Stochastic Performance Model for Web Server Capacity Planning in Fog Computing

Paulo Pereira*, Jean Araujo†, Matheus Torquato§, Jamilson Dantas*, Carlos Melo*, and Paulo Maciel*
*Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil
†Universidade Federal do Agreste de Pernambuco, Garanhuns, Brazil
§Instituto Federal de Educação, Ciência e Tecnologia de Alagoas, Arapiraca, Brazil
{prps, jrd, casm3, prmm}@cin.ufpe.br*, jean.teixeira@ufrpe.br†, matheus.torquato@ifal.edu.br§

***Abstract*—Cloud computing is attractive mostly because it allows companies to increase and decrease available resources, which makes them seem limitless. Although cloud computing has many advantages, there are still several issues such as unpredictable latency, no mobility support and so on. To overcome these problems, fog computing extends communication, storage, and computation toward the edge of network. Therefore, fog computing may support delay-sensitive applications, which means that the application latency from end-users can be improved, it also decreases energy consumption and traffic congestion. The demand for performance, availability, and reliability in computational systems grows every day. To optimize these features, it is necessary to improve the resource utilization such as CPU, network bandwidth, memory, storage, and so on. Although fog computing extends the cloud computing resources and improves the quality of service, executing capacity planning is an effective approach to arranging a deterministic process for web-related activities, and it is one of the approaches of optimizing web performance. The goal of capacity planning in fog computing is preparing the system for an incoming workload, so we are able to optimize the system's utilization while minimizing the total task execution time, which happens before sending the load toward the cloud environment or not sending it at all. In this paper, we evaluate the performance of a web server running in a fog environment. We also use QoS metrics to plan its capacity. We proposed performance closed-form equations extracted from a continuous-time Markov chain (CTMC) model of the system.**

***Keywords*—cloud computing, fog computing, performance evaluation, continuous-time Markov chain, capacity planning, analytical modeling**

I. Introduction

Cloud computing allowed computing resources to be available as services, which created a new model of business [1]. Cloud providers usually charge for the resource utilization during a time period (e.g. hourly) and also permit customers to use computing resources that are in any geographic location through the Internet [2]. Currently, cloud services consumption are compared to utilities (e.g., electricity, water and gas) because the customer pays for the service without knowing about the infrastructure that provides it [3]. Most cloud services operate with the pay-as-you-go model, which has no boundaries on capacity or utilization [1]. However, cloud computing substantial yet unsolved challenges such as large end-to-end response time, which may cause lower throughput and higher discard rate. These problems are generally caused by the large distance between cloud data centers and the end-user [4]. Therefore, fog computing becomes an alternative solution to overcome these problems because it proposes to provide computing resources directly at the edge of the network, which may generate services especially for the future of the Internet. Despite the resources in the cloud and fog environments seem limitless in the user perspective, it is important to use capacity planning for determining the number of fog resources [5]. Proper fog planning can save money from the companies' budget and maximize efficiency of the provided services.

Executing capacity planning has many advantages, but companies still face challenges. An operating capacity planning problem may be that the performance of web services can be lowered down by bottlenecks in the fog environment. One definition of bottleneck can be system performance limitation as a whole by one or many components, procedures, operation or resources [6]. A fixed resource allocation may cause delays due to the conflicts between service demands and web server capacities.

Although fog environments may seem attractive because they present lower cost alternatives, there is a risk of wasting time and money. That is why one of the most important reasons to utilize proper planning strategies is financial cost. Proper capacity planning avoids paying for resources that are not being used effectively. In other words, capacity planning decreases the probability of companies spending money without necessity. Even though it seems fail-proof, in many cases the fog computing is only cheaper when used properly. Pay-as-you-go services can be very attractive due to its feature that the customers pay just for the resource they are using in a period of time [1]. Nevertheless, pay-as-you-go services may be really expensive if the customers pay for large amounts of capacity that are not being used.

In this paper, we have two main contributions: First one is to perform a performance evaluation and show how a web server in a fog node is affected by an incoming workload. The second one is to propose closed-form equations to support fog environment administrators to evaluate which element in their infrastructure is critical for the system's performance. It also allows observing how a load balancer and the web server impact on the system's throughput and CPU utilization.

This paper is structured as follows. Section II depicts some related works. Section III introduces basic concepts of fog

computing and performance evaluation. The methodology and the test environment are described in Section IV. The CTMC model is presented in detail in Section V. Section VI presents three case studies and the environment that we used as test-bed to evaluate the proposed strategy. Finally, Section VII draws some conclusions of the paper and discusses some directions for future works.

II. Related Works

Many studies have employed hierarchical modeling to represent fog computing architectures. In this section, we summarize some studies related to modeling and performance evaluation in fog computing environment. Wang et al. [7] presented an IoT-based application that manage the visualization of users in a social Virtual Reality (VR)-based Learning Environments (VRLEs). They evaluated the cost-performance trade-off analysis for a distributed system, where they used a Markov chain to calculate the throughput, mean number of records in the system, mean number of records in the queue, and response time. However, the authors did not analyze the fog node utilization and discard rate, they did not perform a capacity planning either.

Munir et al. [8] provided an evaluation that aims to improve the performance, energy efficiency, latency and scalability for IoT applications. In this study, it is proposed an Integrated Fog Cloud IoT (IFCIoT) architecture, which supports intelligent transportation systems, environmental monitoring and real-time agricultural data analysis. Using their study, the authors enhance the performance of the applications running in the fog environment. However, their evaluation did not include throughput, discard rate and CPU utilization of the fog node.

Bhattcharya et al. [9] proposed an analytical model, where it was able to compare the performance of offloading between smartphone and a fog server. Their simulation shows that offloading to a fog computing environment may increase the performance more than offloading to a cloud server. In their work, the authors opted for using simulation, but using this technique for performance evaluation may be not clear what the trade-off is among different parameters. On the other hand, analytical models usually provide better information of how the parameters interact [10]. Urgaonkar et al. [11] proposed a Markov Decision Problem model for fog computing environment, where the authors provided a method to minimize the operational cost while keeping the desired QoS. The authors did not provide an evaluation of throughput, discard rate, server utilization, and response time.

Sarkar et al. [12] investigated the interaction between the fog computing layer and the cloud layer. They built an analytical model to evaluate the fog computing environment performance in terms of power consumption, latency, and cost. Their analytical model did not provide an evaluation of throughput, utilization, discard rate, and response time.

Krishnan et al. [13] used Raspberry Pi to implement a practical fog computing environment, where it was proposed to move the computation from the cloud server to the fog network by introducing an android app store on the network. The authors were able to improve the response time using the fog computing. The author did not evaluate other metrics, such as discard rate and throughput.

El Kafhali et al. [14] proposed an analytical model to evaluate the performance of a fog computing environment, where the dynamicity of IoT systems was captured. From their model, the authors derived formulas for key performance metrics: response time, discard rate, system throughput, CPU utilization, and the number of messages request. The authors validated their analytical model by using simulations. Although our proposed analytical model also gives the response time, throughput, discard rate and CPU utilization, we cross-validated our model using a real system environment, which we measured those performance metrics and compared with our models outputs.

Kamiyama et al. [15] proposed a fog computing environment, where the system were a smart grid distributed geographically. They proposed a performance evaluation of web servers in fog nodes. The authors also evaluated the impact of prioritizing HTTP access among fog nodes. Although they suggested an improvement of the response time, the authors did not evaluate the throughput, discard rate, and CPU utilization.

Liu et al. [16] also proposed an analytical model that aimed to evaluate the performance of heterogeneous fog environments, which are composed of containers and virtual machines. They used a continuous time Markov chain to evaluate their fog environment. However, the authors evaluated only the probability of the job immediate service, they did not evaluate other performance metrics, neither they proposed a bottleneck investigation.

Ghosh et al. [17] proposed an approach for distributed event-based analytics. The authors used CEP queries to perform over multiple event streams generated at the edge. Their main goal was to find a distributed arrangement of the CEP queries in order to increase the latency. Tadakamalla et al. [18] proposed an analytical model and a tool to discover bottlenecks in the fog and cloud computing environment. Our study proposes capacity planning and performance evaluation.

III. Background

Performance evaluation methods are important to analyze fog computing environments. This section presents a brief explanation of fog computing and performance evaluation technique.

A. Fog Computing: An Overview

Basically, fog computing is an extension of the cloud computing paradigm, where the computational resources are extended from core the core of network to the edge. It also provides services, and they can be remotely accessed through the Internet or a private network [6]. Fog computing offers a reduction of the burden of data centers in traditional cloud computing, and also a reduction of response time for the services provided (e.g. IoT applications). According to Vaquero et al. [19] fog computing is a scenario where several types of devices communicate and potentially cooperate with each other and with the network to provide storage and computation without intervention of third parties. Therefore, the fog computing may be providing network functions, new services or new applications. Fog computing can also be defined as a system-level horizontal architecture, which supports services of computing, storage and distributed resources [20].

Therefore, it may aggregate data at some access points, which have lower costs, increase efficiency, throughput, reduce data traffic, and also provide security as the data do not have to cross the Internet [21].

Fog computing also includes elements of data-processing and applications running in cloud and edge devices. It also makes the management and programming of computing networking easier. Moreover, it supports end-user mobility, resource, interface heterogeneity, and services that need low latency [22].

B. Performance Evaluation

One of the main goals of system administrators is to improve performance as much as possible while decreasing the cost. Performance evaluation is to verify a system's behavior according to some metric. So when there is a need to compare several configuration scenarios or there is a need to decide between two similar systems to know which one is better for a specific scenario, we use performance evaluation [23]. Before executing a performance evaluation, the researcher should select which technique will be used. There are three major performance evaluation methods: analytical modeling, simulation, and measurement [10].

Analytical models are basically mathematical models that are used to represent and predict a system behavior [24]. These mathematical models can be used to abstract and describe the performance of computer systems, so analytical models are more cost-effective than other methods such as simulation. Mathematical models describe system stages and calculate system behavior over a finite period of time [24]. Analytical models are constructed to understand the system behavior and to measure performance. They are also used to predict the behavior of certain elements within the system, which happens by inputting changes to different components, so bottlenecks in the environment can be identified [25]. However, analytical models also have drawbacks. They may be very complex as the state space increases, which may exhaust the system resources when trying to calculate some metric. In some cases, it is almost impossible to find a closed-form solution or very costly [10].

On other hand, simulation produces one sample path per run, hence it requires many runs. It does not return an exact result either, on the contrary, it produces only a confidence interval for the results. Analytic solution by contrast does not need multiple runs. In most situations, the simulation takes much longer than analytical models to generate results. Usually simulation is considered when the assumptions made by analytical models are not appropriate [26]. Measurement is only possible if the proposed system or a similar one already exists, so through specific tools, it is possible to monitor the system and then to suggest improvements [10].

Continuous-time Markov chain (CTMC) can be represented by a directed graph with labeled transitions, indicating the probability or rate at which such transitions occur [27]. In Markov chains, the states represent different conditions that the system may follow. The transitions between the states indicate the occurrence of events, such as the arrival of tasks or completion of service. If we observe the number of busy servers as a time function, we can consider it as a random variable or function $X(t)$. Each modification of *X* over *(t)* is called state $X_n(t)$. The set of all possible states is the state space of the model. Thus, it is possible to find the transition probabilities from a state to its successor $X_{n+1}(t)$. However, it is needed to specify the probability distribution function of $X_n(t)$. In this study, we focus on analytical modeling using Continuous-Time Markov Chains (CTMC), but we use measurements to validate our model.

IV. Methodology and System Description

This section presents a proposed methodology and a fog computing environment, which we used to evaluate the modeling proposal for capacity planning and performance evaluation. Our methodology is based on [28], [29], [30], [31], and it proved to be effective. Following this methodology, we are able to determine a way to obtain the information that we need to propose an analytical model. The methodology is divided into the following macro-tasks: defining performance metrics, monitoring the system, building closed-form solutions, validating the model, and finally analyzing the results. This section also describes the system built to validate the closed-form solutions. Figure 1 depicts an organization chart that summarizes the methodology used in this paper.

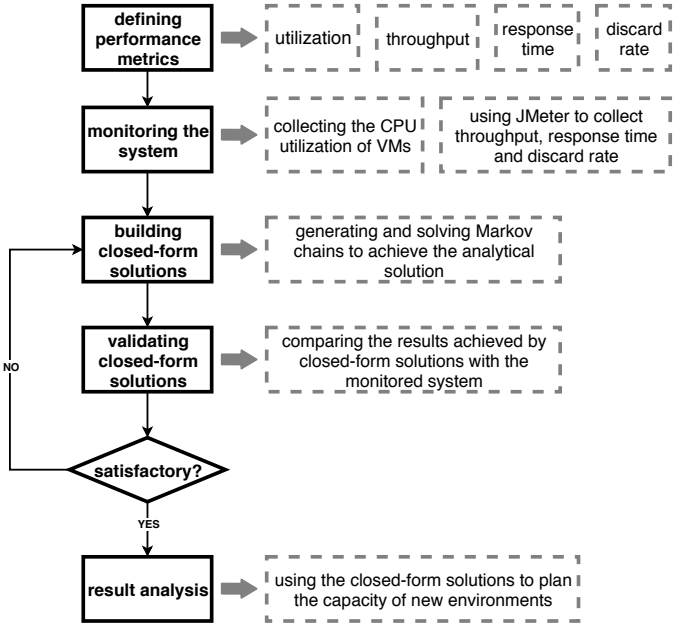


Fig. 1: Supporting methodology for the analytical models.

Each macro-task is subdivided into micro-activities. The first macro-task is where we define which performance metrics we are interested in. To start this task, we have first to understand the system. Figure 4 depicts a generalization of our environment. In this study, we are interested in four metrics: utilization of virtual machines (VMs), throughput, response time and discard rate of TCP requests to a web server. After defining the metrics of interest, we build our experimental environment and start monitoring it.

Our experimental environment was composed of two physical machines. One physical machine, which is used to generate the load, runs Elementary OS 0.4.1 Loki with 8 GB RAM

memory, 1 TB of storage and Intel i5 CPU 3.20 GHz. The other physical machine, which was used to run the virtual machines, have 8 GB RAM memory, 1 TB of storage, Intel i5 CPU 3.20 GHz and run XenServer 7.2.0 as the hypervisor platform and the Hostapd as software access point with an antenna Alfa AWUS036NH. The Figure 2 depicts the concept of the environment.

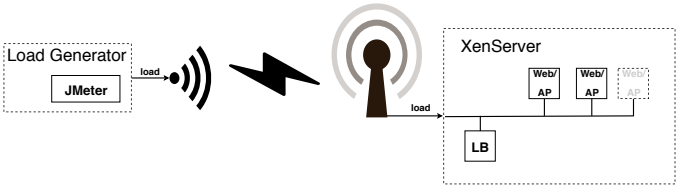


Fig. 2: The experimental environment.

We have two types of virtual machines (VMs): a Load Balancer with 1 core CPU, 1GB RAM Memory, Linux (Ubuntu Server 16.04), Nginx 1.12.2 and a Web Server with 1 core CPU, 1GB RAM Memory, Linux (Ubuntu Server 16.04), Apache 2.24. These two types of virtual machines are kept in the internal storage of XenServer as VM templates, which makes it easier to instantiate new virtual machines when it is needed. When the system scales out, a web server VM is created and instantiated from a template, and when the system scales in, a virtual machine is destroyed and its storage is erased from the server.

It is worth pointing out that the load balancing mechanism adopted was round-robin, which assigns to each server an equal amount of requests in circular order, handling all servers without priority. In other words, if a system has three web servers and one request comes, it will be sent to the first web server; when the second request comes, it will be sent to the second web server; when the third request comes, it will be sent to the third web server; when the fourth request comes, it will be sent to the first web server, and so on. Therefore, the requests are distributed equally to the application servers.

After we built the experimental environment, we conducted several case studies by generating workload using Apache JMeter™[1] 4.0 benchmark. We monitored the system, we collected the CPU utilization of each VM, and using the benchmark, we were able to collect the other metrics of interest. It is worth pointing out that we assumed that the web-service utilization is the average CPU utilization of VMs.

To build the closed-form solutions we first generated a queue network model to better understand the internal behavior of whole service, Figure 4. After that, we generated and solved a Markov chain, Figure 3,where we were able to find some patterns, which made us able to develop closed-form solutions for the metrics of interest.

The validation macro-task is the activity of comparing the output values of the closed-form solutions with the monitored metrics of the experimental environment. In this task, we compare statistically if our model is equivalent to the real system. The first case study shows step-by-step how the validation was performed. When the values of closed form solutions are satisfactory, we are able to used them to evaluate the performance of our system under different settings, and we are also able to plan the capacity of new environments.

[1]http://jmeter.apache.org/

V. Continuous-time Markov Chain Model

The continuous-time Markov chain (CTMC) depicted in Figure 3 is a performance model of a web service deployed in our experimental environment. This model captures the main activities of the system, from client requests to service completion. In this CTMC model, we did not represent the auto-scaling mechanism due to the difficult to represent instant utilization as a metric to instantiate new virtual machines.

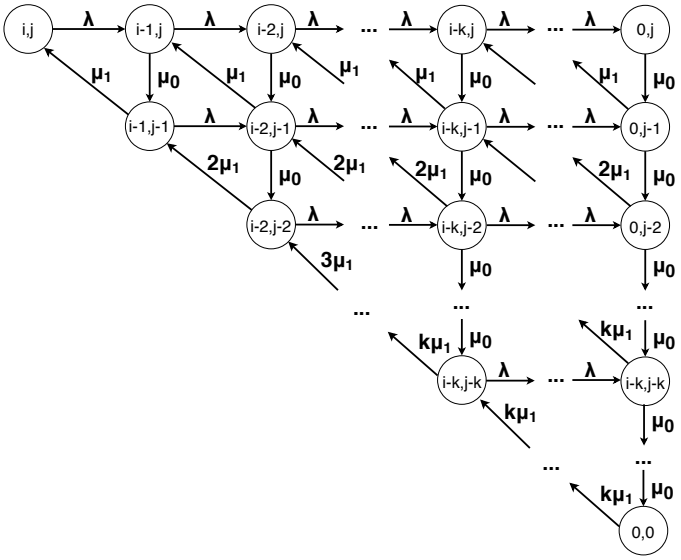


Fig. 3: Markov chain.

This continuous-time Markov chain (CTMC) model was proposed in order to achieve a closed-form solution for calculating the CPU utilization, throughput, response time and discard rate of a system with *n* virtual machines. Using closed-form solutions, we are able to plan infrastructures of any size. One important generalization of our environment can be represented by a network of queues. Such networks are able to model problems of contention that arise when a request is processed by more than one component in the system [32]. For instance, any request that arrives in the system is processed by the load balancer first, then it is processed by the web server. The Figure 4 consists of two components with service rates, μ_0 and μ_1, which correspond to the load balancer and web server processing rates, respectively. The external arrival rate is λ.

The output of the node labeled **LB** is the input to the node labeled **WS**, where these nodes correspond to load balancer and web server, respectively. We assume that service time at both nodes is exponential, and the arrival is also exponential. The state changes when one of the two servers completes some job. Since by our assumptions, all times are exponentially distributed, it follows that the stochastic process is a homogeneous continuous-time Markov chain. The Figure 3 shows the Markov chain that represents our system. The *i* and *j* indices correspond to the amount of request that the load

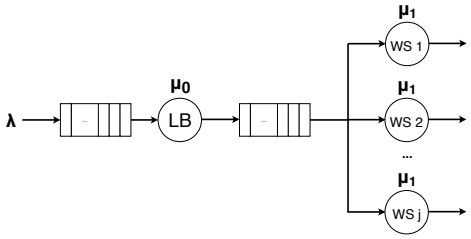


Fig. 4: Queue network in a Fog node.

balancer and the web server can handle, in other words, these indices represent the size of the queue of the components, where *i* is the size of the load balancer's queue, and the *j* is the size of the web server's queue.

We chose to use Markov chain due to its characteristics of randomly determined process, which represents better our system than a deterministic modeling technique. Markov chain widespread in many fields of engineering and sciences [33]. Based on the continuous-time Markov chain (CTMC) model presented in Figure 3, we are able to obtain the CPU utilization, the throughput, discard rate, and the response time of the system by calculating the probability of the states of the Markov chain. In order to calculate the CPU utilization, we sum the probabilities of the states that represent one or more requests in the web server queue. In other words, there is not any request in the queue of the web server in the first row of the Figure 3. It implies that we are able to calculate the CPU utilization of the web server by subtracting the probability sum of the states in the first row from 1.

In order to calculate the throughput of the system, we sum the probabilities of the states that represent one or more requests in the web server queue times the rate of processing the request (μ_1). It is important pointing out that it may have several web servers running concurrently. Therefore, there is the probability of finishing more than one request at the same time. It implies that we are able to calculate the system's throughput by summing the probability of the states times the amount of requests in the queue that the state represents times the request processing rate (μ_1). .

For the response time, we use the Little's formula [32], which states that the mean number of jobs in a queuing system in the steady state is equal to the product of the throughput and the response time. Assuming that N is the mean number of jobs in system and TP is the throughput and RP is the response time, Little's formula gives us $N = TP \times RT$. Hence, we have:

$$RT = \frac{N}{TP}$$

Thus, we are able to manipulate the input parameters of the system and test how they affect our environment. It is worth pointing out that working with closed-form solutions may decrease the amount of time that is used to evaluate an environment. Once we find the closed-form solutions, it is faster to use them to evaluate a system than to perform simulations or measurements.

Expanding the concepts of the previous paragraphs to obtain Equations in function of the CTMC parameters, we achieve the Equations 1, 2, 3 and 4, which are the closed-form solution for calculating the CPU utilization (U), the system throughput (TP), response time (RT) and discard rate (DR) respectively, in function of the queue size of the components.

Utilization:

$$U = 1 - \left[(n-1)! \mu_1^{n-1} \sum_{k=0}^{n-1} a_k \mu_0^{n-1-k} \right], \tag{1}$$

Throughput:

$$TP = \sum_{l=0}^{n-2} \sum_{k=n-1-l}^{n-1} \mu_1 (n-1)! \mu_0^{2(n-1)-(k+l)} \mu_1^l a_k, \tag{2}$$

Response Time:

$$RT = \frac{\sum_{h=0}^{n-1} \sum_{l=n-1-h}^{n-1-h} \sum_{k=n-1-l}^{n-1} (h+k) \frac{(n-1)!}{h!} \mu_0^{2(n-1)-(k+l)} \mu_1^l a_k}{TP}, \tag{3}$$

Discard Rate:

$$DR = \lambda \frac{\sum_{j=0}^{n-1} \frac{(n-1)!}{(n-1-j)!} \lambda^{(n-1)} \mu_0^{n-1-k} \mu_1^j}{\sum_{j=0}^{n-1} \frac{(n-1)!}{(n-1-j)!} \lambda^i \mu_1^j b_j}, \tag{4}$$

with

$$a_k = \frac{\lambda^k}{\sum_{j=0}^{n-1} \frac{(n-1)!}{(n-1-j)!} \mu_1^j b_j}, \qquad 0 \leq k \leq n-1,$$

and

$$b_j = \sum_{i=n-1-j}^{n-1} \lambda^i \mu_0^{2(n-1)-(i+j)}, \qquad 0 \leq j \leq n-1.$$

VI. Case Studies

To evaluate our proposal, we built an environment and performed two case studies, which the first one support that our closed-form solutions represent a fog computing environment. In other words, the first case study is to validate our analytical model. The second represents how our closed-form solutions may be used to perform a capacity planning, how it can be used to find a bottleneck on the system environment.

The buffer size of the load balancer is of 16 KB, so it can store around 80 HTTP requests, assuming that a simple HTTP request has 200 Bytes [34]. On the other hand, the Apache 2.24 Web Server has a component called ListenBackLog, which is the maximum length of the queue of pending requests. Its size is of 511 requests by default [35]. However, the Linux kernel changes the size of this queue. For instance, if the value of a socket's listen-backlog exceeds that of net.core.somaxconn sysctl value (defaults to 128 on stock builds), the kernel quietly shrinks the socket's listen-backlog to net.core.somaxconn [36]. Therefore, we consider that our Web Server buffer has a size of 128 requests for each virtual machine. So the buffer of a web server environment depends on how many virtual machines are instantiated.

TABLE I: CTMC parameters.

Parameters	Description
λ	Mean time between arrivals
μ_0	Time for Load Balancer forward request
μ_1	Time for Web Server process request

The rates assigned to all timed transitions in the CTMC model is presented in Table I. The values for rates of μ_0 and μ_1 were obtained in a test-bed that will be described in next section. The Apache JMeter benchmark [37] was used in these tests. For λ, the rate between requests arrivals was chosen according to the type of system evaluated. We configured the value of λ in JMeter benchmark.

A. Case Study I - Performance Evaluation

To validate our closed-form solution, we developed the environment described in Section IV, where we measured and performed the experiments as described. We collected two metrics: CPU utilization (in percentage), throughput (request/seconds). Then, we compared to results achieved with the closed-form solutions, which were extracted from CTMC described in Section V. The experiment lasted for about 90 hours, where three scenarios were tested.

We chose the paired sample design and used the paired difference test to compare if the real environment and the closed-form solution value means differ. In a paired design, the method of analysis takes individual differences for each pair of samples and treat the differences as a sample from a single population. The paired test is interested in the difference between the real system and the closed-form solution at each plot. So we treat each sample plot as a pair of results (Y_i^r, Y_i^m), where Y_i^r corresponds to the ith sample plot of the real system, and Y_i^m corresponds to the ith sample plot of the model, then we calculate the difference (d_i), that is $d_i = Y_i^r - Y_i^m$. These differences give us a new distribution of values that has its own sample statistics. We took a 95% confidence interval for the average of the difference, then we compared against the null hypothesis. The null hypothesis states that the mean is equal to zero, in other words, it states that there is no difference between the real system and the closed-form solution. On the other hand, the alternative hypothesis states that there is a difference between the real system and the closed-form solution.

To validate our closed-form solution we performed mea-

surements in three distinct scenarios: **1 Virtual Machine**, **2 Virtual Machines**, and **3 Virtual Machines** for processing the requests of the web server. Two metrics were measured in the real system: **throughput** and **CPU utilization**. These metrics were measured under four distinct workloads for each scenario. In other words, each scenario has four distinct workload values. Table II summarizes the measurement values.

To compare our closed-form solution to the real environment, we performed the same three scenarios described above. In other words, we collected the same metrics that we did in the real system. We collected these metrics by performing four distinct workloads in the closed-form solution for each scenario. Workloads are represented in the closed-form solution by changing the arrival rate, which is represented by λ. To represent each scenario, we increased the number $(n-1)$, where they represent the size of a web server buffer, consequently the number of virtual machines that are available to process the request in the system. The following tables show the metric values achieved using the closed-form solution.

To support that our closed-form solution represents the system, we performed paired difference test in three scenarios, where the first scenario only one virtual machine was processing the user requests, second scenario two virtual machines were dividing the user request processing, and finally the last scenario three virtual machines were dividing the processing. So comparing the metrics in these scenarios between the system and the closed-form solution, we are able to validate the model.

1) Scenario #01: Performing the paired difference test, we can compare these sets of measurements to assess whether their means differ. So using the paired test to compare the real system throughput against the closed-form solution throughput, we have the difference results in Table III.

Calculating the confidence interval of the differences, we got the following result $CI : [-66.0231, 61.0369]$. Therefore, we are 95% confident that the average of the throughput difference between the real system and the closed-form solution would be between -66.0231 and 61.0369. Zero is in the confidence interval, then we do not have evidence to refute the null hypothesis for the throughput in Scenario #01. Therefore, for this Scenario #01, we are not able to refute that our analytical model works precisely. Table IV shows the paired difference test for CPU utilization.

If we calculate the confidence interval of the differences of the utilization, we got the following result $CI : [-3.2402, 6.8758]$. We are 95% confident that the average of the CPU utilization difference between the real system and the closed-form solution would be between -3.2402 and 6.8758. Zero is in the confidence interval as well, which means that we do not have evidence to refute the null hypothesis for the CPU utilization in Scenario #01.

2) Scenario #02: For Scenario #02, we also used the paired test to compare the real system throughput against the closed-form solution throughput. So we have the difference results in Table V.

For the Scenario #02, we also calculated the confidence interval of the differences. The following result is $CI : [-324.674, 258.409]$ with 95% of confidence. The average of the throughput difference between the real system and the closed-form solution would be between -324.674 and 258.409. As we are able to see zero is in the confidence interval, which also means that we do not have evidence to refute the null hypothesis for the throughput in Scenario #02. Table VI depicts the paired difference test for CPU utilization.

Now calculating the utilization's confidence interval of the differences, we got the following result $CI : [-0.003172, 0.0032]$, also with 95% confidence. As zero is in the confidence interval, we are not able to refute the null hypothesis for the CPU utilization in Scenario #02.

3) Scenario #03: Applying the same methodology for Scenario #03, we were able to achieve the difference results in Table VII for the throughput.

In Scenario #03, we first calculated the confidence interval for the average of the throughput difference between the real system and the closed-form solution, which is -32.588 and 103.858. Therefore, zero is in the confidence interval one more time, which means that we do not have evidence to refute the null hypothesis for the throughput in Scenario #03. Table VIII shows the paired difference test for CPU utilization.

Finally, we calculated the confidence interval of the differences for the CPU utilization. The following result is $CI : [-0.006689, 0.019121]$, which means that we are 95% confident that the average of the CPU utilization difference between the real system and the closed-form solution would be between -0.006689 and 0.019121. As the others experiments, zero is in the confidence interval, which makes us conclude that we do not have evidences to refute the null hypothesis for the CPU utilization in Scenario #03.

After we performed and analyzed all these scenarios, we could confirm that zero is in all the confidence intervals that we tested, which means that we do not have evidences to refute the null hypothesis. In other words, there is no significant difference between specified population, and any observed difference is caused by sampling or experimental error. Then we can conclude that our closed-form solution represents the behavior of the real system. Therefore, we may conclude that our closed-form solution represents our real environment, so we can use it to plan the capacity of a fog environment or check for bottlenecks in the system. Either way, our closed-form solution gives a good approximation of a real system.

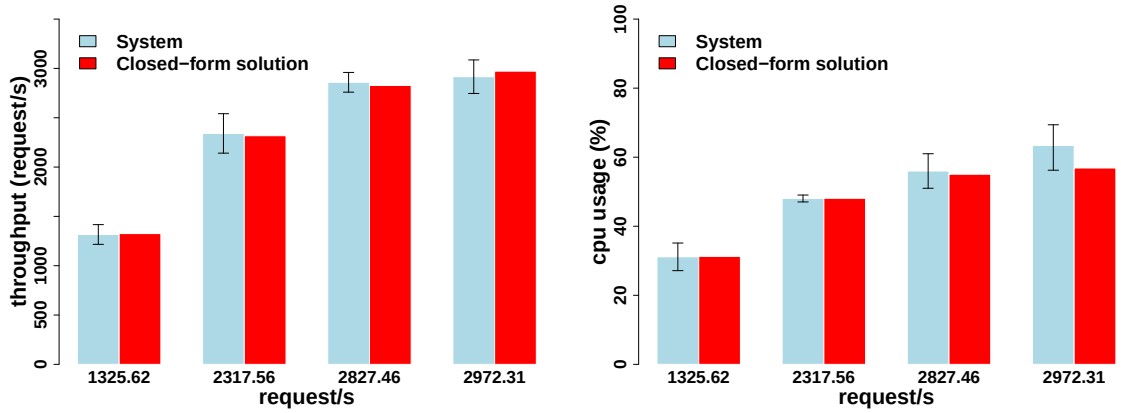
Figure 5 shows the behavior of throughput and CPU utilization when the workload is increased for Scenario #01. As we can see, the real system behavior and the closed-form solution are very similar, and they intercept each other in several points. It supports that our closed-form solution corresponds to the real system indeed.

Figure 6 depicts the behavior of throughput and CPU utilization for Scenario #02, where it was used 2 virtual machines. Although the model throughput behavior looks like linear, it also intercepts many times the real throughput behavior in several points. The same thing happens to CPU utilization for the system and the closed-form solution.

Figure 7 represents the behavior of throughput and CPU utilization for Scenario #03, where it was used 3 virtual machines. The throughput and CPU utilization for the real

TABLE II: Measurement values

Scenario	Workload(req/s)	System TP (req/s)	System utilization(%)	CTMC TP (req/s)	CTMC utilization(%)
1 VM	1325.63	1316.54	31.14	1325.63	31.26
	2317.56	2341.05	48.04	2317.56	48.09
	2827.46	2859.29	56.00	2827.46	55.06
	2972.31	2916.10	63.41	2972.30	56.87
2 VMs	2841.15	2787.89	55.25	2841.15	55.24
	3796.48	3601.98	65.91	3796.49	65.84
	4831.45	4767.11	74.98	4831.45	74.51
	5671.00	5161.12	79.42	5671.01	79.90
3 VMs	3294.26	3279.66	60.56	3294.27	60.63
	4696.00	4553.60	73.46	4696.00	73.52
	5668.10	5473.46	81.41	5668.10	79.89
	5740.71	5734.17	81.39	5740.71	80.29



(a) Throughput (request/s)

(b) CPU utilization (%)

Fig. 5: 1 VM - Scenario #01.

TABLE III: Throughput (req/s) paired test (Scenario #01)

Request/s	System	Model	$d_i = Y_i^r - Y_i^m$
1325.63	1316.54	1325.63	-9.08
2317.56	2341.05	2317.56	23.49
2827.46	2859.29	2827.46	31.83
2972.30	2916.10	2972.30	-56.20
Mean	2358.24	2360.74	-2.49
SD	741.10	745.02	39.92

TABLE IV: CPU utilization (%) paired test (Scenario #01)

Request/s	System	Model	$d_i = Y_i^r - Y_i^m$
1325.63	37.38	37.51	-0.13
2317.56	65.53	65.58	-0.05
2827.46	80.94	80.01	0.93
2972.30	90.65	84.11	6.54
Mean	49.65	47.83	1.81
SD	13.83	11.66	3.17

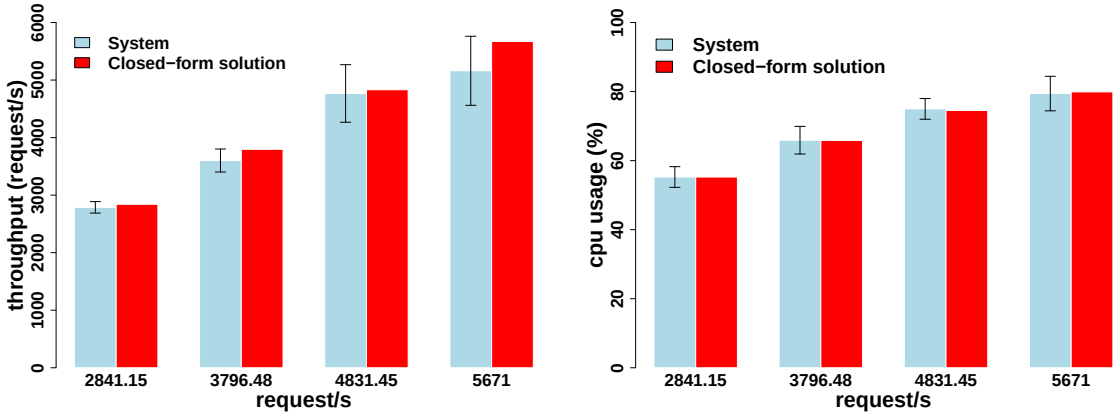
TABLE V: Throughput (req/s) paired test (Scenario #02)

Request/s	System	Model	$d_i = Y_i^r - Y_i^m$
2841.15	2787.89	2841.15	-53.25
3796.48	3601.98	3796.49	-194.83
4831.45	4767.11	4831.45	-64.33
5671.00	5161.12	5671.01	-509.88
Mean	4079.44	4285.02	-205.57
SD	1086.17	1230.57	212.81

TABLE VI: CPU utilization (%) paired test (Scenario #02)

Request/s	System	Model	$d_i = Y_i^r - Y_i^m$
2841.15	55.25	55.24	0.01
3796.48	65.91	65.84	0.06
4831.45	74.90	74.52	0.38
5671.00	79.42	79.90	-0.48
Mean	68.89	68.88	0.01
SD	10.69	10.77	0.39

system is slightly higher than the ones for the closed-form solution. However, it is still very similar, and we can see that the closed-form solution is a good approximation of the real environment. So from now on, we assume that our closed-form



(a) Throughput (request/s)

(b) CPU utilization (%)

Fig. 6: 2 VMs - Scenario #02.

TABLE VII: Throughput (req/s) paired test (Scenario #03)

Request/s	System	Model	$d_i = Y_i^r - Y_i^m$
3294.26	3279.22	3294.27	-15.05
4696.00	4553.60	4696.00	-142.39
5668.10	5473.46	5668.10	-194.63
5740.71	5734.17	5740.71	-6.54
Mean	4760.11	4849.77	-89.65
SD	1109.55	1141.15	93.58

TABLE VIII: CPU utilization (%) paired test (Scenario #03)

Request/s	System	Model	$d_i = Y_i^r - Y_i^m$
3294.26	60.56	60.63	-0.08
4696.00	73.46	73.52	-0.05
5668.10	81.41	79.89	1.52
5740.71	81.39	80.29	1.10
Mean	74.20	73.58	0.62
SD	9.83	9.17	0.81

solution represents the real system. In other words, the system and the closed-form solution have similar behavior, and we do not have evidence to refute that they are different.

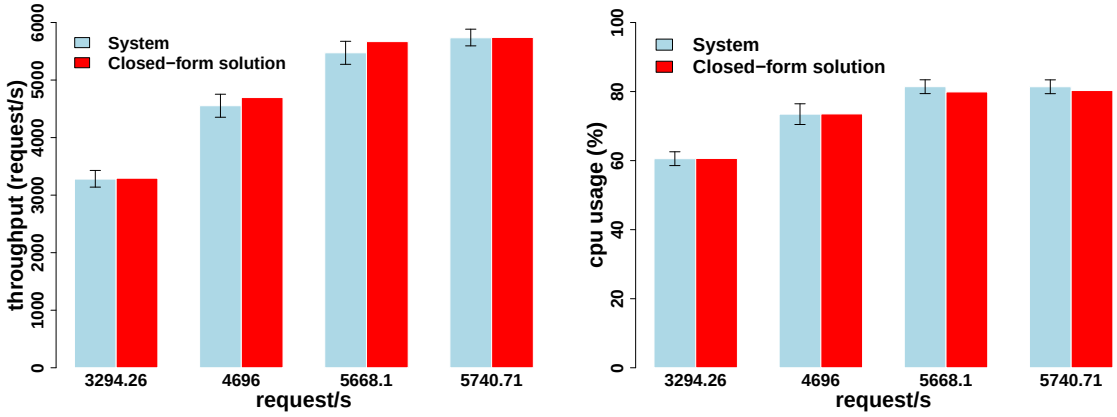
This case study performs not only a validation of the proposed closed-form solutions, but also a performance evaluation of a web server running in a fog node. The performance evaluation was restricted to 3 virtual machines at the most because it was also used as measurements to validate the analytical modeling, and it is a baseline environment. Figure 5, Figure 6 and Figure 7 show how the throughput increases as virtual machines are added to the system. From Figure 5(a) to Figure 6(a), we are able to notice that there is a considerable throughput improvement when we added a new virtual machine. On the other hand, from Figure 6(a) to Figure 7(a), adding a new VM did not affect the throughput as we expected. In other words, the main goal of the experiment is to validate the proposed analytical model, which was achieved.

B. Case Study II - E-commerce Web Site Super Bowl Day

When the request rate to a web server exceeds its capacity, the server is overloaded, its response time increases to an unacceptable level, and requests start timing out, in other words, they are abandoned, typically after some service has been received. Abandonments lead to retries, and the effective load on the server increases further. In this situation, in the absence of an overload control mechanism, the server ends up being busy doing unproductive work and the throughput degrades. E-commerce web servers, e.g., retail Web sites, often experience such overload situations, triggered by events such as closing time of a sale or intense shopping days [38].

The contemporary ways of doing business via the Internet impose pressure on E-commerce Web site's management teams to ensure high Quality of Service (QoS) levels of Web services being delivered to e-Customers. These include excellent performances and high dependability (e.g. reliability and availability) of E-commerce hardware infrastructure. This is crucial since big response times and/or frequent and high discard rate usually converts to significant financial losses. E-customers have high expectations for the user experience when buying online and have little or no patience for poor performances, low responsiveness, especially during critical time periods like Black Friday or Cyber Monday, when online shopping bursts. Using our closed-form solutions for the performability metrics, we are able to plan the capacity of the whole E-commerce system as a function of the future workload levels. Web service providers may want longer-term relationships with users of their services. These relationships generate service level agreements (SLAs), legally binding contracts that establish bounds on QoS metrics.

Assuming a hypothetical scenario, we have an E-commerce Web site hosted in a cloud. In order to reduce the burden of the cloud system, the web site is also in a fog environment to process requests from the east coast. This environment is preparing for a surge of customers because of the Super Bowl, which is taking place in New York City, and the management does not know how many customers the site will attract during the game. However, some market analysts expect about 30



(a) Throughput (request/s)

(b) CPU utilization (%)

Fig. 7: 3 VMs - Scenario #03.

to 40 million visitors more than the average load during the two-hour period right before the game. Assuming that the E-commerce Web site must respond to a request for service in 0.002s at most, and the discard rate must be as low as possible in order to not lose sales opportunities. Considering the market analysts' expectations, we intend to propose a system that can support 5,555 visitors per second more than the average load, which means that our system's throughput must be at least 5,555 requests per second.

Considering that our system is the same that the one described in the previous section, we are able to verify using our closed-form solutions how many more virtual machines are necessary to supply the demand during the two-hour period right before the Super Bowl. According to the system described in the previous section, the load balancer takes 0.000129 seconds to transfer a request to a virtual machine, and the web server takes 0.000283 seconds to complete a request. So, with that in mind, we have the following rates in Table IX.

In order to explore a better utilization of the system, we used our closed-form solution to understand the behavior of the system under the workload that will be described in this case study. We tested five scenarios, which are: 1 VM to the web server, 2 VMs, 3 VMs, 4 VMs, and 5 VMs. We also varied the arrival rate λ between 1000 requests per second and 30,000 requests per second to see how the system would behave.

Using the closed-form solution for the utilization metric, which is represented by Equation 1, we are able to analyze how the system behaves as the arrival rate increases. $n-1$ is the buffer size of the web server, which is 128 request per virtual machine. λ is the arrival rate, and μ_0 and μ_1 are the times that the load balancer and the web server take to process a request, respectively.

Figure 8(b) depicts the behavior of the system's utilization as the arrival rate increases. As we may notice, the utilization tends to be stationary, that happens because when there is no space in the buffer, the system starts to discard the requests. So the system just processes the amount that it can support in the buffer. The solution should aim at maximizing the resource utilization of the system. Ideally, a request must not be denied by the server if there is sufficient buffer in the web server.

Using the closed-form solution for the throughput metric, which is represented by Equation 2, we are able to know how many virtual machines we need to process the workload during the two-hour period right before the Super Bowl. The throughput of this system is important because it represents revenue, so the higher the throughput is the greater the revenue is.

As we can see in Figure 8(a), three virtual machines are more than enough to process the workload that market analysts expect. As we are also able to see, the throughput with our system's configurations tend to be stationary around 8,000 requests per second, which suggest that it may have a bottleneck in our system.

Analyzing the results of the closed-form solution for the discard rate metric, which is represented by Equation 4, we are able to notice how many requests we may discard per second during the two hours before the game. Analyzing this metric is very important, and it must be as low as possible in order to maximize the revenue because each request that is discarded may be a sale that the E-commerce Web Site loses.

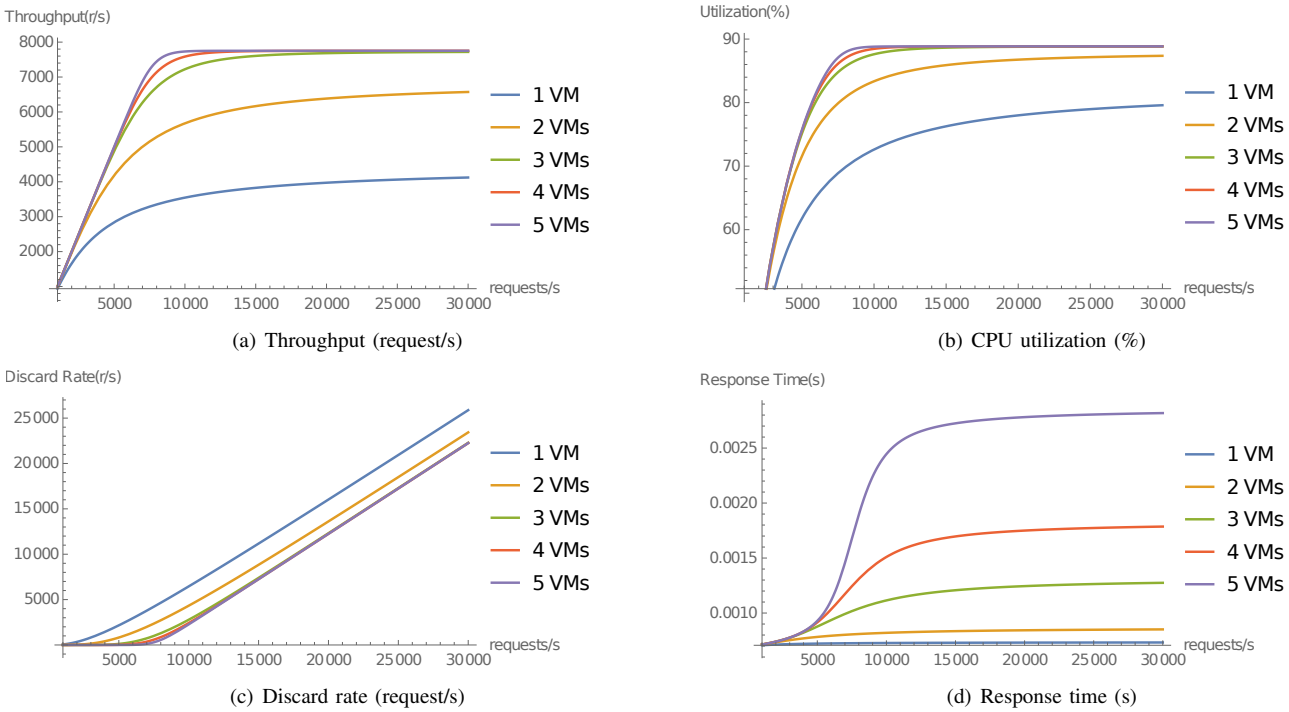
As we already expected, the more virtual machine the system has to process the arrival requests, the less the discard rate is. So analyzing the Figure 8(c), we can see that around 8,000 requests per second the system start discarding many requests. When the web server buffer gets completely full the discard rate of the system starts to increase drastically. It also suggests that it may have a bottleneck in our system.

When we analyze the closed-form solution for the response time metric, which is represented by Equation 3, we are able to see how long the system will process the request of the customer. It is important to maintain the response time as low as possible to not lose sales. The more the system takes to respond to the customer the more annoyed the customer gets, which may lead to a disbelief in the company, and regaining business credibility is very hard.

Figure 8(d) depicts the behavior of the system's response time as the arrival rate increases. As we may notice, the

TABLE IX: Input parameters of the closed-from solution

Parameter	Value	Description
λ	5,555	request arrival rate per second
μ_0	0.000129	time load balancer takes to transfer a request (s)
μ_1	0.000283	time web server takes to process a request (s)



(a) Throughput (request/s)

(b) CPU utilization (%)

(c) Discard rate (request/s)

(d) Response time (s)

Fig. 8: Performance evaluation

response time also tends to be stationary. Having a minimal response time is desirable from the perspective of the end user. The goal of the system is to ensure that the mean response time (or some percentile of the response time) seen by application requests is no greater than the desired target response. Using our closed-form solution, we can compute the feasible response time goals per system. Figure 8(d) shows that when our system has less virtual machines, the response time stabilizes quicker. That happens because the system just starts discarding requests, as we are able to see in Figure 8(c). Comparing Figure 8(d) and Figure 8(c), we see as the number of virtual machines increases, the discard rate decreases. However, when the system is getting closer to its limit, the response time increases considerably as the number of virtual machines grows because when the system gets closer to its limits, it may cause queue overflow and long response time requests due to TCP retransmissions [39].

Considering the results achieved using our closed-form solutions, we are able to infer that three virtual machines of the one described in the previous section are enough to handle the expected workload during the two-hour period right before the Super Bowl. Considering a virtual machine described in the previous section reserved on Amazon, the average cost is US$ 0.0168 per hour [40]. Therefore, the E-commerce provider would spend less than one dollar to have more 3 virtual machines for two hours. Suppose that 1 percent of the requests generate a sale and that each sale generates an average of US$ 10.00. Thus, the upper bound on revenue throughput is US$ 500.00 per second. Managers may find this type of metric more meaningful because it gives them an indication of how the system capacity and infrastructure might limit business revenue. So far, we can see how analytical modeling is important to make assumptions about what the system needs to perform in a specific scenario. It is worth pointing out that using the analytical modeling, we can know the number of VMs that are needed for this scenario.

The most important concepts in managing Web services is guaranteeing performance, availability, and return on investment, which is possible only if the system infrastructure is ready to provide high-quality service. Web services infrastructure is complex enough to preclude any guess-work when it comes to capacity planning [41]. When planning system capacity, it is very important to make sure that the system is able to handle the peak and not just the average load. Although we were able to support the number of requests expected by the market analysts, there is a bottleneck in our infrastructure, as we can observe in Figure 8(a). Therefore, if performance problems are encountered, an understanding of what is causing the slow response should be the starting point for resolving the problem, and the development approach,

architecture, utilization, and environment must be included in the search for a solution.

In this hypotetical scenario, the system has two main components, the load balancer, and the web server. In order to find which component is the bottleneck, we used our closed-form solutions varying the load balancer processing rate and the web server processing rate, then we compare to know which one has the most influence on the utilization, throughput, discard rate and response time.

It is worth pointing out that a bottleneck in a system occurs when the capacity of an application is severely limited by a single component, like the neck of a bottle that is slowing down the overall flow. The bottleneck component has the lowest throughput of all parts of the flow path of the information. In other words, bottlenecks affect the performance system by slowing down the flow of information among the components of the system. When the network speed is the bottleneck, increasing network speed generates the best performance improvement. But when the bottleneck is the server itself, the best choice depends on analyzing the components of the system.

To know which one has the most influence in the metrics, we set the arrival rate to 30,000 requests per second. We ran two times for each metric, where first time we ran varying the service rate of the load balancer from 1,000 to 30,000 request per second, keeping the web server processing time as described in Table IX. The second time we varied the web server processing rate from 1,000 to 30,000 request per second, keeping the load balancer processing time as described in Table IX. We assumed that the system has three virtual machines running.

The results are summarized in the Figure 9, where we are able to see that throughput increases as we increase the load balancer's capacity of processing requests per second. Consequently, utilization of the web server increases while the load balancer capacity grows, Figure 9(b), that happens because the load balancer processes request way quicker than the web server, so the web server consumes all its computation capacity. When we increase the capacity of the web server of processing requests, we are able to see that utilization declines considerably. The discard rate keeps constant as we increase the web server capacity, however, when we increase the load balancer processing capacity the discard rate declines, which indicates that the most responsible for the discard rate in our system is the load balancer. Figure 9(d) also shows that the load balancer is the one that most influential in the response time. Therefore, we are able to conclude that if we need to achieve better results in the throughput, discard rate and response time, we need to improve the load balancer capacity first. Consequently, the utilization of the web server grows. For this situation, using our analytical modeling, we can identify that the load balancer is critical and needs to be improved to increase the overall system's performance.

VII. Conclusions and Future Work

This study presented the performance evaluation using continuous-time Markov chain, where it was extracted closed-form equations to evaluate the performance and plan the capacity of a system. Our analysis was performed using the real environment and analytical model. Performance studies were conducted using different workloads in a real environment using closed-form solutions. The results show that the system throughput and CPU utilization may be improved effectively by focusing on a reduced set of factors, which produce a large variation on steady-state performance. A continuous-time Markov chain model is also proposed to precisely define the system structure and the interactions between the fog computing components. Important aspects of the system such as the processing rate of the load balancer and the web server were described. We also performed a bottleneck investigation to know which component has the most impact on the system QoS metrics. The main contribution of our work is closed-form solutions to plan and evaluate the performance of a fog computing environment.

In this study, we are able to conclude that the time to process requests in the load balancer has more impact in the system performance than the web server processing rate, it means that improving the virtual machines capacity of load balancer needs a higher priority. Among all the components, the load balancer is the most suitable component to improve the system performance. This approach may be used for planning private fog infrastructures, which may help to satisfy user expectations regarding system performance.

As future work, we intend to test and model techniques of auto-scaling to see how it affects the system's throughput and CPU utilization. We also intend to use other metrics of resource consumption, such as memory utilization, latency, and so on.

Acknowledgment

We would like to thank the Coordination of Improvement of Higher Education Personnel – CAPES, National Council for Scientific and Technological Development – CNPq, and MoDCS Research Group for their support. This work was also supported by a grant of contract number W911NF1810413 from the U.S. Army Research Office (ARO).

References

[1] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[2] J. Araujo, R. d. S. Matos, P. R. M. Maciel, and R. Matias, "Software aging issues on the eucalyptus cloud computing infrastructure." in *SMC*, 2011, pp. 1411–1416.

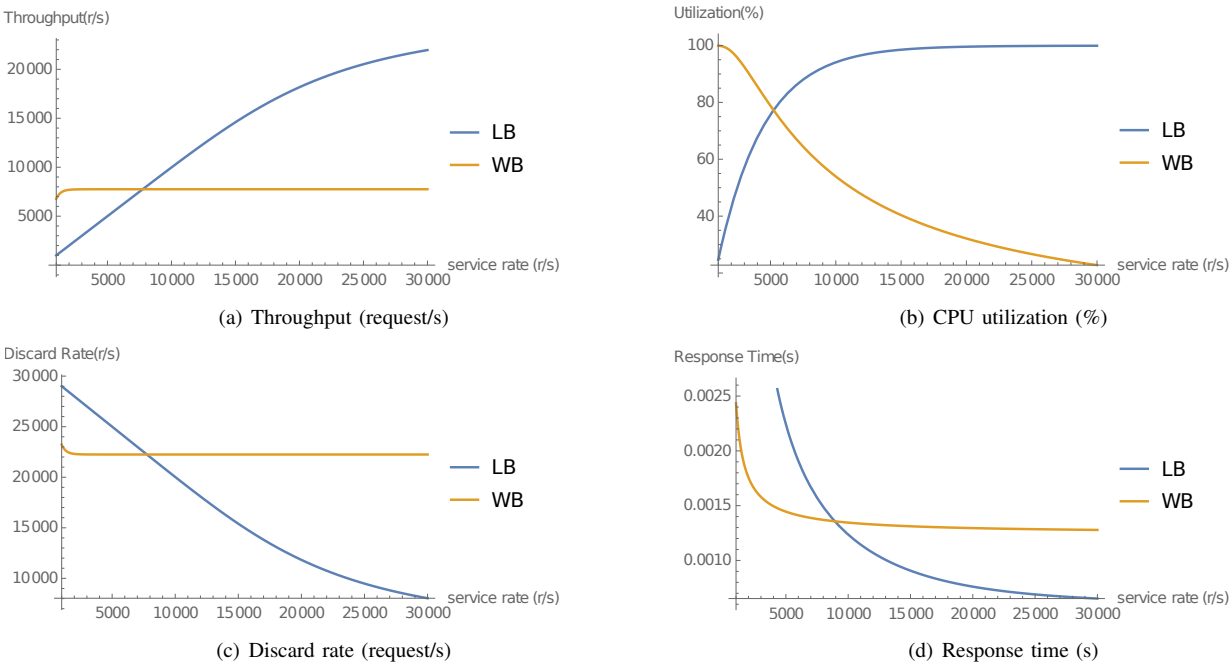
[3] E. Bauer and R. Adams, *Reliability and availability of cloud computing*. John Wiley & Sons, 2012.

[4] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*, 2015, pp. 37–42.

[5] I. Stypsanelli, O. Brun, S. Medjiah, and B. J. Prabhu, "Capacity planning of fog computing infrastructures under probabilistic delay guarantees," in *2019 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 2019, pp. 185–194.

[6] J. Araujo, R. Matos, P. Maciel, R. Matias, and I. Beicker, "Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure," in *Proceedings of the Middleware 2011 Industry Track Workshop*. ACM, 2011, p. 4.

[7] S. Wang, S. Valluripally, R. Mitra, S. S. Nuguri, K. Salah, and P. Calyam, "Cost-performance trade-offs in fog computing for iot data processing of social virtual reality," in *2019 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 2019, pp. 134–143.



(a) Throughput (request/s)

(b) CPU utilization (%)

(c) Discard rate (request/s)

(d) Response time (s)

Fig. 9: Analyzing bottleneck

[8] A. Munir, P. Kansakar, and S. U. Khan, "Ifciot: Integrated fog cloud iot: A novel architectural paradigm for the future internet of things." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 74–82, 2017.

[9] A. Bhattcharya and P. De, "Computation offloading from mobile devices: Can edge devices perform better than the cloud?" in *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*. ACM, 2016, pp. 1–6.

[10] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

[11] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, 2015.

[12] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of internet of things," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 46–59, 2015.

[13] Y. N. Krishnan, C. N. Bhagwat, and A. P. Utpat, "Fog computing—network based cloud computing," in *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*. IEEE, 2015, pp. 250–251.

[14] S. El Kafhali and K. Salah, "Efficient and dynamic scaling of fog nodes for iot devices," *The Journal of Supercomputing*, vol. 73, no. 12, pp. 5261–5284, 2017.

[15] N. Kamiyama, Y. Nakano, K. Shiomoto, G. Hasegawa, M. Murata, and H. Miyahara, "Priority control based on website categories in edge computing," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2016, pp. 776–781.

[16] B. Liu, X. Chang, B. Liu, and Z. Chen, "Performance analysis model for fog services under multiple resource types," in *2017 International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2017, pp. 110–117.

[17] R. Ghosh and Y. Simmhan, "Distributed scheduling of event analytics across edge and cloud," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 4, p. 24, 2018.

[18] U. Tadakamalla and D. Menascé, "Fogqn: An analytic model for fog/cloud computing," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. IEEE, 2018, pp. 307–313.

[19] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[20] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1826–1857, 2018.

[21] M. Verma and N. B. A. K. Yadav, "An architecture for load balancing techniques for fog computing environment," *International Journal of Computer Science and Communication*, vol. 8, no. 2, pp. 43–49, 2015.

[22] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[23] D. A. Menasce and V. A. Almeida, *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR, 2002.

[24] S. S. Gokhale and K. S. Trivedi, "Analytical modeling," *Encyclopedia of Distributed Systems*, 1998.

[25] G. V. Caliri, "Introduction to analytical modeling," in *Int. CMG Conference*, 2000, pp. 31–36.

[26] B. Tuffin, P. K. Choudhary, C. Hirel, and K. S. Trivedi, "Simulation versus analytic-numeric methods: illustrative examples," in *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2007, p. 63.

[27] R. Matias and J. Paulo Filho, "An experimental study on software aging and rejuvenation in web servers," in *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, vol. 1. IEEE, 2006, pp. 189–196.

[28] P. Pereira, J. Araujo, R. Matos, N. Preguiça, and P. Maciel, "Software rejuvenation in computer systems: An automatic forecasting approach based on time series," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2018, pp. 1–8.

[29] J. Araujo, R. Matos, V. Alves, P. Maciel, F. Souza, K. S. Trivedi *et al.*, "Software aging in the eucalyptus cloud computing infrastructure: characterization and rejuvenation," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, p. 11, 2014.

[30] J. Araujo, P. Maciel, M. Torquato, G. Callou, and E. Andrade, "Availability evaluation of digital library cloud services," in *Dependable Sys-*

tems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on. IEEE, 2014, pp. 666–671.

[31] P. Pereira, J. Araujo, and P. Maciel, "A hybrid mechanism of horizontal auto-scaling based on thresholds and time series," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 2019, pp. 2065–2070.

[32] K. S. Trivedi, *Probability & statistics with reliability, queuing and computer science applications*. John Wiley & Sons, 2008.

[33] C. Melo, J. Dantas, R. Maciel, P. Silva, and P. Maciel, "Models to evaluate service provisioning over cloud computing environments-a blockchain-as-a-service case study," *Revista de Informática Teórica e Aplicada*, vol. 26, no. 3, pp. 65–74, 2019.

[34] M. Sachdeva, G. Singh, and K. Kumar, "An emulation based impact analysis of ddos attacks on web services during flash events," in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*. IEEE, 2011, pp. 479–484.

[35] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh, "Online response time optimization of apache web server," in *International Workshop on Quality of Service*. Springer, 2003, pp. 461–478.

[36] "Apache listenbacklog," https://www.unixteacher.org/blog/linux/tuning-apache-listenbacklog/, accessed: 2018-10-19.

[37] A. JMeter, "Apache software foundation," 2018.

[38] N. Singhmar, V. Mathur, V. Apte, and D. Manjunath, "A combined lifo-priority scheme for overload control of e-commerce web servers," *arXiv preprint cs/0611087*, 2006.

[39] Q. Fan and Q. Wang, "Performance comparison of web servers with different architectures: a case study using high concurrency workload," in *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*. IEEE, 2015, pp. 37–42.

[40] E. Amazon, "Amazon web services," *Available in: http://aws. amazon. com/es/ec2/(November 2018)*, 2018.

[41] V. A. Almeida and D. A. Menascé, "Capacity planning an essential tool for managing web services," *IT professional*, vol. 4, no. 4, pp. 33–38, 2002.