A low cost workload generation approach through the cloud for capacity planning in Service-Oriented Systems

Carlos H. G. Ferreira
ICMC/USP
São Carlos, Brazil
chgferreira@usp.br

Julio C. Estrella,
Luiz H. Nunes,
Luis H. V. Nakamura,
Rafael M. Libardi
ICMC/USP
São Carlos, Brazil
{jcezar, lhnunes,
nakamura,
mira}@icmc.usp.br

Bruno G. Batista
IMC/UNIFEI
Itajubá, Brazil
brunoguazzelli@unifei.edu.br

Maycon L. Peixoto
UFBA
Salvador, Brazil
maycon.leone@ufba.br

Dionísio M. Leite
UFMS
Ponta Porã, Brazil
dionisio.mlf@gmail.com

Stephan Reiff-Marganiec
University of Leicester,
Leicester - UK
srm13@le.ac.uk

ABSTRACT

This paper presents a cloud approach for low cost capacity planning evaluations. To perform these evaluations we have to specify and measure the workload on the target system to discover issues and make the necessary adjustments. However, due to high costs, these evaluations are usually done using simulations, which does not consider stochastic effects. We propose to use a tool named PEESOS, a generic and flexible approach to apply real workloads and measure used resources on these real systems. As a proof of concept, our case study use a real ticket sales service to evaluate the influence of scalability in the resource provisioning to show how PEESOS can lower the cost of such real evaluations. The results show the efficiency and savings that we can obtain using PEESOS for large-scale capacity planning evaluations before the real services are deployed. This approach can avoid several problems that real services faces when they launch.

Keywords

Capacity Planning; Workload Generation; Service-Oriented Systems.

1. INTRODUCTION

Capacity planning can lead organisations to understand how performance issues happen and ensure that a system will meet the expected performance [3]. The process of determining the production capacity, using the current performance system as a baseline, is the focus of capacity planning [2].

Planning the capacity of a system is not a trivial task and requires that a series of steps be followed. Firstly, understand the target environment and their main functional and non-functional requirements. As the main goals are identified, a workload must be characterised to measure the system performance under a specific condition. The performance evaluation should consider a set of metrics that represents the system purpose. Finally, based on the performance evaluation, an analytical model that accurately predicts the system performance must be developed and applied to predict and optimise the system performance [3, 10].

Nowadays, several tools provide a set of methods to perform capacity planning in Service-Oriented Systems (SOS). However, these tools use local environments to perform the workload or analytical models to reproduce the system characteristics, which cannot be the same as found in a real scenario [21]. In this way, our methodology will use the **P**lanning and **E**xecution of **E**xperiments in **S**ervice **O**riented **S**ystems (PEESOS) [1] tool [12] to perform the verification and validation of the capacity planning with a low cost.

As stated by Nunes et al. [12], the main functionality of PEESOS is facilitate the process of conducting functional testing and capacity planning in service-oriented systems. This paper address the workload generation to perform an efficient capacity planning experiment, using PEESOS tool. This step aims to understand how each service request impacts on the allocated resources. Differently of the mechanisms presented in the literature, we propose a financial low cost approach using a cloud environment to generate a real workload on SOS. The use of cloud resources enables to allocate resources geographically distributed using the 'pay-as-you-go' model, which allows analysing the performance of a target system in different scenarios such as presented by Morgan and Humer (2013) [11] and Jones (2014) [8].

We intend to use the PEESOS tool to consolidate the service implantation in cloud environments, paying for the exact resources that will be used, as in the 'pay-as-you-go' business model, saving money and operational resources. In this way, the main contribution of this paper is a generic and flexible approach, which demands low financial resources to generate workloads in capacity planning experiments. Exploring the main characteristics of a cloud, different types of workload can be applied in several kinds of SOS. This results in a system behaviour closer to real scenarios and enables to establish more accurate analytical and performance models. To validate our proposal, experiments were carried out by

[1]http://peesos.wsarch.lasdpc.icmc.usp.br

PEESOS, which show how the tool can be used to assist in the capacity planning with a low cost. By means of the results, it was possible to identify the real limits of a service, and how the allocated resources impacted the system performance.

The paper is organised as follows: Section 2 presents a literature review of existing approaches to perform capacity planning in service-oriented systems. Section 3 describes the basic background related to this paper. Section 4 describes the methodology and experimental design, as well as the results analysis. Finally, the conclusions and directions for future work are presented in Section 5.

2. RELATED WORK

Several approaches were proposed for service-oriented systems testing and evaluation, and to ensure their efficiency and compliance with the Service Level Agreements (SLAs) and Quality of Service (QoS) requirements. In this section, we present some approaches for service-oriented systems available in the literature.

According to Vögele et al. [19], the challenge in evaluating Web Services is the fact that services belong from different parties. To study the workload that these services receive, it is necessary to quantify the amount of requests per hour. This allows the extraction of test case scenarios to perform a system evaluation. Using this study as a base, the authors developed a mechanism to generate automatic workload for these services and supporting the capacity planning design.

Lopes et al. [9] propose a model to guide the infrastructure capacity planning of cloud services. This model helps to calculate how many instances are needed to run a Web server workload. A set of information about the workload, application, reservation and on-demand markets are used as input in this model. The problem with this solution is the difficulty in obtaining information about the workload to extract precise parameters. In addition, the analytical model does not consider the dynamic changes presented in the Internet.

Barna et al. [4] also presents an analytical solution based on Queueing Network Model to set the test workload parameters in the system. The charge level is set automatically using the Kalman Filter that receives monitoring input parameters. Furthermore, the authors do not present the system in an intuitive way to perform this activity.

Sousa et al. [16] highlight the importance to ensure the quality of service in financial applications. They use the capacity planning in Electronic Funds Transfer (EFT) systems. They used a set of machines hosted in Amazon EC2 to perform the experiments. The available machines were divided into two different groups. The first group hosted the EFT system, while the second group performed the requests to the target system. They considered the response time and the resource utilisation as response variable, applying different levels of workload and EC2 machines. The results showed the use of capacity planning helped to estimate the quality of service provided by the application.

The authors in [15] proposed a framework called ASTORIA to automate and simplify the evaluation of performance and scalability of RIAs (Rich Internet Applications). This framework uses scripts that automate the generation of tests. This solution allows allocating a larger number of distributed clients from virtual machines and reduces costs for conducting the assessments.

Batista et al. [5] carried out a performance evaluation of a module for resource management in a cloud environment that includes handling available resources during execution time and ensuring the quality of service defined in the service level agreement. An analysis was conducted of different resource configurations to define which dimension of resource scaling has a real influence on client requests. The results were used to model and implement a simulated cloud system, in which the allocated resource can be changed on-the-fly, with a corresponding change in price. However, the proposed module has not been validated with workloads in a real scenario. Thus, our approach enables validate other approaches, trying to ensure the system performance for both clients and providers.

Oberle and Szabo [14] propose an architecture with a focus on TaaS (Test as a Service). The prototype allows specifying the test scenario composed by the service that will be tested and the distributed nodes that will be used. A study was performed using a Web Service developed with the Jersey framework. Al-Ghuwairi et. al. [1] present a test technique to evaluate the quality of a TaaS approach. The authors argue that the cloud test requires enough amounts of resources to ensure the service performance. Nevertheless, the authors do not care about the performance of the physical resources of the TaaS provider. In this case, our approach can be used to ensure the total capacity planning in TaaS providers. In the same context, Minzhi et al. [20] proposed a platform for distributed load testing in cloud Web Services.

Tchana et al. [18] emphasise that, despite the cloud computing provide the resources needed for testing workloads, the challenge is in allocating, deploying and managing these resources efficiently. Based on the CLIF framework, the authors proposed a self-scalable BaaS (Benchmark as a Service) solution for generating different workloads.

Tabib et. al [17] discuss the challenges of integrating e-commerces with the Cloud using different types of services offered by the Cloud. The result of this integration is a final service integrated with various other cloud services. Thus, it is important to consider the performance, capacity and reliability of e-commerce due to the use of external services. Our approach draws attention to these problems and can be used for capacity planning in order to determine the performance of the final e-commerce or check the performance of each service involved.

Some of the related work depends greatly of efficient workload tests for capacity planning. Our approach is based on these requirements, and can be differentiated from the mentioned papers on the following points: it does not require analytical modelling for experimentation and neither abstraction of the target system; it is independent of specific protocols and of the technologies target systems; it has an interface that assists in the planning and execution of reproducible experiments; it allows knowing the costs and the necessary resources for the experimentation, and it is applicable in various types of Web systems. The next Section presents the background necessary to understand the remainder of the paper.

3. BACKGROUND

The motivating example for our evaluation is a service-oriented architecture prototype named WSARCH (Web Service Architecture) [2] [6] and a tool for planning and execution of experiments named PEESOS [12].

3.1 PEESOS

PEESOS [12] is a tool based on the DCA-Services architecture [13] to design and execute capacity planning in service-oriented systems environments. PEESOS helps the user to conduct a full factorial experiment [7] using a template with a set of default entries. It also provides a real workload testbed environment, where collaborative clients perform the requests to the target system. Figure 1 shows the PEESOS workflow, which can be summarised as:

1. User plans experiment;
2. PEESOS transfers the services to the target environment;
3. The target environment deploys the service and sends acknowledgement to PEESOS;
4. PEESOS transfers the client application to the collaborative environment and starts the experiment;
5. The PEESOS clients performs a set of requests to the target system. The set of requests are performed according to a previously chosen workload distribution;
6. The target environment sends the request response to the clients;
7. As soon as the programmed set of requests are performed, an acknowledgement is returned to PEESOS.

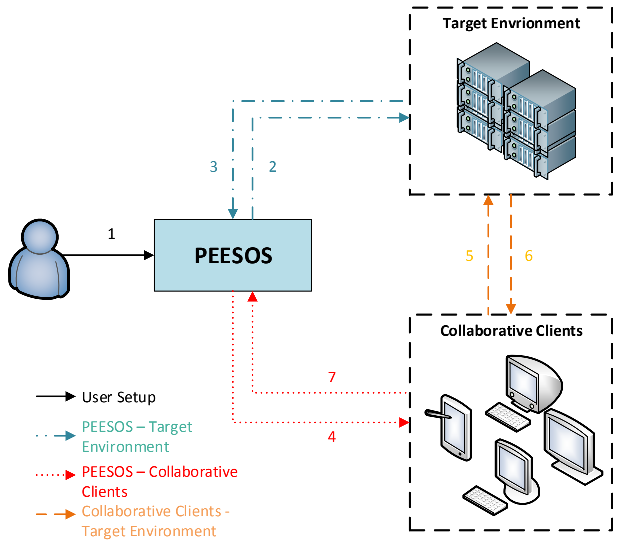

Target Envrionment
3
2
1
PEESOS
5
6
Collaborative Clients
7
4
User Setup
PEESOS – Target Environment
PEESOS – Collaborative Clients
Collaborative Clients - Target Environment

Figure 1: PEESOS Workflow [12]

[2]http://wsarch.lasdpc.icmc.usp.br

3.2 WSARCH

The WSARCH is a service-oriented architecture (SOA) prototype to provide web services with Quality of Service (QoS). WSARCH is composed of the traditional components of a SOA: the broker, the providers and the Universal Description, Discovery and Integration (UDDI) repository. Moreover, it has an additional component named logserver responsible for ensuring the web services QoS. Figure 2 shows how WSARCH works. The sequence of steps performed by it can be summarised as [6]:

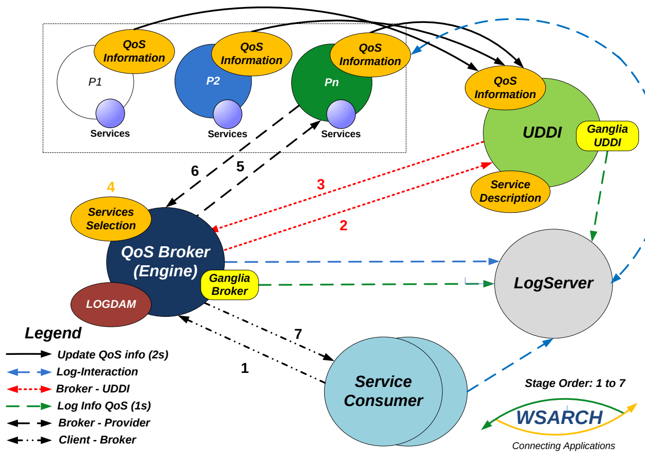


Figure 2: *Web Service Architecture* - WSARCH [6]

1) The client application requests a service with QoS attributes to the Broker; 2) The Broker requests the service address from to the UDDI repository; 3) The UDDI repository looks for the requested service and returns the available providers, which have the desired service and their QoS. 4) Then, the Broker chooses the most suitable provider according to the requested QoS attributes 5) As soon as the provider is selected a request to the desired service is performed. 6) The provider executes the required service and returns the result to the Broker. 7) Finally, the Broker returns the service response to the client application.

The *Log Server* component is a database, which store the quality attributes values during the WSARCH components transactions. In addition, the QoS attribute offered by providers are updated every second. The QoS attribute values are collected using the Ganglia monitor deployed in WSARCH and transmitted from one module to another under *Broker* management. WSARCH has a standard selector named *Default Selector*, which works directly in the *Broker*. This selector takes QoS values as parameter for provider selection and prevents them being overloaded. The service providers update these QoS values periodically. In this case, the default algorithm (Round Robin) was used for select providers.

4. PERFORMANCE EVALUATION

To validate our approach, a set of experiments will be conducted. The experiments are related to a ticket sales application hosted in the cloud, and the experimental design is based in the methodology presented by Jain [7]. Following, we need to define the system needs and after the validation of our approach.

4.1 Definition of system capacities

The capacity planning of a system demands a set of inputs, such as workload levels and resources utilisation, the cloud-based workload need to be the more realistic and scalable as possible. On the other hand, the challenge is to obtain a similar characterisation for both expected workload and performed workload.

In this sense, our study case is motivated by a ticket sale application hosted in the WSARCH architecture, as showed in Figure 3.

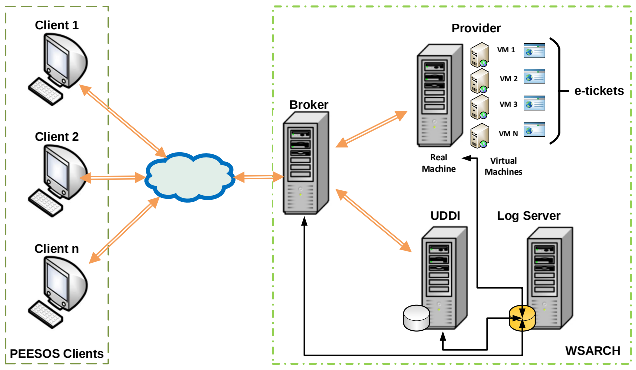


Figure 3: Request flow of the ticket sale application.

At first, as show In Figure 3, the set of clients perform several requests to the ticket sale application hosted in the WSARCH architecture. The WSARCH architecture processes the requests and returns the response to the client.

In our experiment planning, we wanted to check the behaviour of the system under different conditions. In this sense, we used 25 and 100 clients, where each client has performed five requests to the ticket sale application. Table 1 shows the WSARCH provider hardware and software settings.

Table 1: Ticket Sale Provider Configuration

Provider Configuration	
Processor	1 vCPU
Memory	1 GB
Hard Disk	50 GB
Operation System	Linux Ubuntu Server 14.04.1 64 bits LTS
Application	Apache Axis2 1.6 with Tomcat 8.0

Our workload uses an exponential distribution because it is able to perform a series of requests to the ticket sale server in a short time-period as happens in big concerts in the real world. Equation 1 represent the exponential distribution formula used to characterise our workload, where $x = 5.000$ [12].

$$f(x,\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{para } x \geq 0, \quad (1a) \\ 0 & \text{para } x < 0. \quad (1b) \end{cases}$$

The Equation 2 represents the requests acceptance rate (δ). The total of requests is given by σ, the number of clients by κ and the requests per clients is ϕ.

$$\delta = \left(\frac{\sigma}{\kappa * \phi}\right) * 100 \qquad (2)$$

Table 2 shows the percentage of requests accepted by the provider. It is observed that with 25 clients, the service provider has an requests acceptance rate by 100%. On the other hand, with 100 clients, the requests acceptance rate drops to only 71%. This demonstrates that our application needs to be adjusted because initial capacity is insufficient for demand imposed. For this, we will continue to investigate the initial capacity of the resources involved.

Table 2: Requests acceptance rate

Scalability	**Clients**	**Acceptance Rate**
Horizontal	25	100%
	100	100%
Vertical	25	100%
	100	100%

The Figure 4 shows the boxplot graph of the response time in milliseconds for the service provider . The outliers reflect the system warm-up period of Java Virtual Machine (JVM). The service processing time reflects the number of requests handled in parallel. It is possible to observe that both clients 25 and 100 clients, the response time is less than 1ms. However, the system is overloaded as more clients perform requests in a short period and then increases the service processing time.

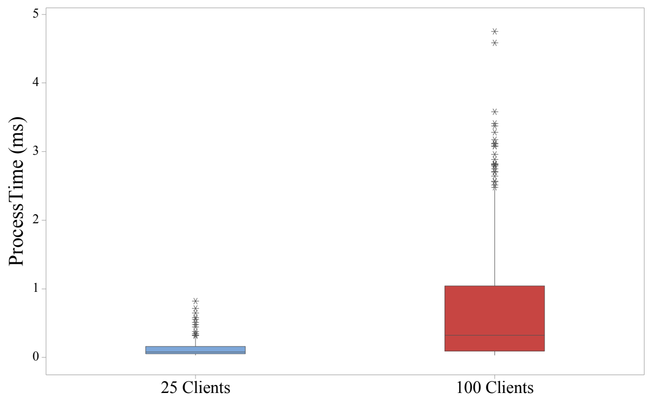
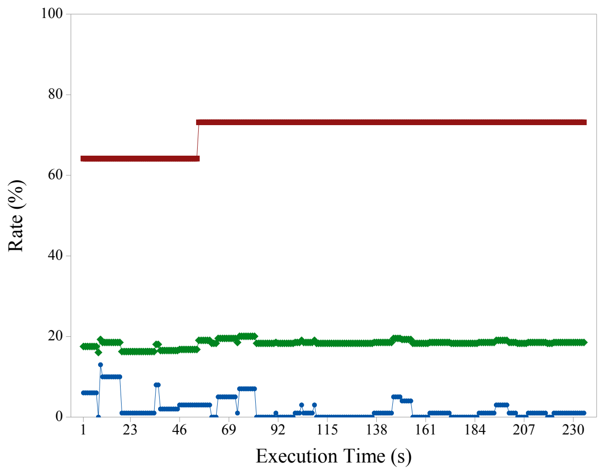


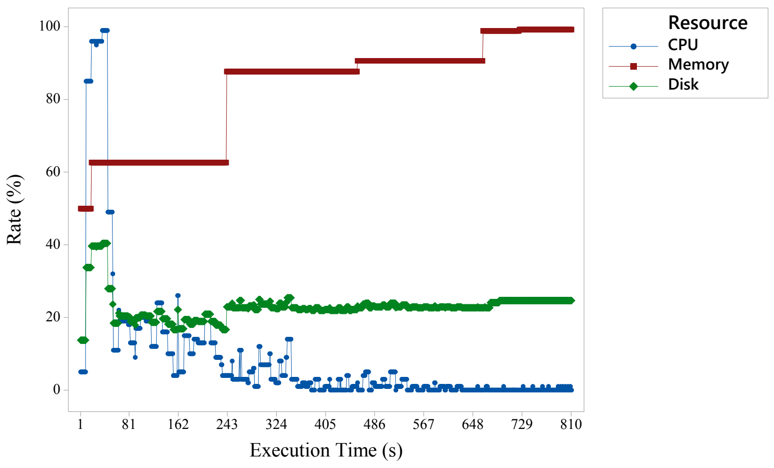
Figure 4: Boxspot processing time

It is important to identify how and why this occurs. The next results demonstrate the use of provider resources, where CPU utilisation, Memory utilisation, and disk usage rates are presented. Figure 5(a) shows the resources utilisation rate for 25 clients. It is possible to observe that the CPU utilisation rate initiates with a consumption that varies around 10%. Then, this rate is stabilised, and stays switching between 1% and 5%. When the CPU rate decreases, the memory utilisation rate increases around 65% to 73%. Finally, the utilisation rate of the disc that shows little variation during the experiment and remains at approximately 20% utilisation.

Figure 5(b) shows the resource utilisation rates with 100 clients. In this scenario, the CPU utilisation rate starts



(a) 25 Clients



(b) 100 Clients

Figure 5: Resources utilisation

close to 100% and decreases during the experiment. On the other hand, the memory utilisation ratio increases, reaching a maximum of 100%. The disk has a greater variation when the experiment started, however, it stabilises over the time.

Whereas application must to meet all clients, the results show a negative impact on service. This first evaluation is part of capacity planning. The next step is to carry out the necessary adjustments and evaluate the application, repeating this activity until the goal is reached. Thus, losses are avoided fulfilling the required expectations.

4.2 Experiment Design

According to the performance problem previously introduced, we consider two scale approaches to attend the requests: the horizontal and vertical scaling. We also plan our experiments according to the factors and levels of Table 3.

Table 3: Design of experiments

Factor	Levels	
Scalability	Vertical	Horizontal
Number of clients	25	100
Request per Client	5	
Workload	Exponential - 5000	

In the vertical scaling, the VM initially receives twice the processing and memory resources, going from 1 vCPU to 2 vCPUs and from 1GB to 2GB. On the other hand, the horizontal scaling adds a new identical virtual machine (VM). For this case, the WSARCH broker uses the Round Robin algorithm to schedule the requests. The response variables are the same defined in the previous experiment.

4.3 Results

Table 4 shows the rate of accepted requests according to the adopted scalability approach. In this case, both approaches were effective, assuring 100% of accepted requests.

The response time of requests in milliseconds of both approaches can be seen in Figure 6. Regardless of the approach and the number of clients, the processing times are very close. In general, the vertical approach presents slightly

Table 4: Requests acceptance rate

Clients	Acceptance Rate	Scalability
25 Clients	100%	100%
100 Clients	100%	100%

higher times than the horizontal approach, especially for 100 clients. In this experiment, the outliers were also preserved to observe the behaviour of all performed requests. Furthermore, is possible to note a reduction in the processing time comparing to the server without capacity planning.

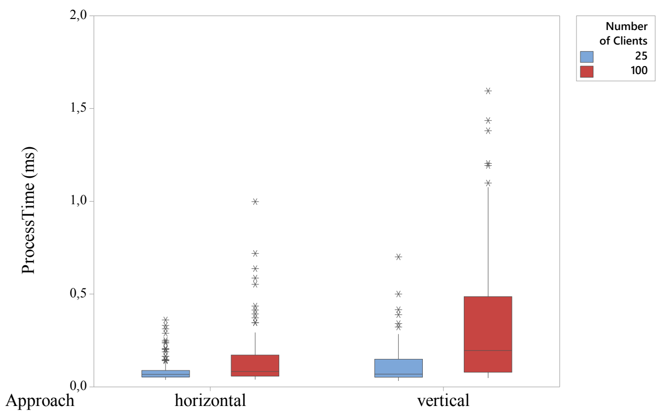
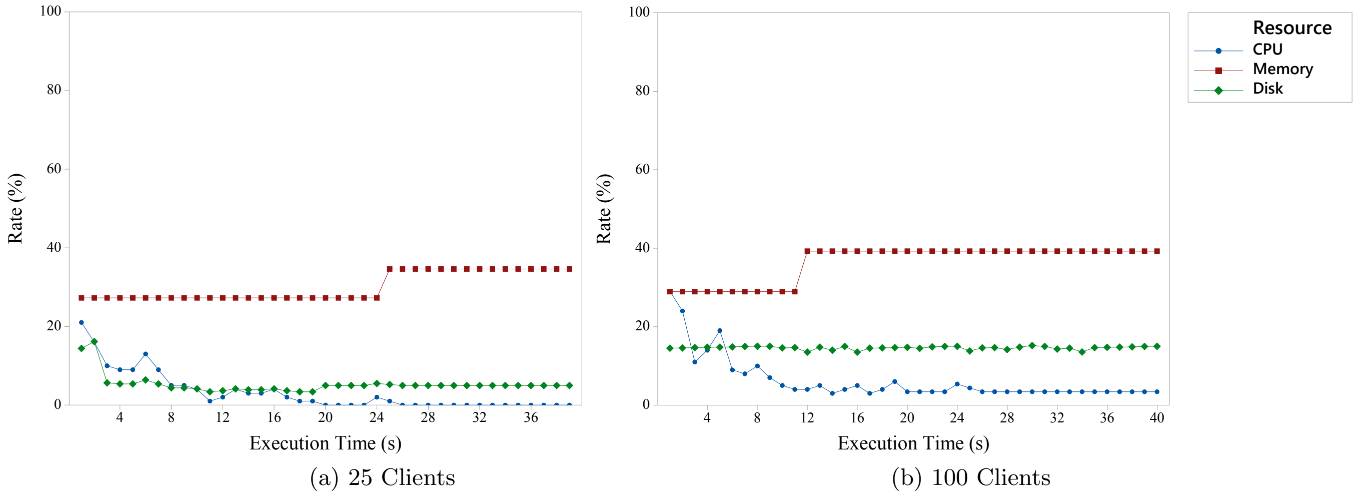


Figure 6: Boxspot processing time for vertical and horizontal scalability

Figures 7(a) and 7(b) show the resources utilisation rate according to the number of clients. The CPU utilisation rate starts with 20% for 25 clients and decrease to 4% during the experiment execution. On the other hand, the memory utilisation rate starts with 24% and increase to 35%. The disk utilisation rate varies initially and stabilises during the experiment.

Regarding 100 clients, Figure 7(b), the CPU has initially



(a) 25 Clients

(b) 100 Clients

Figure 7: Resources utilisation rate vertical approach

a utilisation rate of 29%, which decreases to 7%. For the memory, the utilisation rate starts with 30% and increases to 39%, while the disk varies around 17%. Therefore, the results with the vertical scaling shows the amount of resources are enough to attend the requests, reflecting on the accepted requests, i.e., all requests are answered. The next experiments analyse the providers 1 and 2 with the horizontal scaling approach for 25 and 100 clients.

Figures 8(a) and 8(b) show the providers utilisation rates for 25 clients. In the first provider with 25 clients, Figure 8(a), the initial CPU utilisation rate is 15%, which stabilises in 3% later. On the other hand, the memory has an initial utilisation rate of 21%, reaching 38.5% during the experiments execution. The disk has a small variation, around 8%.

Considering the provider 2, it has a similar behaviour, since it is a replica of provider 1 (Figure 8(b)). The WSARCH scheduler applies the Round Robin police, which is responsible for the same quantity of attended requests.

In general, the consumption of resources is bigger in the horizontal approach. This is because the vertical approach has only a provider with more allocated resources.

Figures 8(c) and 8(d) present the providers utilisation rate for 100 clients. The results are similar in both provider 1 and 2 with 100 clients as with 25 clients. The CPU utilisation rate initially reaches 40% and decreases varying between 10% and 16%. On the other hand, the memory has an initial utilisation rate of 30% and reaches 41% during the experiments execution. Finally, the disk utilisation varies around 10%.

Analogous the vertical scalability, horizontal scalability also ensures the acceptance rate of the resources in 100%. The indexes show that resource consumption for 100 clients is higher than for 25 clients. Regarding the vertical scalability, we noticed that the consumption rates are higher as expected.

4.4 Analysis of Results

As showed in the results of system capacities, the ticket sale application is not efficient, with high levels of CPU utilisation and reduction in the acceptance requests rate. Furthermore, the memory usage is correlated with CPU use. Usually, when the CPU levels decreases, the memory usage increases. The hard disk usage had a small variation.

Moreover, each experiment has a distinct execution time due to the requests increase, while the CPU and memory usage are directly proportional to the number of clients.

The ticket sale application uses multiple computational resources. The initial request number demands intense CPU processing, but the competition to access the database blocks the process until their answers. This behaviour explains the memory usage in the overload scenarios and justifies the non-answered requests.

The second set of experiments showed that the processing time decreases less than we expected. The used approaches aim to answer all requests. Besides the processing time, the application has the database access time to check the ticket availability and to finish the purchase. This time is not used in our evaluation due to the system architecture constraints.

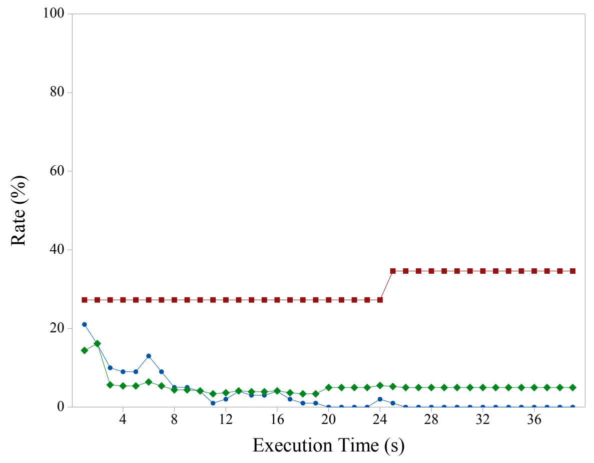
Based on the results, the processing time did not decrease proportionally to resources with the scalability measures solutions, because of the database access time. The database access time is included in all experiments, regardless of the amount of allocated resources to the VM, as the database is located in another storage machine. All requests were answered and the total time to answer them is smaller when we scale the service horizontally.

PEESOS allows the bottleneck identification stressing the applications. Furthermore, the standardisation provided according to the experiments planning allows the analysis of different scenarios and applications. In this paper, we used a maximum of 100 clients to analyse the tickets sale system.

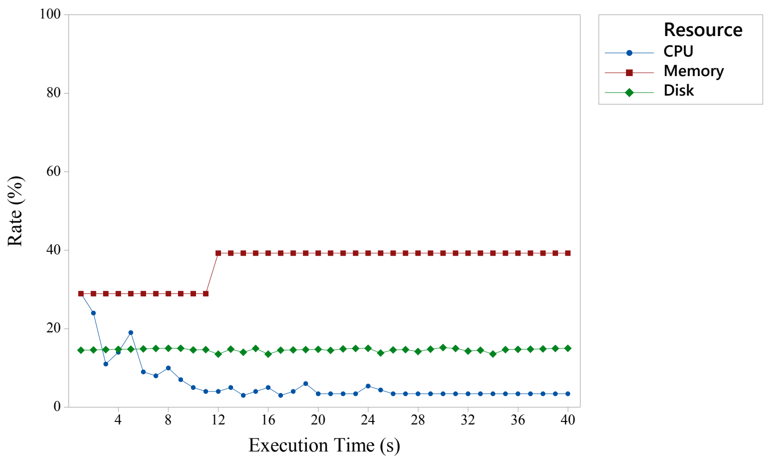
The results show how our approach is efficient to characterise workload for capacity planning of a system. Lastly, we define the cost to repeat this assessment, using a cloud provider based on the Amazon EC2 [3] prices to allocate these 100 clients. Table 5 presents these costs.

Therefore, the cost to perform this analysis is reduced, considering that we have to replicate the experiments many times to ensure the real behaviour of the system.

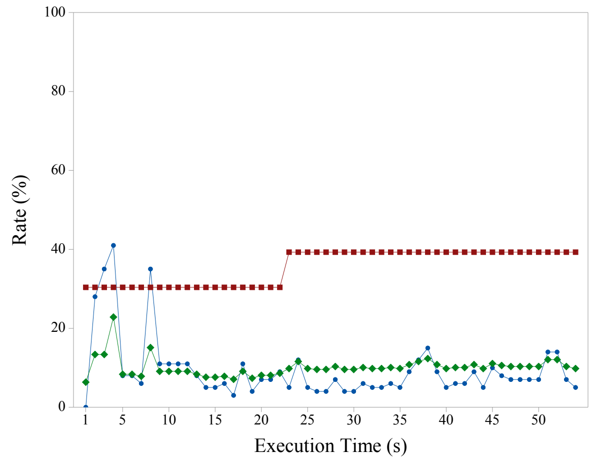
[3]http://aws.amazon.com/pt/ec2/pricing/



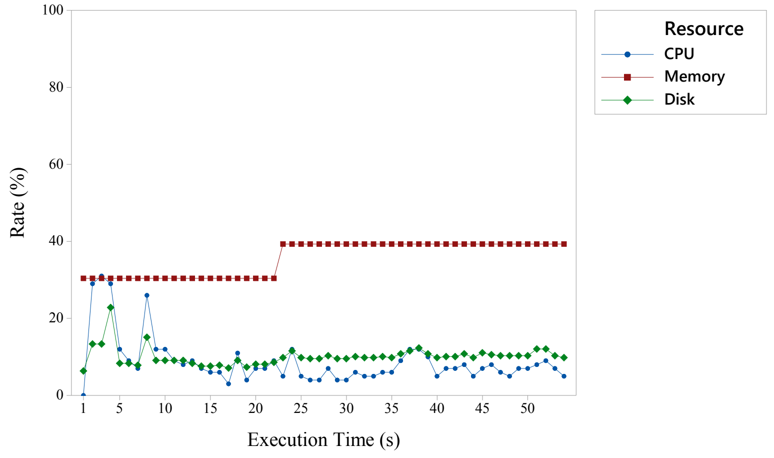
(a) Provider 1 - 25 Clients



(b) Provider 2 - 25 Clients



(c) Provider 1 - 100 Clients



(d) Provider 2 - 100 Clients

Figure 8: Resources utilisation rate horizontal approach

Table 5: Costs for the experiments

Number of instances	**Total Prince**
100 Instances	**$1,30**
200 Instances	*$2,60*
300 Instances	*$3,60*
500 Instances	*$6,50*
1000 Instances	*$13,00*

In Table 5, the first value refers the evaluation costs for our experiment followed by the estimated costs to increase the client amount. In this way. it is possible modelling and characterising the resources utilisation for precise estimates.

5. CONCLUSION

Ensure the quality of service-oriented systems is a challenging activity due of the dynamic and unpredictable environment of the distributed systems. Thus, the performance of recurring problems directly affect clients and providers. The capacity planning when applied efficiently allows ensuring the quality of service requirements and the efficient use of resources, at a fair price. In this paper, we applied a low cost workload generation approach for the capacity planning evaluation of a ticket sales service on a cloud environment.

The payment model adopted by the cloud encourages the use of our approach in two aspects. Firstly, we generate and use real workload through the cloud computing resources to characterise and manage resources efficiently. Secondly, the workload generated also must allows the extraction of the necessary parameters to ensure proper evaluation of the service. The combination of these two aspects results in an efficient workload generation that can decrease the costs of a capacity planning evaluation.

Our results explored the limits of a service, allocating more resources with vertical and horizontal scalability. We also demonstrated how the use of PEESOS tool can save money in these evaluations. Thus, we conclude that the capacity planning process is an expensive activity and the use of specialised tools leads to better use of available resources. In our studies, we used the cloud environment, however, it is possible to adopt the same methodology to other environments.

As future work, we will evaluate large-scale services with a larger amount of clients to evaluate PEESOS performance and its behaviour when we the number of clients and requests increases. We will also develop a performance model adjusting parameters in our solution and validate it through real tests.

Acknowledgment

The authors would like to thank National Council for Scientific and Technological Development (CNPQ, process 139917/2014-4) and São Paulo Research Fundation (FAPESP, processes 11/09524-7, 13/26420-6, 11/12670-5), for the support of this research.

6. REFERENCES

[1] Al-Ghuwairi, A.-R., Eid, H., Aloran, M., Salah, Z., Baarah, A. H., and Al-oqaily, A. A. A mutation-based model to rank testing as a service (taas) providers in cloud computing. In *Proceedings of the International Conference on Internet of Things and Cloud Computing* (New York, NY, USA, 2016), ICC '16, ACM, pp. 18:1–18:5.

[2] Allspaw, J. *The Art of Capacity Planning: Scaling Web Resources*. O'Reilly Media, Inc., 2008.

[3] Almeida, V., and Menasce, D. Capacity planning an essential tool for managing web services. *IT Professional 4*, 4 (Jul 2002), 33–38.

[4] Barna, C., Litoiu, M., and Ghanbari, H. Autonomic load-testing framework. In *Proceedings of the 8th ACM International Conference on Autonomic Computing* (New York, NY, USA, 2011), ICAC '11, ACM, pp. 91–100.

[5] Batista, B. G., Estrella, J. C., Ferreira, C. H. G., Leite Filho, D. M., Nakamura, L. H. V., Reiff-Marganiec, S., Santana, M. J., and Santana, R. H. C. Performance evaluation of resource management in cloud computing environments. *PloS one 10*, 11 (2015), 21.

[6] Estrella, J. C., Santana, R. H. C., and Santana, M. J. *WSARCH: An Architecture for Web Services Provisioning with QoS Support: Performance Challenges*. VDM Verlag Dr. Müller, 2011.

[7] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Professional Computing. Wiley, 1991.

[8] Jones, G. Xbox one fans warned of new xbox live outages after december online issues, December 2014. [Online; posted December 7, 2014].

[9] Lopes, R., Brasileiro, F., and Maciel, P. Business-driven capacity planning of a cloud-based it infrastructure for the execution of web applications. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on* (April 2010), pp. 1–8.

[10] Menascé, D., Almeida, V., Dowdy, L., and Dowdy, L. *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall PTR, 2004.

[11] Morgan, D., and Humer, C. Web traffic, glitches slow obamacare exchanges launch, October 2013. [Online; posted October 1,2013].

[12] Nunes, L., Nakamura, L., Kuehne, B., de Oliveira, E., Libardi, R., Adami, L., Estrella, J., and Reiff-Marganiec, S. Peesos: A web tool for planning and execution of experiments in service oriented systems. In *Web Services (ICWS), 2014 IEEE International Conference on* (June 2014), pp. 606–613.

[13] Nunes, L. H., Ferreira, C. H. G., Nakamura, L. H. V., Libardi, R. M., de Oliveira, E. M., Kuehne, B. T., Souza, P. S. L., Santana, R. H. C., Santana, M. J., Estrella, J. C., and Reiff-Marganiec, S. Dca-services: A distributed and collaborative architecture for conducting experiments in service oriented systems. *International Journal of Services Computing 3* (2015).

[14] Oberle, T., and Szabo, C. An architectural prototype for testware as a service. In *Applied Machine Intelligence and Informatics (SAMI), 2015 IEEE 13th International Symposium on* (2015).

[15] Snellman, N., Ashraf, A., and Porres, I. Towards automatic performance and scalability testing of rich internet applications in the cloud. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* (2011), IEEE, pp. 161–169.

[16] Sousa, E., Maciel, P., Souza, D., Medeiros, E., Lins, F., and Tavares, E. Capacity planning of eft service hosted on elastic iaas. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on* (Oct 2012), pp. 1749–1754.

[17] Talib, A. M., and Alomary, F. O. Cloud computing based e-commerce as a service model: Impacts and recommendations. In *Proceedings of the International Conference on Internet of Things and Cloud Computing* (New York, NY, USA, 2016), ICC '16, ACM, pp. 27:1–27:7.

[18] Tchana, A., De Palma, N., Dillenseger, B., and Etchevers, X. A self-scalable load injection service. *Software: Practice and Experience 45*, 5 (2015), 613–632.

[19] Vögele, C., Brunnert, A., Danciu, A., Tertilt, D., and Krcmar, H. Using performance models to support load testing in a large soa environment. In *Proceedings of the Third International Workshop on Large Scale Testing* (New York, NY, USA, 2014), LT '14, ACM, pp. 5–6.

[20] Yan, M., Sun, H., Wang, X., and Liu, X. 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.

[21] Yin, L., Zeng, J., Liu, F., and Li, B. Ctpv: A cloud testing platform based on virtualization. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on* (March 2013), pp. 425–428.