process by allocating previous or on demand resources that can deploy, manage and gather performance metrics and applications. On the other hand, these works fail to take account of faults that can occur during the workload generation and are not able to provide the desired workload. Thus, PEESOS-Cloud architecture has a workload generation that uses a distributed environment where the requests are performed and sychronized with the aid of a Uniform Resource Identifier (URI). This workload can be generated through a local environment or in the cloud and is capable of analyse the resulting workload that is required due to the dynamics and constraints of the cloud environment.

III. The Problem Statement

In this section, there is a brief description of the issues involved in our research. The workload test denotes that any workload used in performance studies can be classified as either real or synthetic [18], [19]. Since there are two types of load, there is the possibility that the execution and application may be either centralized or distributed.

The real or known workload is derived from *traces* or any basic information that can identify the incoming requests within a given target system. One advantage is that this workload is a closer representation of reality. On the other hand, it may be difficult to obtain and handle *trace* data due to their complexity and the amount of information involved. The difficulty of represent different scenarios is another constraint for a current workload, since it restricts the evaluation of the system to a single perspective [3]. Alternatively, synthetic workloads can be constructed by means of a pseudo-random number generator with a probability distribution and descriptive parameters to create a desired behavior or extracted by modeling a real load [3], [20]. The main advantage of this type of load is that it has the ability to test the system under different conditions and variations.

Whatever type of workload, is employed, every attempt is made to ensure its quality so that the generated workload can be as similar as possible to what is specified. When the workload is executed centrally, the mischaracterisation caused by the presence of noise occurs as a result of problems such as the overloading of the components responsible for generating the workload, especially when a single machine has to represent many clients [9], [14]. [21] shows how the excessive number of threads created in the instances that carry out load generation have an adverse effect on the generation of the workload. In addition to these problems that can occur on the machines that host the environment, network noise is also a serious issue, (especially when cloud is used) because the resources tend to be even more distant geographically. Moreover, when the cloud is used as a resource, factors such as the performance of the service provider and the communication and network conditions can influence the generation workload. This is because the quality of the generated workload depends on how the system will distribute and coordinate the clients who carry out the requests within the target system. If a fault occurs in the contracted resources at runtime, it is difficult to make a decision in time and thus be able to avoid impairing the workload generation [22].

This work makes direct use of the workload generation whether it is a synthetic or real distribution. Thus, it is a challenging task to quantify the noise since it depends on the approach that is adopted. Since the workload is individually configured for each client, it is difficult to know a) when a request should be handled, b) when this was done, and c) which client carried it out. To obtain this information from an experiment, it would be useful to assess and quantify the noise that affects the generation of the workload. Furthermore, metrics can be employed to quantify the difference between the characteristics of the workload which reaches the target system rather than the workload that was specified. The metrics can include the position, dispersion, central tendency, focus and skew, and correlation measured, as well as, the characterization by means of statistical distributions that best fits the data [18], [19].

Thus, maximizing the generated workload so that it is nearer the specified workload is a non trivial task. However, it makes it possible to check the final characteristics of the workload that is generated and imposed on the target system to have an analysis of the real workload before performance metrics.

IV. PEESOS-Cloud architecture

A. Overview

In this paper we propose an architecture to evaluate oriented-service system using distributed workloads. Exploring how cloud environments can be used to generate a workload, requires an analysis of how the environment is used and how the target system handles it. Thus, it is extremely important to establish what impact the workload has on the target system and anticipate possible problems that may occur during the experiment.

The proposed architecture is formed of three modules:

- **PEESOS-Cloud Manager:** this orchestrates and manages the components of the PEESOS-Cloud architecture. The experimental information (such as workload, service, clients, target system and database for match-fixing) is defined in this module. In addition, it is responsible for communication with the other modules.
- **Target System:**: this hosts the target application and receives the client requests – usually through a RESTful interface. This means that the target system does not rely on technology and uses an URI to call the service. The metrics regarding usage of the systems resources such as CPU, memory and disk are measured.

- **Clients:** performs the imposed workload for the target system. It simulates real users and collects pre-defined metrics. The clients are allocated in hosts that are able to communicate with the PEESOS-Cloud server and the target system. The workload is generated by requests from a client to a URI service. Some metrics are collected for these clients such as execution time, response time, rate of successful requests and the status of the requests.

Figure 1 represents an abstraction of the PEESOS-Cloud architecture. It should be stressed that each module has a monitor with different responsibilities. It provides a communication channel for the client with PEESOS-Cloud Manager and is capable of performing requests for the Target System module. In the target system, the monitor is able to render the desired service and provide experiment information to the PEESOS-Cloud Manager. Finally, in the PEESOS-Cloud Manager, the monitor is responsible for exchanging messages that orchestrate the experiment with the Clients and Target System.

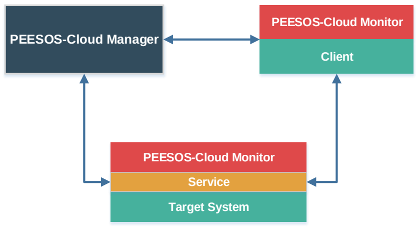


Fig. 1. Modules of PEESOS-Cloud

B. Workload Generation Model

A generic model was purposed to be employed for distributed workload generation and check its main characteristics regardless of the noise present in the environment. To achieve this, the proposed model is based on the decomposition of the desired workload and will be applied in the target system as a time series. Regardless of the used workload (synthetic or real), it is possible to express its arrival rate in an application server via a time series containing the number of requests to be performed per second. Thus, a time series must express the number of requests as a workload (**w**) at a predefined instant of time (**i**), represented by (**t**), which can be expressed with a demand d(**t**) and a possible noise n(**t**) represented by $w(t_i) = d(t_i) + n(t_i)$.

TABLE II
TIME SERIES WORKLOAD

Time	t_1	t_2	$t_{\dots}$	t_m
Number of Requests	$\mathbf{r_{t1}}$	$\mathbf{r_{t2}}$	$\mathbf{r_{t\dots}}$	$\mathbf{r_{tm}}$

On the basis of the number of desired requests at a specific time and the current noise level, it is possible to adopt a new approach to generate and evaluate workloads. The use of a time series means that the number of requests can be established in an instant of time, and in this way, the workload imposed on the target system can be characterized. On the other hand, it is essential to decompose and synchronize the requests properly to avoid decharacterizing the desired workload. Table II provides an example of a time series, in which a set of requests is defined for each point in time.

As the time series is defined, the decomposition process is initiated in accordance with the number of requests to be handled at each instant of time and the number of available clients. Table III represents the decomposition workload that is set out in Table II. This expresses the number of requests and the number of clients required to represent this workload in a predefined instant of time, as well as which client will make those requests at each instant of time. For this reason, a request matrix is established with dimension m x n, where m represents an instant of time, n corresponds to the total number of available clients and the intersection between m x n indicates if the client n will be perform the request in the instant of time m.

TABLE III
WORKLOAD DECOMPOSITION

Clientes/ Requisições/t	$\mathbf{c_1}$	$\mathbf{c_2}$	$\mathbf{c_{(\dots)}}$	$\mathbf{c_{(n-1)}}$	$\mathbf{c_n}$
$\mathbf{r_{t_1}}$	$r_{t1,1}$	$r_{t1,2}$	$r_{t1,(\dots)}$	$r_{t1,(n-1)}$	$r_{t1,n}$
$\mathbf{r_{t_2}}$	$r_{t2,1}$	$r_{t2,2}$	$r_{t2,(\dots)}$	$r_{t2,(n-1)}$	$r_{t2,n}$
$\mathbf{r_{t(\dots)}}$	$r_{t(\dots),1}$	$r_{t(\dots),2}$	$r_{t(\dots),(\dots)}$	$r_{t(\dots),(n-1)}$	$r_{t(\dots)n}$
$\mathbf{r_{tm}}$	$r_{tm,1}$	$r_{tm,2}$	$r_{tm,(\dots)}$	$r_{tm,(n-1)}$	$r_{tm,n}$

C. Integration of the Modules

The proposed architecture acts in a distributed and independent way, which means that it is necessary to address questions regarding communication and synchronization. A Network Time Protocol (NTP) server is set for the whole environment. Figure 2 shows the flow of the operating system architecture which is used to evaluate the system. In this scenario, the question of the availability of potential clients was taken into account. Thus, to evaluate a system the following stages must be followed:

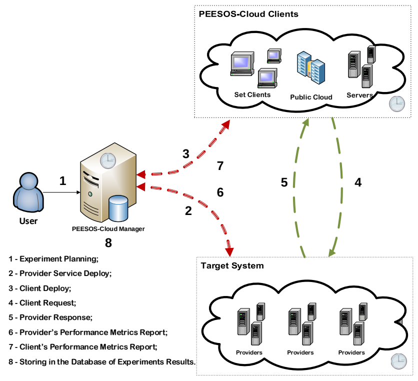


Fig. 2. PEESOS-Cloud Workflow

1) The user configures an experiment, using scripts to define the parameters for the workload to be applied, the service and evaluation of the target system ;
2) The application PEESOS-Cloud Monitor of the target system receives the information of the service to be used.

After, the service is deployed in provider on the basis of the information provided by the module PEESOS-Cloud Server;

3) The potential clients receive information about the experiment and the workload;
4) Depending on the features of the workload, the clients makes requests within a target system;
5) The request response and metrics of interest of the clients as execution time, are sent to the clients. In the same instant the providers are being monitored with regard to the rate of resource utilization;
6) In this stage, when the experiment has already been completed, the providers of the PEESOS-Cloud send the results, of the monitoring, to the PEESOS-Cloud Server;
7) Like the providers, the PEESOS-Cloud Monitor sends the results of the requests to the PEESOS-Cloud Server when the experiment has already been completed;
8) All the information is stored in a database, and the completed experiment is ready for analysis.

The NTP synchronism becomes essential for the experiment execute as expected. All this process is based in message exchange using Sockets and JavaScript Object Notation (JSON). The feedback from the operations in all the activities is evaluated to control for possible mistakes.

V. Experimental Evaluation

In this section, PEESOS-Cloud architecture was used to perfom a performance evaluation of a CPU-Bound application offered as a RESTful service . The driving-force behind this study is the desire to make use of PEESOS-Cloud to evaluate service-oriented systems deployed in the cloud, and allocate resources from this infrastructure. This is followed by adopted methodology for the planning, execution and analysis of the experiments [18], in which the model for the experiments was based on a full factorial design. We performed 10 replications of each experiment, to achieve a confidence level of 95%. The workload was analysed in all the experiments. The characteristics of the specified workload were taken into account for this. Thus, when conducting the experiments for all the combinations and replications, the resulting features of workload were analysed and compared with the used workload. In the final stage, we analysed the effects of workload on the evaluation.

The workload was analysed in all the experiments. The characteristics of the specified workload were taken into account for this. Thus, when conducting the experiments for all the combinations and replications, the resulting features of workload were analysed and compared with the used workload. In the final stage, we analysed the effects of workload on the evaluation.

A. Experiments Design

Table IV shows the factors and levels of the experiments. In this work, the average of number of requests per second was used to check the application performance at different levels of the workload. The value of its levels were utilized as parameters of a Poisson distribution. In this way, the workload fitted this probability distribution, and assumed that every replication 300 seconds. The method of generating workload refers to how the PEESOS-Cloud clients were instantiated. In the centralized approach, all the clients were hosted in a single physical node, since the workload generation was carried out in a centralized way, while, in the distributed way, the clients were divided between 5 machines. Each client only had to carry out at most, one request per second.

TABLE IV
Experiment Design

Factor	Levels
Mean of Requests/s	15, 30, 45
Workload Generation Mode	Centralized, Distributed

All the architecture components were hosted in a private cloud, located at the University of Sao Paulo[1]. The testbed was set (as shown in Figure 3).

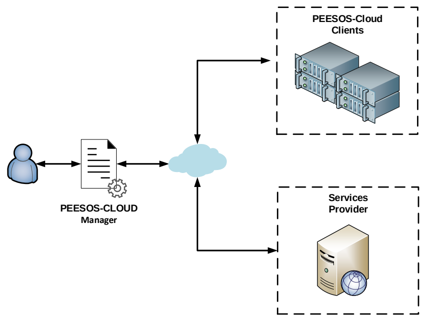


Fig. 3. Experiments Environment

The analysed response variables are:

- **Workload generated by clients:** check if the specified workload was correctly generated by the PEESOS-Cloud architecture;
- **Workload processed by the provider:** check the features of the workload imposed on the target system.
- **Service execution time by the provider:** check the service execution time for the defined workloads;

Table V describes the computational environment used for the experiments. The application in which the feature is a CPU-bound operation, calculates the factorial of the number 1000. This value is empirically defined through preliminary tests, and takes particular account of the computational resources used.

TABLE V
Specification of computational resources used.

	Processor	Memory	Operating System
Virtual Machine	1 vCPU	1 GB	Ubuntu Server 14.04.1 LTS x64
Physical Machine	Intel® Core^{TM}2 Quad Q9400 2.66 GHz	8 GB	Ubuntu Server 14.04.1 LTS x64
Environment PEESOS-Cloud	Intel® Core^{TM} I7 4790 3.60GHZ	32 GB	Ubuntu Server 14.04.1 LTS x64

[1]http://infra.lasdpc.icmc.usp.br/

B. Results

The graph in Figure 4 contains the workload generated by PEESOS-Cloud through the evaluated application. X axis shows the workload generator module grouped by the workloads with the means of requests. The Y axis shows the means of requests per second for each workload during the experiment.

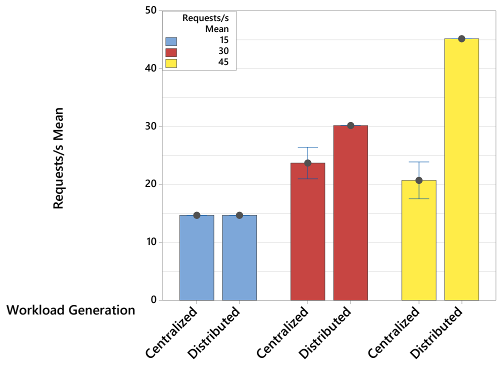


Fig. 4. PEESOS-Cloud Workload Generation

It can be observed that when the workload generation mode is distributed, 100% of requests are generated correctly in every experiment, when the total amount of time for the workload is 300 seconds. In contrast, when the workload generation mode is centralized, for loads with averages of of 30 and 45 seconds, the characteristic for the mean average of requests per second is violated. In this case, when the workload averaged 30 requests per second (for the centralized mode), a load lower than 23.70 requests per second was generated (a negative difference of 21.71%). In the same way, when the workload averaged 45 requests per second in centralized mode, the load is also lower, averaging 20.70 requests per second was generated by PEESOS-Cloud clients (a reduction of 54.17%).

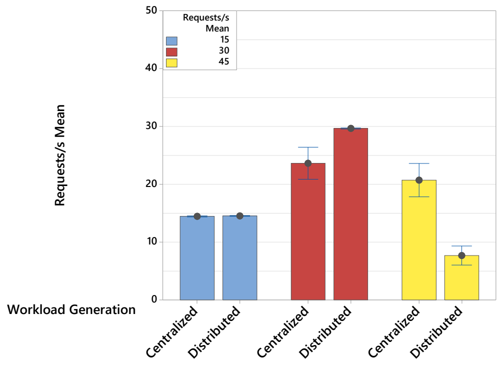


Fig. 5. Processed workload in providers

In the graph in Figure 5, the workload processed by the service provider can be seen. With regard to the load that was generated by PEESOS-Cloud clients, in most of the experiments the provider met the demand, unless the used load had an average of 45 requests per second, and was generated by the distributed mode. In this case, it served an average of 7.68 requests per second. When analysing the workload with an average of 15 requests per second, the average of the processed workload was 14.44 and 14.42 requests per second, for both the generation modes (distributed and centralized), and was statistically the same. In the case of the workload with an average of 30 requests per second, the mean average of the processed workload was 23.62 and 29.62 requests per second, the higher figure being for the load generated with the distributed mode. The same applied to the load with an average of 45 requests per second, where the average was 29.69 requests and 7.68 for the processed loads with both distributed and centralized methods.

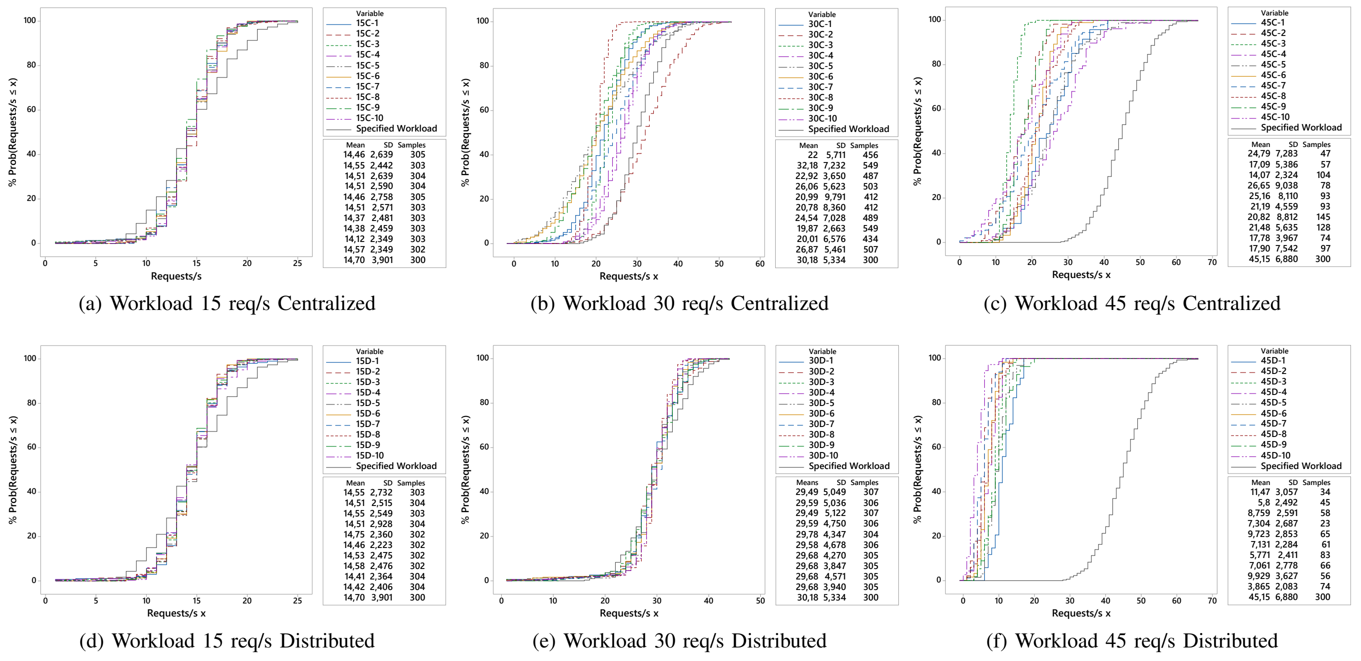
As well as the use of the averages of requests per second, Figure 6 shows the Cumulative Distribution Function (CDF) of the processed workload for each experiment, together with the replications of the averages, standard deviation and number of samples. From the the cumulative distribution, it can be seen that the experiments represented in Figures 6a, 6d and 6e behave in the same way. The replication curves and the specified workload overlap in the most graphic area, and approximately 50% of the values are lower and the other 50% are higher than the average for workload requests. This suggests that the execution time of the load was satisfactory, which means that the total time of requests (represented by *Samples*) was close to 300 seconds, with a maximum variation of 2.33%, following what is specified by the load. In view of this, it should be underlined that even the average rate of processed workloads is similar to the specified workloads, with only a tiny statistical difference.

The results of the experiments in Figures 6b and 6c showed that there was a greater variation. In this case, it is clear that almost 100% of the number of the requests per second that were carried out, have a value lower than the average of requests per second of the used load. A difference in the specified load can be seen in Figure 6f as well, because of the incapacity of the provider to process all the requests. Given that the charge together with the workload that was processed by the services providers, in the graph in Figure 7 it is possible to observe the mean execution time of the factorial services in the provider that were carried out for each experiment.

For the workload with an average of 15 requests per second, the average time of service execution was 128.50 and 131.50 milliseconds, for both distributed and centralized generation modes respectively, and it was statically the same. When the workload had an average of 30 requests per second, the mean execution time of service was 312.06 and 375.35 milliseconds, which was greater in the generated load for the distributed mode. Lastly, when verifying the execution times for the load that has an average of 45 requests per second, there was found to be a huge difference between centralized and distributed modes, which were 321.20 and 771.7 milliseconds.

C. Analysis of results

The results made it clear how the generation mode can interfere with the final characteristic of the load. When evaluating the average, it is possible to detect bottlenecks when the



Mean	SD	Samples
14,46	2,639	305
14,55	2,442	303
14,51	2,639	304
14,51	2,590	304
14,46	2,758	305
14,51	2,571	303
14,37	2,481	303
14,38	2,459	303
14,12	2,349	303
14,57	2,349	302
14,70	3,901	300

(a) Workload 15 req/s Centralized

Mean	SD	Samples
22	5,711	456
32,18	7,232	549
22,92	3,650	487
26,06	5,623	503
20,99	9,791	412
20,78	8,360	412
24,54	7,028	489
19,87	2,663	549
20,01	6,576	434
26,87	5,461	507
30,18	5,334	300

(b) Workload 30 req/s Centralized

Mean	SD	Samples
24,79	7,283	47
17,09	5,386	57
14,07	2,324	104
26,65	9,038	78
25,16	8,110	93
21,19	4,559	93
20,82	8,812	145
21,48	5,635	128
17,78	3,967	74
17,90	7,542	97
45,15	6,880	300

(c) Workload 45 req/s Centralized

Mean	SD	Samples
14,55	2,732	303
14,51	2,515	304
14,55	2,549	303
14,51	2,928	304
14,75	2,360	302
14,46	2,223	302
14,53	2,475	302
14,58	2,476	302
14,41	2,364	304
14,42	2,406	304
14,70	3,901	300

(d) Workload 15 req/s Distributed

Means	SD	Samples
29,49	5,049	307
29,59	5,036	306
29,49	5,122	307
29,59	4,750	306
29,78	4,347	304
29,58	4,678	306
29,68	4,270	305
29,68	3,847	305
29,68	4,571	305
29,68	3,940	305
30,18	5,334	300

(e) Workload 30 req/s Distributed

Mean	SD	Samples
11,47	3,057	34
5,8	2,492	45
8,759	2,591	58
7,304	2,687	23
9,723	2,853	65
7,131	2,284	61
5,771	2,411	83
7,061	2,778	66
9,929	3,627	56
3,865	2,083	74
45,15	6,880	300

(f) Workload 45 req/s Distributed

Fig. 6. CDF of workload processed by provider

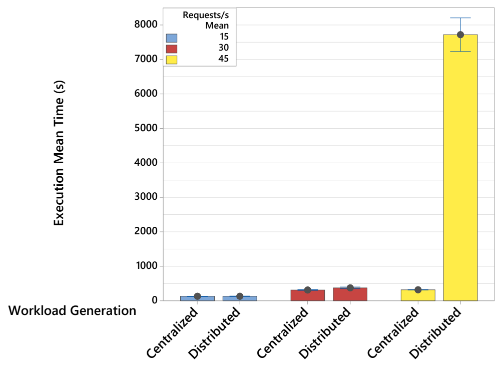


Fig. 7. Execution Time of the Factorial Service

workload was generated, as it was observed in the centralized mode and averages of 30 and 45 requests. The reason for this is that the overload in the physical node made the PEESOS-Cloud clients fail when generating the requests. In addition, it should be noted, that in the cases of experiments that underwent the mischaracterisation (Fig. 7) as a result of noises in the specified workload, the average execution times were statistically the same, which suggests that the mischaracterization of the workload occurred when the load was being generated. As well as this, there was a noticeable difference when the mischaracterization of the specified workloads was compared with the processed workloads. This reduced the average to 21.71% and 54.17% for the loads of 30 and 45, respectively that were generated by the centralized mode (Fig. 4).

The difference between the generated workloads and those that were specified is 2.33% when the average number of requests is noted; this applied to the scenarios who had no problems when generating the workload. An increase in execution time for all the load causes a skew due to standard deviation, as demonstrated. In this case, there was a tiny difference in the average processed workload due to variations in the replications. The effect of generated workloads on the service provider in the final load processed, suggests that, (except for the scenario where there is an average load of 45 requests per second generated in a distributed way), the loads were processed as expected. In the specific scenario of the average of 45 requests per second generated in a distributed way, the impact of the workload was greater than the maximum that could be supported by the services provider, and that after an initial experimental period, the provider was unable to answer the requests. As a result, only an average of 17.01% of requests were met. The high value in the execution time in this case is caused by the waiting time in the queue while the request is being processed. As every request generates a Java process that has to be scheduled for execution by the operating system, this experiment produced a higher number of concurrent requests per second that the system can handle.

The application server used to offer a service appeared to be unstable while the experiments were being carried out. The allocated resources in VM for the processed load, resulted in factors such as *Timeout*. In addition, the requirement for more resources to meet the concurrent requests resulted in a failure on the part of the nprovider to reach the processing limit, and only some of the 300 seconds of experiments where performed. This means that it is necessary to adopt scalability policies and optimization techniques.

VI. Conclusion and Future Work

This paper proposed an architecture for conducting experiments in services by taking note of the characteristics of the workloads. We found in the literature that there are problems arising from the workload generation when evaluating service-oriented systems.

The workload-aware approach proposed by PEESOS-Cloud proved to be effective, as was shown in the experimental evaluation. However, the centralized workload generator failed to comply with the specifications because it reduced the number of requests per second due to a lack of computing power in the clients. On the other hand, when distributed clients were used, the workload characteristics were retained by the client. In particular, in the experiment with 45 requests per second carried out with the distributed clients, the provider could not handle this level of requirement s and this revealed a limited a capacity. This meant that proving the requirement of (1) led to an accurate generation of requests (whether centralized or distributed) and (2) the results showed how they arrived at the provider. Moreover there were failings on the part of the clients in the generation of the requests and the provider lacked the capacity to cope with the workload. This result confirms that workload-aware systems lead to better experimental outcomes and thus, a consistent performance analysis.

On the other hand, when distributed clients were used, the workload characteristics have been retained at clients side. Specifically, in the experiment with 45 requests per second coming from the distributed clients, the provider could not handle such level of exigency exposing a capacity limitation. And so, proving the requirement of (1) ensuring an accurate generation of requests (whether centralized or distributed) and (2) and logging how they arrive at the server side — limitation of the clients in the generation of the requests and the provider lack of capacity to cope with the workload, respectively. This results confirms that workload-aware systems produce better experiment executions and then consistent performance analysis.

In future work, we intend to use PEESOS-Cloud to develop and validate new QoS levels for services performance studies. Finally, we will propose business models for TaaS (Test as a Service) and BaaS (Benchmark as a Service) that take account of aspects of workloads and the allocation of resources in the cloud.

Acknowledgment

The authors would like to thank National Council for Scientific and Technological Development (CNPQ, process 139917/2014-4) and USP.

References

[1] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[2] Z. Zheng, Y. Zhang, and M. Lyu, "Investigating qos of real-world web services," *Services Computing, IEEE Transactions on*, vol. 7, no. 1, pp. 32–39, Jan 2014.

[3] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida, *Performance by Design: Computer Capacity Planning By Example*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[4] L. Qi, W. Dou, J. Ni, X. Xia, C. Ma, and J. Liu, "A trust evaluation method for cloud service with fluctuant qos and flexible sla," in *Web Services (ICWS), 2014 IEEE International Conference on*, June 2014, pp. 345–352.

[5] D. Morgan and C. Humer, "Web traffic, glitches slow obamacare exchanges launch," October 2013, [Online; posted October 1,2013]. [Online]. Available: http://www.reuters.com/article/2013/10/01/us-usa-healthcare-idUSBRE98T14R20131001

[6] G. Jones, "Xbox one fans warned of new xbox live outages after december online issues," December 2014, [Online; posted December 7, 2014]. [Online]. Available: http://www.express.co.uk/entertainment/gaming/544198/Xbox-One-Microsoft-Xbox-Live-down-December-Christmas-offline

[7] J. Bloomberg, "The slow, the crashed, the out of stock: Blackfriday fail twitter report," November 2015, [Online; posted November 27,2015]. [Online]. Available: http://www.forbes.com/sites/jasonbloomberg/2015/11/27/the-slow-the-crashed-the-out-of-stock-blackfriday-fail-twitter-report

[8] Z. Jiang and A. Hassan, "A survey on load testing of large-scale software systems," *Software Engineering, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[9] A. Tchana, N. De Palma, B. Dillenseger, and X. Etchevers, "A self-scalable load injection service," *Software: Practice and Experience*, vol. 45, no. 5.

[10] B. Dillenseger, "Clif, a framework based on fractal for flexible, distributed load testing," *annals of telecommunications - annales des télécommunications*, vol. 64, no. 1-2, pp. 101–120, 2009. [Online]. Available: http://dx.doi.org/10.1007/s12243-008-0067-9

[11] A. Tchana, B. Dillenseger, N. De Palma, X. Etchevers, J.-M. Vincent, N. Salmi, and A. Harbaoui, "Self-scalable benchmarking as a service with automatic saturation detection," in *Middleware 2013*, ser. Lecture Notes in Computer Science, D. Eyers and K. Schwan, Eds. Springer Berlin Heidelberg, 2013, vol. 8275, pp. 389–404. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45065-5_20

[12] T. Oberle and C. Szabo, "An architectural prototype for testware as a service," in *Applied Machine Intelligence and Informatics (SAMI), 2015 IEEE 13th International Symposium on*, Jan 2015, pp. 15–19.

[13] M. Cunha, N. Mendonca, and A. Sampaio, "A declarative environment for automatic performance evaluation in iaas clouds," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, June 2013, pp. 285–292.

[14] M. Chhetri, S. Chichin, Q. B. Vo, and R. Kowalczyk, "Smart cloud-bench – automated performance benchmarking of the cloud," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, June 2013, pp. 414–421.

[15] M. Yan, H. Sun, X. Wang, and X. Liu, "Ws-taas: A testing as a service platform for web service load testing," in *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, Dec 2012, pp. 456–463.

[16] P. Budai and B. Goldschmidt, "Performance analysis of cloud-based application," in *Large-Scale Scientific Computing*. Springer, 2014, pp. 476–483.

[17] L. H. Nunes, L. H. Vasconcelos Nakamura, B. Tardiole Kuehne, E. M. de Oliveira, D. O. Libardi, M. Rafael, L. Junqueira Adami, J. C. Estrella, and S. Reiff-Marganiec, "Peesos: A web tool for planning and execution of experiments in service oriented systems," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 606–613.

[18] R. Jain, *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.*, ser. Wiley professional computing. Wiley, 1991.

[19] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[20] H. Xi, J. Zhan, Z. Jia, X. Hong, L. Wang, L. Zhang, N. Sun, and G. Lu, "Characterization of real workloads of web search engines," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, Nov 2011, pp. 15–25.

[21] S.-J. Hsieh, G.-H. Luo, S.-M. Yuan, and H.-W. Chen, "A flexible public cloud based testing service for heterogeneous testing targets," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, Sept 2014, pp. 1–3.

[22] J. Zhou, S. Li, Z. Zhang, and Z. Ye, "Position paper: Cloud-based performance testing: Issues and challenges," in *Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services*, ser. HotTopiCS '13, 2013, pp. 55–62.