Hindawi
Mobile Information Systems
Volume 2018, Article ID 1351386, 14 pages
https://doi.org/10.1155/2018/1351386

Research Article

PRESENCE: Monitoring and Modelling the Performance Metrics of Mobile Cloud SaaS Web Services

Abdallah A. Z. A. Ibrahim,[1] **Muhammad Umer Wasim**,[1,2] **Sebastien Varrette**,[1] **and Pascal Bouvry**[1,2]

[1]*FSTC-CSC/ILIAS–Parallel Computing and Optimization Group (PCOG), University of Luxembourg, 2 Avenue de l'Université, L-4365 Esch-sur-Alzette, Luxembourg*
[2]*Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg City, Luxembourg*

Correspondence should be addressed to Abdallah A. Z. A. Ibrahim; abdallah.ibrahim@uni.lu

Received 18 January 2018; Revised 4 May 2018; Accepted 21 May 2018; Published 14 August 2018

Academic Editor: Andrea Gaglione

Service Level Agreements (SLAs) are defining the quality of the services delivered from the Cloud Services Providers (CSPs) to the cloud customers. The services are delivered on a pay-per-use model. The quality of the provided services is not guaranteed by the SLA because it is just a contract. The developments around mobile cloud computing and the advent of edge computing technologies are contributing to the diffusion of the cloud services and the multiplication of offers. Although the cloud services market is growing for the coming years, unfortunately, there is no standard mechanism which exists to verify and assure that delivered services satisfy the signed SLA agreement in an automatic way. The accurate monitoring and modelling of the provided Quality of Service (QoS) is also missing. In this context, we aim at offering an automatic framework named PRESENCE, to evaluate the QoS and SLA compliance of Web Services (WSs) offered across several CSPs. Yet unlike other approaches, PRESENCE aims at quantifying in a fair and by stealth way the performance and scalability of the delivered WS. This article focuses on the first experimental results obtained on the accurate modelisation of each individual performance metrics. Indeed, 19 generated models are provided, out of which 78.9% accurately represent the WS performance metrics for two representative SaaS web services used for the validation of the PRESENCE approach. This opens novel perspectives for assessing the SLA compliance of Cloud providers using the PRESENCE framework.

1. Introduction

As per NIST definition [1], Cloud Computing (CC) is a recent computing paradigm for "*enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*" These resources are operated by a Cloud Services Provider (CSP), which typically delivers its services using one of three traditional models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), or Software-as-a-Service (SaaS) [2, 3]. This classification has since evolved to take into account the federation of more and more diverse computing resources. For instance, recent developments around Fog and Edge computing permitted to enlarge the scope of CC around Mobile CC, which offer new types of services and facilities to mobile users [4–7]. This leads to stronger business perspectives bringing more and more actors in the competition as CSPs.

In this article, we focus on the performance evaluation of Web Services (WSs) deployed in the the context of the SaaS model by these actors acting as CSPs. These services could be used through cloud users' mobile devices or normal computers [8, 9]. In practice, WSs are delivered to the cloud customers on a pay-per-use model while the performance and Quality of Service (QoS) of the provided services are defined using services contracts or Service Level Agreement (SLA) [10, 11]. In particular, SLAs define the conditions and characteristics of the provided WS and its costs and the

penalties encountered when the expected QoS is not met [12, 13]. Unfortunately, there is no standard mechanism which exists to verify and assure that delivered services satisfy the signed SLA agreement. Accurate measures of the provided Quality of Service (QoS) is also missing most of the time, which render even more difficult the possibility to evaluate on a fair basis different CSPs. The ambition of the proposed PRESENCE framework (PeRformance Evaluation of SErvices on the Cloud) is to fill this gap by offering an automated approach to evaluate and classify in a fair and by stealth way the performance and scalability of the delivered WS across multiple CSPs.

In this context, the contributions of this paper are four-fold:

(1) The presentation of the PRESENCE framework, the reasoning behind its design and organization

(2) The definition of the different module composing the framework and based on a Multi-Agent System (MAS) acting behind the PRESENCE client, which aim at tracking and modeling the WS performance using a predefined set of common performance metrics

(3) The validation and first experimental results of this module over two representative WSs relying on several reference backend services at the heart of most WSs: Apache HTTP, Redis, Memcached, MongoDB, and PostgreSQL

(4) The cloud Web Service (WS) performance metrics models such as throughput, transfer rate and latency (read and write) for HTTP, Redis, Memcached, MongoDB, and PostgreSQL.

The appropriate performance modeling depicted in this paper is crucial for the accuracy and dynamic adaptation expected within the *stealth* module of PRESENCE, which will be the object of another article. This article is an *extended* version of our presented paper during the 32nd IEEE International Conference of Information Networks (ICOIN), 2018 [14], which received the best paper award. Compared to this initial paper, the present article details the modelling of the Web Services performance metrics and brings 19 generated models, out of which 78.9% accurately represent the WS performance metrics for the two SaaS WSs.

This paper is organized as follows: Section 2 details the background of this work and reviews related works. The PRESENCE framework is described in Section 3, together with some implementation details. We then focus on the validation of the monitoring module on several reference backend services at the heart of most WSs—details and experiment results are discussed in Section 4. Finally, Section 5 concludes the paper and provides some future directions and perspectives opened by this study.

2. Context and Motivations

As mentioned before, a SLA defines the conditions and characteristics of a given WS, their costs and the penalties encountered when the expected QoS is not met. Measuring the performances of a given WS is therefore key to evaluate whether or not the corresponding SLA is satisfied—especially from a user point of view which can thus request penalties to the CSP. However, accurate measures of the provided Quality of Service (QoS) is missing most of the time as performance evaluation is challenging in a cloud context considering that the end-users do not have a full control of the system running the service. In this context, Stantchev in [15] provides a generic methodology for the performance evaluation of cloud computing configurations based on the Non-Functional Properties (NFP) (such as, response time) of individual services. Yet, none of the steps were clearly detailed, and the evaluation is based on a single benchmark, measuring a single metric. Lee et al. in [16] propose a comprehensive model for evaluating quality of SaaS after defining the key features of SaaS, deriving the quality attributes from the key features and defining the metrics for the quality attributes. This model serves as a guideline to SaaS provider to characterize and improve the provided QoS but obviously does not address user-based evaluation. However, we used part of the proposed ontology to classify our own performance metrics. Gao et al. [17] propose a Testing-as-a-Service (TaaS) infrastructure and report a cloud-based TaaS environment with tools (known as CTaaS) developed to meet the needs in SaaS testing, performance, and scalability evaluation. One drawback of this approach is that its deployment cannot be hidden from the CSP, which might in return maliciously increase the capacity of the allocated resources to mitigate artificially the evaluation in favor of its own offering. Wen and Dong in [18] propose a quality characteristics and standards for the security and the QoS of the SaaS services. Unfortunately, the authors did not propose any practical steps to evaluate the cloud services, but only a set of recommendations.

Beyond pure performance evaluation, and to the best of our knowledge, the literature around SLA assurance and violation monitoring is relatively sparse. Cicotti et al. in [19, 20] propose a quality of services monitoring approach and SLA violation reporter which are based on APIs queries and events. Called QoSMONaaS, this proposed approach is measuring the Key Performance Indicator (KPI) for the services provided from the CSPs to the cloud customers. Ibrahim et al. in [10, 21] provide a framework to assure SLA and evaluate the performance of the cloud applications. They use the simulation and local scenarios to test the cloud applications and services. Hammadi and Hussain in [22] propose a SLA monitoring framework by a third party. The third party assesses the QoS and assures the performance and no violation in the SLA. Nevertheless, none of the above mentioned approaches feature the dynamic adaptation of the evaluation campaign as foreseen within the PRESENCE proposal.

Finally, as regards to the CSPs ranking and classification, Wagle et al. in [23, 24] provide a ranking based on the estimation and prediction of the quality of the cloud provider services. The model of estimating the performance is based on the prediction methods such as ARIMA & ETS.

Motivated by recent scandals in the automotive sector, which demonstrate the capacity of solution providers to adapt the behaviour of their product when submitted to an evaluation campaign to improve the performance results,

this article presents PRESENCE, which aims at covering a large set of real benchmarks contributing to all aspects of the performance analysis while hiding the true nature of the evaluation to the CSP. Our proposal is detailed in the next section.

3. PRESENCE: Performance Evaluation of Services on the Cloud

An overview of the PRESENCE framework is proposed in Figure 1 and is now depicted. It is basically composed of five main components:

(1) A set of *agents*, each of them responsible for a specific performance metric measuring a specific aspect of the WS QoS. Those metrics have been designed to reflect scalability and performance in a representative cloud environment. In practice, several reference benchmarks have been considered and evaluated to quantify this behaviour.

(2) The *monitoring and modeling module*, responsible for collecting the data from the agent, which is used together with an application performance model [25] to assess the performance metric model.

(3) The *stealth module*, responsible to dynamically adapt and balance the workload pattern of the combined metric agents to make the resulting traffic indistinguishable from a regular user traffic from the CSP point of view.

(4) The *virtual QoS aggregator and SLA checker module*, which takes care of evaluating the QoS and SLA compliance of the WS offered across the considered CSPs. This ranking will help decision maker to determine which provider is better to use for the analyzed WS.

(5) Finally, the *PRESEnCE client* (or *Auditor*) is responsible for interacting with the selected CSPs and evaluating the QoS and SLA compliance of Web Services. It is meant to behave as a regular client of the WS and can eventually be distributed across several parallel instances even if our first implementation operates a single sequential client.

3.1. The PRESENCE Agents. The PRESENCE approach is used to collect the data which represent the behaviour of the CSP and reflect the performance of the delivered services. In this context, the PRESENCE agents are responsible for a set of performance metrics measuring a specific aspect of the WS QoS. Those metrics have been designed to reflect scalability and performance in a representative cloud environment, covering different criteria summarized in Table 1. The implementation status and coverage of these metrics within PRESENCE at the time of writing is also detailed.

Most of these metrics are measured through a set of reference benchmarks, and each agent is responsible for a specific instance of one of these benchmarks. Then a multiobjective optimization heuristic is applied to evaluate the audited WS according to the different performance domains raised by the agents. In practice, a low-level hybrid approach combining Machine Learning (for deep and reinforcement learning) and evolutionary-based metaheuristics compensate the weaknesses of one method with the strengths of the other. More specifically, a Genetic Programming Hyper-heuristic (GPHH) approach will be used to automatically generate heuristics using building blocks extracted from the problem definition and the benchmarks domains. Such a strategy has been employed with success in [26, 27], and we are confident it could be efficiently applied to fit the context of this work. It is worth to note that the metrics marked as not yet implemented within PRESENCE at the time of writing are linked to the cost and the availability of the checked service. The current paper validates the approach against a set of classical WSs deployed in a local environment and introduces a complex modelling and evaluation for the performance metrics.

As regards the *stealth* module, PRESENCE aims at relying on a GA [28] approach to mitigate and adapt the concurrent executions of the different agents by evolving their respective parameters and thus the visible load pattern toward the CSP. An *Oracle* is checked upon each iteration of the GA and based on a statistical correlation of the resulting pattern against a reference model corresponding to a regular usage. When this oracle is unable to statistically distinguish the outgoing modeled pattern of the client from a regular client, we consider that we can apply one of the found solutions for a real evaluation campaign of the checked CSP. Finally, the virtual QoS aggregator and SLA checker rely on the CSP ranking and classification proposed by Wagle et al. in [23, 24].

3.2. Monitoring and Modeling for the Dynamic Adaptation of the Agents. The first step to ensure the dynamic adaptation of the workload linked to the evaluation process resides in the capacity to model accurately this workload based on the configuration of a given agent. Modelling the performance metric will help the other researchers to generate data representing the CSP's behaviour under a high load and under the normal usage in just a couple of minutes without any experiments. In this context, the multiple runs of the agents are stored and analyzed in a Machine Learning process. During the training part, the infrastructure model representing the CSP side which contains the SaaS services is first virtualized locally to initiate the first collection of data sample and setup the PRESENCE client (i.e., the auditor) based on a representative environment. The second phase of the training involves "real" runs permitting the modeling of each metrics. In practice, PRESENCE relies on a simulation software called *Arena* [29] to analyse the data returned from the agents and get the model for each individual performance metric. Arena is a simulation software by Rockwell Corporation. It is used in different application domains, from manufacturing to supply chain and from customer service and strategies to internal business processes. It includes three modules, respectively, called *Arena Input Analyser*, *Output Analyser*, and *Process Analyser*. Among the three, the *Input Analyser* is useful for determining an appropriate distribution for collected data. It allows the user to make a sample set of raw data (e.g., latency of Cloud-based

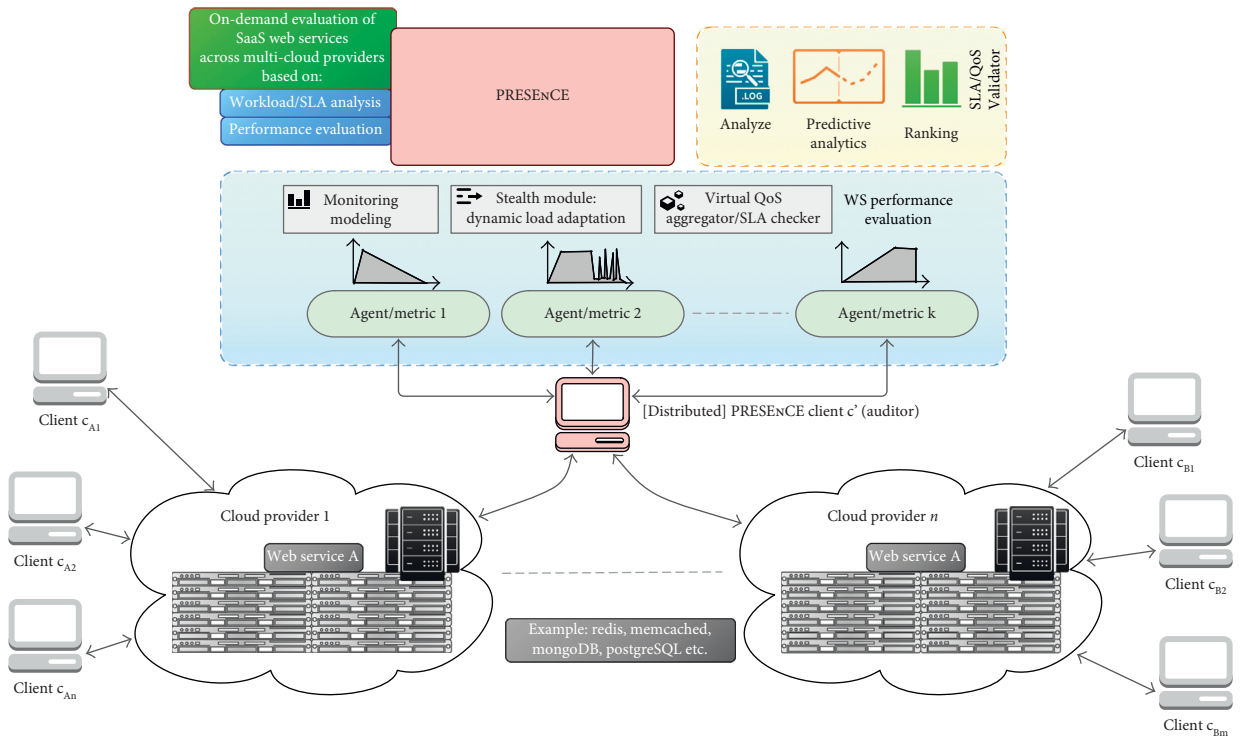


FIGURE 1: Overview of the PRESENCE framework. The figure is reproduced from Ibrahim et al. [14] (under the Creative Commons Attribution License/public domain).

TABLE 1: Performance metrics used in PRESENCE.

Domain	Metric/Implementation status		Metric type
Scalability	Number of transactions	✓	Workload/ performance indicator
	Number of requests	✓	
	Number of operations	✓	
	Number of records	✓	
	Number of fetches	✓	
Reliability	Parallel connections (clients)	✓	Workload
	Number of pipes	✓	
	Number of threads	✓	
	Workload size	✓	
Availability	Response time	×	Performance indicator
	Up time	×	
	Down time	×	
	Load balancing	×	
Performance	Latency	✓	Performance indicator
	Throughput	✓	
	Transfer rate	✓	
	Miss/hit rate	✓	
Costs	Installing costs	×	Quality indicator
	Running costs	×	
Security	Authentication	✓	Security indicator
	Encryption	✓	
	Auditability	✓	

WS) and fit it into a statistical distribution. This distribution then can be incorporated directly into a model to develop and understand the corresponding system performance.

Then, assuming such a model is available for each considered metric, PRESENCE aims at adapting its client c' (i.e., the auditor) to ensure the evaluation process is performed in a stealth way. In this paper, 19 models have been generated for each agent—they are listed in the next section. Of course, once a model is provided, we should *validate* it, that is, ensure that the model is an accurate representation of the actual metric evolution and behaves in the same way.

There are many tests that could be used to validate on the models generated. These tests are used to check the accuracy of the models by verifying on the null hypothesis. The tests are such as t-test, Wilcoxon–Mann–Whitney test, and Anova test. In PRESENCE, this is achieved by using t-test by comparing means of raw data and statistical distribution generated by the agent analysis. The use of t-test is based on the fact that the variable(s) in question (e.g., Throughput) is normally distributed. When this assumption is in doubt, the nonparametric Wilcoxon–Mann–Whitney test is used as an alternative to the t-test [30]. As we will see, 78.9% of the generated models are proved as an accurate representation of the WS performance metrics exhibited by the PRESENCE agents. In the next section, the modelling of the performance metrics are detailed besides the experiment results of PRESENCE.

4. Validation and First Experimental Results

This section presents the results obtained from the PRESENCE approach within the *monitoring and modeling* module as a prelude to the validation of the stealth module and the virtual QoS aggregator and SLA checker left for the sequel of this work.

4.1. Experimental Setup. In an attempt to validate the approach on realistic workflows, we tested PRESENCE against two traditional and core web services:

(1) A multi-*DataBase (DB) WS*, which offers both SQL (i.e., PostgreSQL 9.4) and NoSQL DBs. For the later case, we deployed two reference in-memory data structure stores, used as a database, cache, and message broker, that is, Redis 2.8.17 (redis.io) and Memcached 1.5.0 (memcached.org), as well as MongoDB 3.4, an open-source document database that provides high performance, high availability, and automatic scaling.

(2) The reference HTTP Apache server (version 2.2.22-13), which is used for testing a traditional *HTTP WS* on the cloud.

Building such a CSP environment was performed on top of two physical servers running Ubuntu 14.04 LTS (Trusty 64) connected over a 1 GbE network.

On the PRESENCE client side, 8 agents are deployed as KVM guests, that is, virtual machines running CentOS 7.3 over 4 physical servers. Each agent is running one of the benchmarking tool listed in Table 2 to evaluate the WS performance and collecting data about the CSP behaviour. Each PRESENCE agent thus measures a specific subset of performance metrics and attributes and also deals with specific kinds of cloud servers. Each measurement is consisting of an average over 100 runs collected by the PRESENCE agent to make the data statistically significant. The tools used for performance evaluation are several reference benchmarking such as Yahoo Cloud Serving (YCSB) [31], Memtire [32], Redis benchmark [33], Twitter RPC [34], Pgbench [35], HTTP load [36], and Apache AB [37]. In addition, iperf [38] (a tool for active measurements of the maximum achievable bandwidth on IP networks) is used in the closed environment for the validation of PRESENCE, as it provides an easy testimonial for the WS access capacity. The general overview of the deployed infrastructure is provided in Figure 2.

4.2. PRESENCE Agents Evaluation Results. The targeted WS of each deployed agent is precised in Table 2. The PRESENCE approach is used to collect the data which represent the behaviour of the CSP, and these data also can indicate and evaluate the performance of the services. As mentioned in the previous section, there are many metrics that can represent the performance. PRESENCE uses some of these metrics as a workload to the CSP's servers and the others as results from the experiments. For example, the number of requests, operations, records, transactions, and fetches are metrics which representing the scalability of the CSP and are used by PRESENCE to increase the workload to see the behaviour of the servers under the workload. Other metrics like parallel or concurrency connections, number of pipes, number of threads, and the workload size are representing the CSP reliability are also used by PRESENCE to increase the workload during the test. Other metrics like response time, up time, down time, transfer rate, latency (read and update), and throughput are indicating the CSP performance and availability and PRESENCE used them to evaluate the services performance. Because PRESENCE uses many tools to evaluate and benchmark the services, it can deal with most of the metrics. But, there are two or three common metrics we will model and represent them in the results, such as latency, throughput, and transfer rate. There are other metrics that represent the security and the costs of the CSPs, and all those metrics are summarized in Table 1 in the previous section. The different parameters (both input and output) which are used for the PRESENCE validation are provided in Table 3. We now provide some of the numerous traces produced by the execution of the PRESENCE agents when checking the performance of the DB and HTTP WSs.

Table 2: Benchmarking tools used by PRESENCE agents.

Benchmark tool	Version	Targeted WS
YCSB	0.12.0	Redis, MongoDB, memcached, DynamoDB, etc.
Memtire-Bench	1.2.8	Redis, memcached
Redis-Bench	2.4.2	Redis
Twitter RPC-Perf	2.0.3-pre	Redis, memcached, Apache
PgBench	9.4.12	Postgresql
Apache AB	2.3	Apache
HTTP Load	1	Apache
Iperf	v1, v3	Iperf server

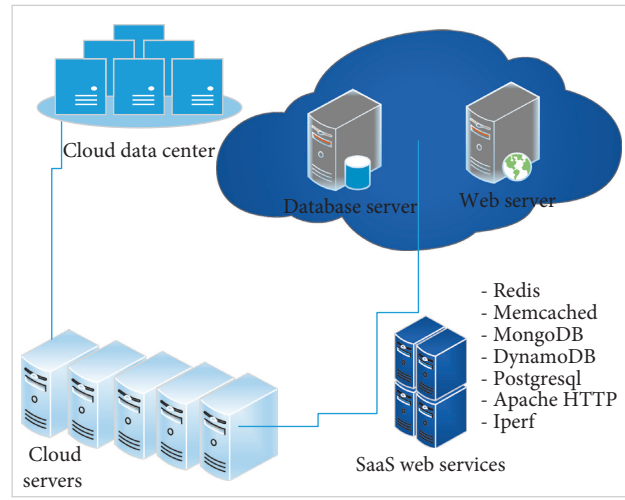


Figure 2: Infrastructure deployed to validate the PRESENCE framework. The figure is reproduced from Ibrahim et al. [14] (under the Creative Commons Attribution License/public domain).

Figure 3 shows the Redis, Memcached, and Mongo measured WS performance under the statistically significant stress produced by the PRESENCE agent running the YCSB benchmarking tool. Figure 3(a) shows the throughput of the three backends and demonstrates that the Redis WS is the best in this metric when compared to the other two WSs where it has the highest throughput. This trend is confirmed when the latency metric is analysed in Figures 3(b) and 3(c).

Figure 4 shows the Redis and Memcached measured WS performance under the stress produced by the PRESENCE agent running the Memtier benchmarking tool. The previous trend is again confirmed in our runs; that is, the Redis-based WS performs better than the Memcached backend for

Table 3: Input/output metrics for each PRESENCE Agent.

PRESENCE agent	Input parameters								
	#Transactions	#Requests	#Operations	#Records	#Fetches	#Parallel clients	#Pipes	#Threads	Workload size
YCSB			✓	✓				✓	✓
Memtire-Bench		✓				✓		✓	✓
Redis-Bench		✓				✓	✓		✓
Twitter RPC-Perf		✓							✓
PgBench	✓					✓		✓	
Apache AB		✓				✓			
HTTP Load					✓	✓			
Iperf								✓	

PRESENCE agent	Output parameters								
	Throughput	Latency	Read latency	Update latency	CleanUp latency	Transfer rate	Response time	Miss	Hits
YCSB	✓		✓	✓	✓			✓	✓
Memtire-Bench	✓	✓				✓		✓	✓
Redis-Bench	✓								
Twitter RPC-Perf	✓	✓						✓	✓
PgBench	✓	✓							
Apache AB	✓					✓			
HTTP Load	✓	✓				✓			
Iperf						✓			

all metrics, for example, throughput, latency, and transfer rate. As noticed in the three plots from the Memtier agents, the latency, throughput, and transfer rate of the Memcached WS have increased suddenly in the end. Such behaviour was consistent across all runs and was linked to the memory saturation reached by the server process before being clearer. Still upon DB WS performance evaluation, Figure 5 details the performance of the PostgreSQL WS under the stress produced by the PRESENCE agent executing the PgBench benchmarking tool. The figure shows the normalized response time of the server and the normalized (standardized) number of Transactions per Second (TPS). The response time is the latency of the service when the TPS corresponds to its throughput. The performance of the WS is affected by the increased workload which is represented by the increasing number of TPSs and parallel clients. The increasing of the TPS let the response time increasing even if the TPS was going down, and after filling in the memory, the TPS decreased again and response time returned back again to a decrease. This behaviour was consistent across the runs of the PgBench agent. Finally, Figure 6 shows the average runs of the PRESENCE agent executing the HTTP Load benchmark tool when assessing the performance of the HTTP WS. We exhibit on each subfigure the behaviour of both the latency and throughput against an increasing number of fetches and parallel clients, which increases the workload of the WS.

Many more traces of all considered agent runs are available but were not displayed in this article for the sake of conciseness. Overall, and to conclude on the collected traces from the many agent runs depicted in this section, we were able to reflect several complementary aspects of the two WSs considered in the scenario of this experimental validation. Yet, as part of the contributions of this article, the generation of accurate models for these evaluations is crucial. They are detailed in the next section.

4.3. WS Performance Metrics Modeling. Outside the description of the PRESENCE framework, the main contribution of this article resides more on the *modeling* of the measured WS performance metrics from the data collected by the PRESENCE agents rather than the runs in themselves depicted in the previous section. Once these models are available, they can be used to estimate and dynamically adapt the behaviour of the PRESENCE client (which combine and schedule the execution of all considered agents in parallel) so as to hide the true nature of the evaluation by making it indistinguishable from a regular client traffic. But this corresponds to the next step of our work. In this paper, we wish to illustrate the developed model from the PRESENCE agents evaluations reported in the previous section. The main objective of the developed model is to understand the system performance behaviour relative to various assumptions and input parameters discussed in previous sections. As mentioned in Section 3.2, we rely on the simulation software called *Arena* to analyse the data returned from the agents and get the model for each individual performance metric. We have performed this analysis for each agents, and the results of the models are presented in the below tables. Of course, such a contribution is pertinent only if the generated model is validated—a process consisting in ensuring the accurate representation of the actual system and its behaviour. This is achieved by using a set of statistical t-tests by comparing the means of raw data and statistical distribution generated by the Input Analyzer of the Arena system. If the result shows that both samples are analytically similar, then the model developed from statistical distribution is an accurate representation of the actual system and behaves in the same way. For each model detailed in the tables, the outcomes of the t-tests in the form of the computed p value (against the common significance level of 0.05) is provided and demonstrate if present the accuracy of the proposed models.

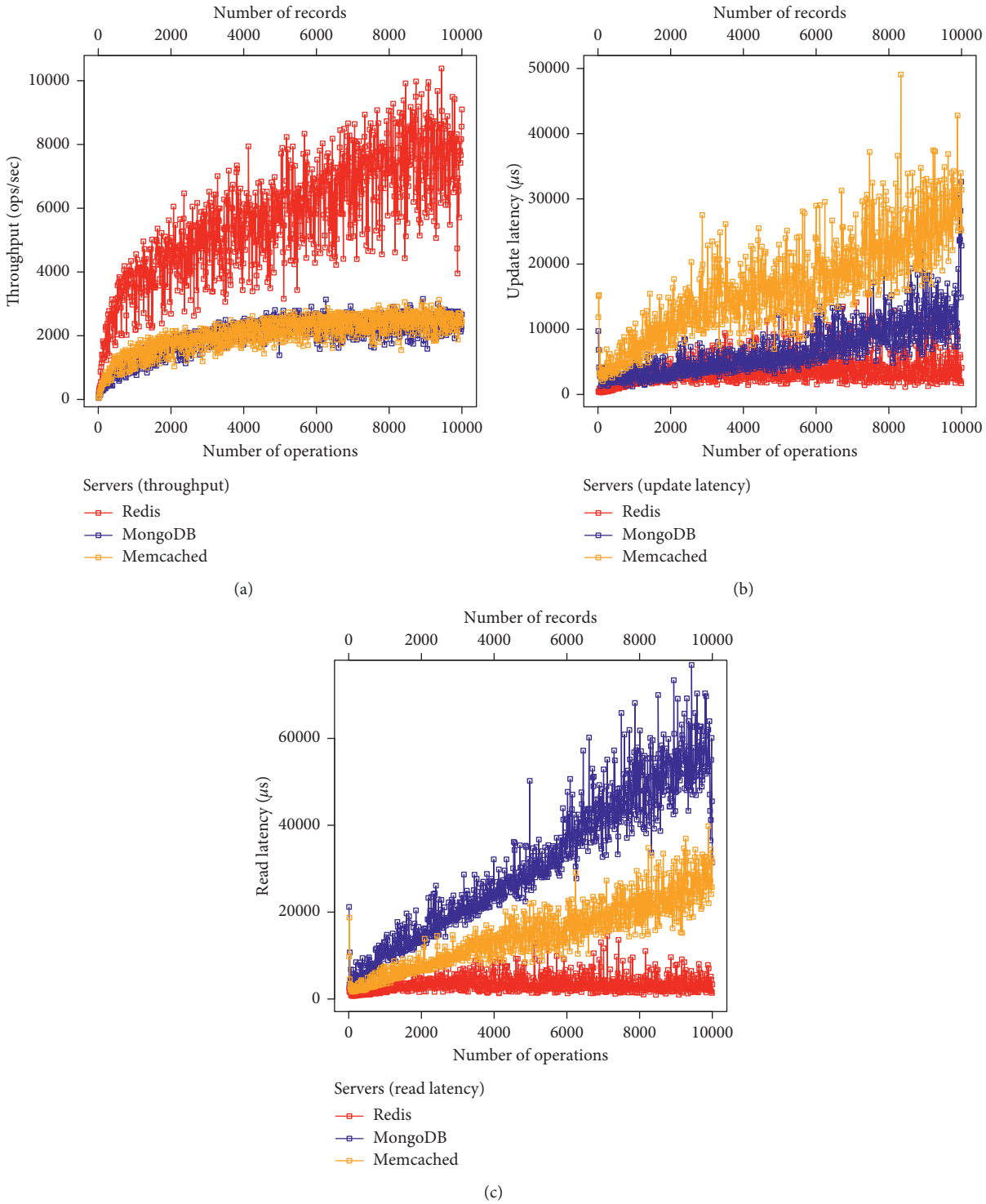


Figure 3: YCSB Agent Evaluation of the NoSQL DB WS. (a) Throughput, (b) update latency, and (c) read latency. The figure is reproduced from Ibrahim et al. [14] (under the Creative Commons Attribution License/public domain).

Tables 4–6 show the model of the performance metrics such as latency (read and update) and throughput for the Redis, Memcached, and MongoDB services by using the data collected with the YCSB PRESENCE agents. In particular, Table 4 shows that the Redis throughput is Beta increasing, Redis latency (read) is Gamma increasing, and Redis latency (update) is Erlang increasing with respect to the configuration of the experimental setup discussed in the previous sections. Moreover, as p values (0.757, 0.394, and 0.503) are greater than 0.05, the null hypothesis (the two samples are the same) is accepted as compared to alternate hypothesis (the two samples are different). Hence, the models for throughput, that is, $-0.001 + 1 * \mathrm{BETA}(3.63, 3.09)$, latency (read), that is,

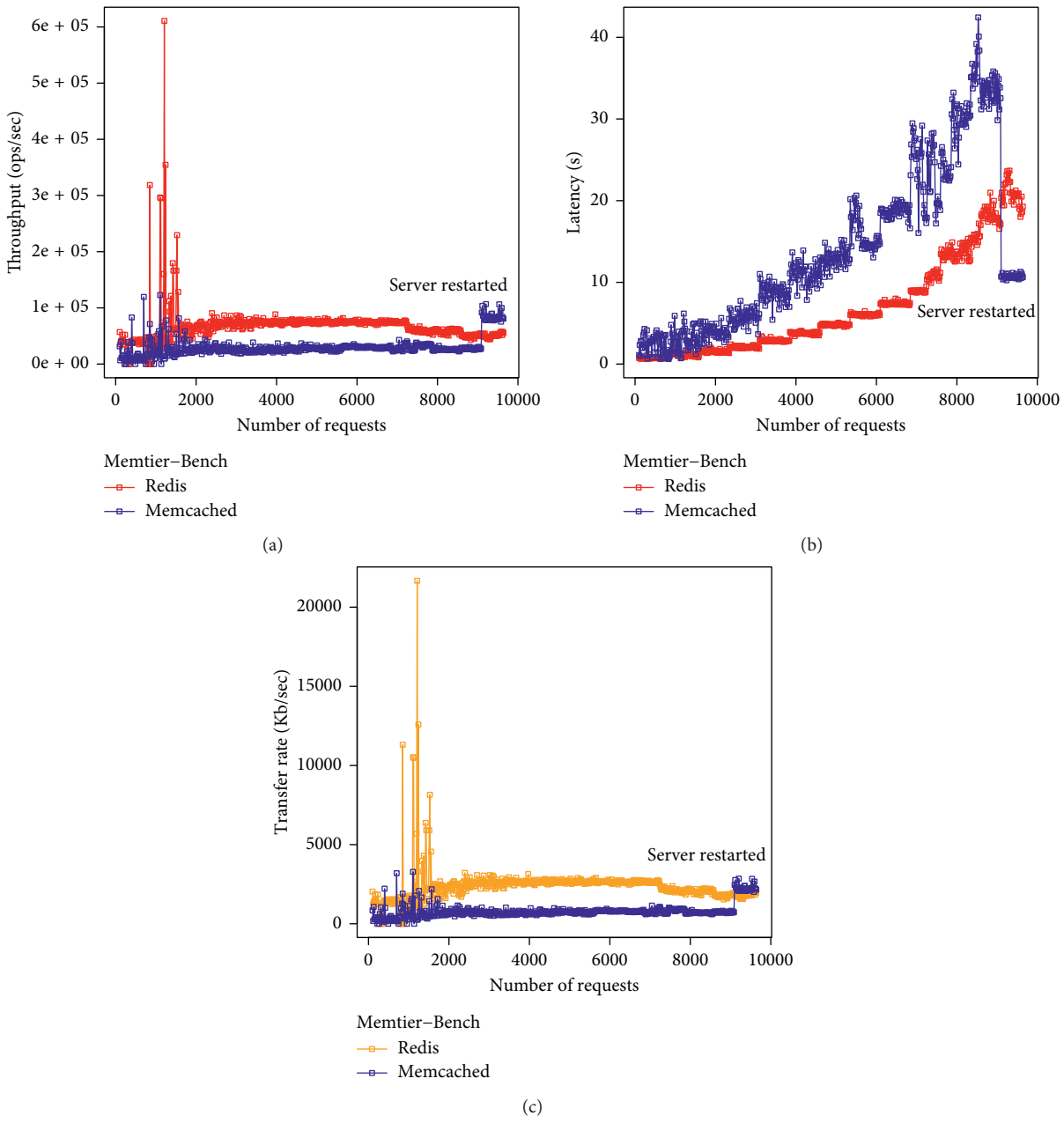


FIGURE 4: Memtier Agent Evaluation of the NoSQL DB WS. (a) Throughput, (b) latency, and (c) transfer rate. The figure is reproduced from Ibrahim et al. [14] (under the Creative Commons Attribution License/public domain).

−0.001 + GAMM (0.0846, 2.39), and latency (update), that is, −0.001 + ERLA (0.0733, 3), are an accurate representation of the WS performance metrics exhibited by the PRESENCE agent in Figure 3.

Similarly, Table 5 shows that MongoDB throughput and latency (read) are Beta increasing and latency (update) is Erlang increasing with respect to the configuration setup. Moreover, as p values (0.388, 0.473, and 0.146) are greater than 0.05, the null hypothesis (the two samples are the same) is accepted as compared to alternate hypothesis (the two samples are different). Hence, the models for throughput, that is, −0.001 + 1∗BETA (3.65, 2.11), latency (read), that is, −0.001 + 1∗BETA (1.6, 2.48), and latency (update), that is, −0.001 + ERLA (0.0902, 2), are an accurate representation of the WS performance metrics exhibited by the PRESENCE agents in Figure 3.

Finally, Table 6 shows that Memcached throughput and latency (read) is Beta increasing and latency (update) is Normal increasing with respect to configuration setup. Again, as p values (0.106, 0.832, and 0.794) are greater than 0.05, the null hypothesis is accepted. Hence, the models for throughput, that is, −0.001 + 1∗BETA (4.41, 2.48), latency (read), that is, −0.001 + 1∗BETA (1.64, 3.12), and latency (update), that is, NORM (0.311, 0.161), are an accurate representation of the WS performance metrics exhibited by the PRESENCE agents in Figure 3.

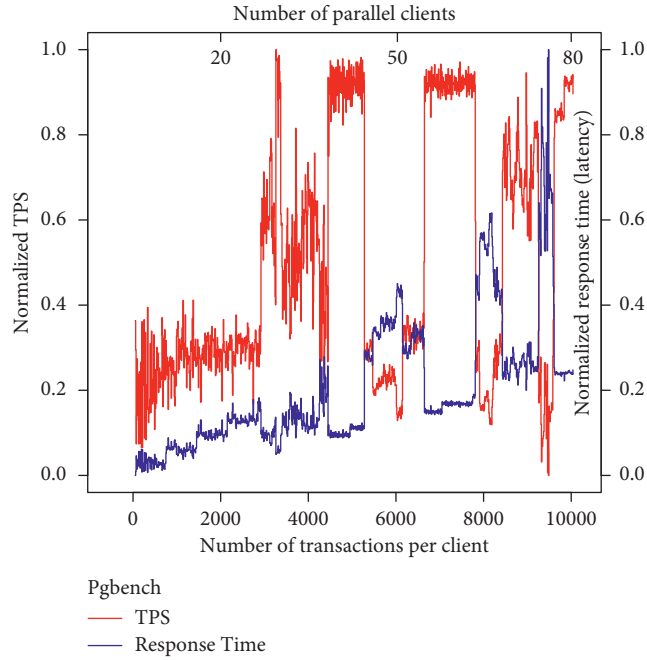


Figure 5: PgBench Agent on the SQL DB WS. The figure is reproduced from Ibrahim et al. [14] (under the Creative Commons Attribution License/public domain).

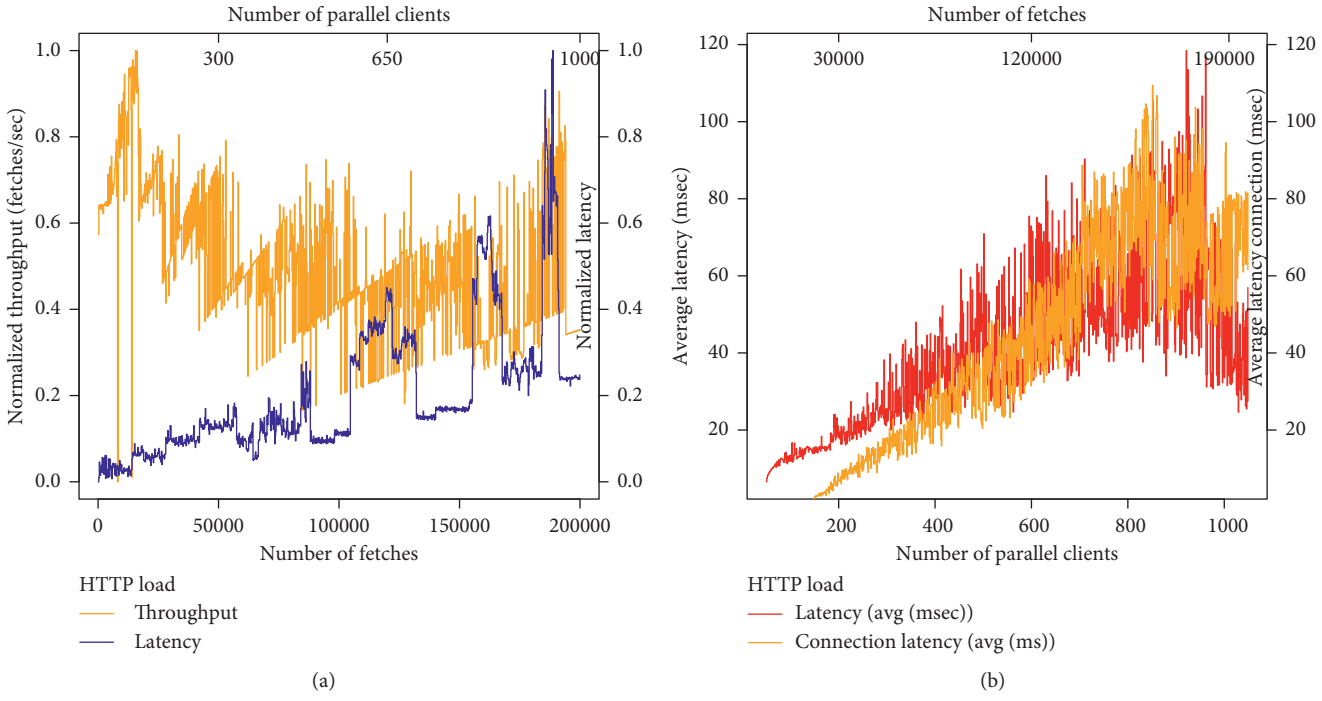


Figure 6: HTTP Load Agent Evaluation on the HTTP WS. The figure is reproduced from Ibrahim et al. [14] (under the Creative Commons Attribution License/public domain).

The above analysis is repeated for the Memtier PRESENCE agent—the corresponding models are provided in Tables 7 and 8 and summarize the computed model for the WS performance metrics such as latency, throughput, and transfer rate for the Redis and Memcached services by using the data collected from PRESENCE Memtier agents. For instance, it can be seen in Table 7 that the Redis throughput is Erlang increasing with respect to time and assumptions in previous section. Moreover, as p value (0.902) is greater than 0.05, the null hypothesis is again accepted as compared to alternate hypothesis (the two samples are different). Hence, the Erlang distribution

Table 4: Modelling DB WS performance metrics from YCSB PRESENCE Agent: Redis.

Metric	Distribution	Model	Expression	p value (t-test)
Throughput	Beta	$-0.001 + 1 * \text{BETA}(3.63, 3.09)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 3.63$ $\alpha = 3.09$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.757 (>0.05)
Latency read	Gamma	$-0.001 + \text{GAMM}(0.0846, 2.39)$ where $\text{GAMM}(\beta, \alpha)$ $\beta = 0.0846$ $\alpha = 2.39$ offset $= -0.001$	$f(x) = \begin{cases} (\beta^{-\alpha}x^{\alpha-1}e^{-(x/\beta)})/\Gamma(\alpha) & \text{for } x > 0 \\ 0 & \text{otherwise,} \end{cases}$ where Γ is the complete gamma function given by $\Gamma(\alpha) = \int_0^{\inf} t^{\alpha-1}e^{-1}\,dt$	0.394 (>0.05)
Latency update	Erlang	$-0.001 + \text{ERLA}(0.0733, 3)$ where $\text{ERLA}(\beta, k)$ $k = 3$ $\beta = 0.0733$ offset $= -0.001$	$f(x) = \begin{cases} (\beta^{-k}x^{k-1}e^{-(x/\beta)})/(k-1)! & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$	0.503 (>0.05)

Table 5: Modelling DB WS performance metrics from YCSB PRESENCE Agent: MongoDB.

Metric	Distribution	Model	Expression	p value (t-test)
Throughput	Beta	$-0.001 + 1 * \text{BETA}(3.65, 2.11)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 3.65$ $\alpha = 2.11$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.388 (>0.05)
Latency read	Beta	$-0.001 + 1 * \text{BETA}(1.6, 2.48)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 1.6$ $\alpha = 2.48$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.473 (>0.05)
Latency update	Erlang	$-0.001 + \text{ERLA}(0.0902, 2)$ where $\text{ERLA}(\beta, k)$ $k = 2$ $\beta = 0.0902$ offset $= -0.001$	$f(x) = \begin{cases} (\beta^{-k}x^{k-1}e^{-(x/\beta)})/(k-1)! & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$	0.146 (>0.05)

model of Throughput, that is, $-0.001 + \text{ERLA}(0.0155, 7)$, is an accurate representation of the WS performance metrics exhibited by the PRESENCE agents in Figure 4(a). The same conclusions on the generated models can be triggered for the other metrics collected by the PRESENCE Memtier agents, that is, latency and transfer rates. Interestingly, Table 8 reports an approximation (blue curve line) for multimodel distribution of memcached throughput and transfer rate. This shows a failure of a single model to capture the system behaviour with respect to the configuration setup. However for the same setup, memcached latency is Beta increasing, and as its p value (0.625) is greater than 0.05, the null hypothesis is accepted. Hence, the model for latency, that is, $-0.001 + 1 * \text{BETA}(0.99, 2.1)$, is an accurate representation of the WS performance metrics exhibited by the PRESENCE agents in Figure 4(b).

To finish on the DB WS performance analysis using PRESENCE, Table 9 exhibits the generated model for the performance model from the Pgbench PRESENCE agents. The first row in the table shows the approximation (blue curve line) for multimodel distribution of the throughput metric, thus demonstrating the failure of a single model to capture the system behaviour with respect to the configuration setup. However for the same setup, the latency metric is log normal increasing, and as its p value (0.682) is greater than 0.05, the null hypothesis (i.e., the two samples are the same) is accepted as compared to alternate hypothesis, that is, the two samples are different. Hence, the model for latency, that is, $-0.001 + \text{LOGN}(0.212, 0.202)$, is an accurate representation

Table 6: Modelling DB WS performance metrics from YCSB PRESENCE Agent: Memcached.

Metric	Distribution	Model	Expression	p value (t-test)
Throughput	Beta	$-0.001 + 1 * \text{BETA}(4.41, 2.48)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 4.41$ $\alpha = 2.48$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.106 (>0.05)
Latency read	Beta	$-0.001 + 1 * \text{BETA}(1.64, 3.12)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 1.64$ $\alpha = 3.12$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.832 (>0.05)
Latency update	Normal	$\text{NORM}(0.311, 0.161)$ where $\text{NORM}(\text{mean}\mu, \text{stdDev}\sigma)$ $\mu = 0.311$ $\sigma = 0.161$	$f(x) = (1/\sigma\sqrt{2\pi})e^{-(x-\mu)^2/2\sigma^2}$ for all real x	0.794 (>0.05)

Table 7: Modelling DB WS performance metrics from Memtier PRESENCE Agent: Redis.

Metric	Distribution	Model	Expression	p value (t-test)
Throughput	Erlang	$-0.001 + \text{ERLA}(0.0155, 7)$ where $\text{ERLA}(\beta, k)$ $k = 7$ $\beta = 0.0155$ offset $= -0.001$	$f(x) = \begin{cases} (\beta^{-k}x^{k-1}e^{-(x/\beta)})/(k-1)! & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$	0.767 (>0.05)
Latency	Beta	$-0.001 + 1 * \text{BETA}(0.648, 1.72)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 0.648$ $\alpha = 1.72$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.902 (>0.05)
Transfer rate	Erlang	$-0.001 + \text{ERLA}(0.0155, 7)$ where $\text{ERLA}(\beta, k)$ $k = 7$ $\beta = 0.0155$ offset $= -0.001$	$f(x) = \begin{cases} (\beta^{-k}x^{k-1}e^{-(x/\beta)})/(k-1)! & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$	0.287 (>0.05)

of the WS performance metrics exhibited by the PRESENCE agents in Figure 6. As regards the HTTP WS performance analysis using PRESENCE, we decided to report in Table 10 the model generated from the performance model from the HTTP Load PRESENCE agents. Again, we can see that we fail to model the throughput metric with respect to the configuration setup discussed in the previous section. However, for the same setup, the response time is Beta increasing, and as its p value (0.165) is greater than 0.05, the null hypothesis is accepted. Hence, the model for latency, that is, $-0.001 + 1 * \text{BETA}(1.55, 3.46)$, is an accurate representation of the WS performance metrics exhibited by the PRESENCE agents in Figure 6.

Summary of the obtained Models: In this paper, 19 models were generated which represent the performance metrics for the SaaS Web Service by using the PRESENCE approach. Out of the 19 models, 15 models, that is, 78.9% of the analyzed models are proved to have accurately represent the performance metrics collected by the PRESENCE agents, such as throughput, latency, transfer rate, and response time in different contexts depending on the considered WS. The accuracy of the proposed models is assessed by the reference statistical t-tests, performed against the common significance level of 0.05.

5. Conclusion

Motivated by recent scandals in the automotive sector (which demonstrate the capacity of solution providers to adapt the behaviour of their product when submitted to an evaluation campaign to improve the performance results), this paper presents PRESENCE, an automatic framework

Table 8: Modelling DB WS performance metrics from Memtier PRESENCE Agent: Memcached.

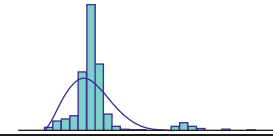
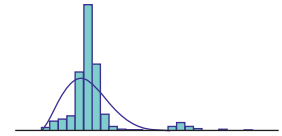
Metric	Distribution	Model	Expression	p value (t-test)
Throughput	—	—		—
Latency read	Beta	$-0.001 + 1 * \text{BETA}(0.99, 2.1)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 0.99$ $\alpha = 2.1$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.625 (>00.05)
Latency update	—	—		—

Table 9: Modelling DB WS performance metrics from Pgbench PRESENCE Agent.

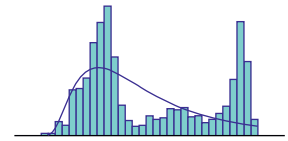
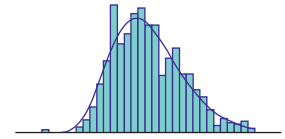
Metric	Distribution	Model	Expression	p value (t-test)
Throughput	—	—		—
Latency	Log normal	$-0.001 + \text{LOGN}(0.212, 0.202)$ where $\text{LOGN}(\log \text{Mean}\mu, \text{LogStd}\sigma)$ $\mu = 0.212$ $\sigma = 0.202$ offset $= -0.001$	$f(x) = \begin{cases} (x^{\beta-1}(1-x)^{\alpha-1})/B(\beta,\alpha) & \text{for } 0 < x < 1 \\ 0 & \text{otherwise,} \end{cases}$ where β is the complete beta function given by $B(\beta,\alpha) = \int_0^1 t^{\beta-1}(1-t)^{\alpha-1}\,dt$	0.682 (>0.05)

Table 10: Modelling HTTP WS performance metrics from HTTP load PRESENCE Agent.

Metric	Distribution	Model	Expression	p value (t-test)
Throughput	—	—		—
Latency	Log normal	$-0.001 + 1 * \text{BETA}(1.55, 3.46)$ where $\text{BETA}(\beta, \alpha)$ $\beta = 1.55$ $\alpha = 3.46$ offset $= -0.001$	$f(x) = \begin{cases} 1/(\sigma x\sqrt{2\pi})e^{-(ln(x)-\mu)^2/2\sigma^2} & \text{for } x > 0 \\ 0 & \text{otherwise} \end{cases}$	0.165 (>0.05)

which aims at evaluating, monitoring, and benchmarking Web Services (WSs) offered across several Cloud Services Providers (CSPs) for all types of Cloud Computing (CC) and Mobile CC platforms. More precisely, PRESENCE aims at evaluating the QoS and SLA compliance of Web Services (WSs) by stealth way, that is, by rendering the performance evaluation as close as possible from a regular yet heavy usage of the considered service. Our framework is relying on a Multi-Agent System (MAS) and a carefully designed client (called the *Auditor*) responsible to interact with the set of CSPs being evaluated.

The first step to ensure the dynamic adaptation of the workload to hide the evaluation process resides in the capacity to model accurately this workload based on the

configuration of the agents responsible for the performance evaluation.

In this paper, a nonexhaustive list of 22 metrics was suggested to reflect all facets of the QoS and SLA compliance of a WSs offered by a given CSP. Then, the data collected from the execution of each agent within the PRESENCE client can be then aggregated within a dedicated module and treated to exhibit a rank and classification of the involved CSPs. From the preliminary modelling of the load pattern and performance metrics of each agent, a stealth module takes care of finding through a GA the best set of parameters for each agent such that the resulting pattern of the PRESENCE client is indistinguishable from a regular usage. While the complete framework is described in the seminal paper for PRESENCE [14], the first experimental results presented in this work focus on the performance and networking metrics between cloud providers and cloud customers.

In this context, 19 generated models were provided, out of which 78.9% accurately represent the WS performance metrics for the two SaaS WSs deployed in the experimental setup. The claimed accuracy is confirmed by the outcome of reference statistical t-tests and the associated p values computed for each model against the common significance level of 0.05.

This opens novel perspectives for assessing the SLA compliance of Cloud providers using the PRESENCE framework. The future work induced by this study includes the modelling and validation of the other modules defined within PRESENCE and based on the monitoring and modelling of the performance metrics proposed in this article. Of course, the ambition remains to test our framework against a real WS while performing further experimentation on a larger set of applications and machines.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

This article is an *extended* version of [14], presented at the 32nd IEEE International Conference of Information Networks (ICOIN), 2018. In particular, Figures 1–6 are reproduced from [14].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors would like to acknowledge the funding of the joint ILNAS-UL Programme on Digital Trust for Smart-ICT. The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [39] (see http://hpc.uni.lu).

References

[1] P. M. Mell and T. Grance, "SP 800–145. The NIST definition of cloud computing," Technical Report, National Institute of Standards & Technology (NIST), Gaithersburg, MD, USA, 2011.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[3] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future Generation Computer Systems Journal*, vol. 56, pp. 684–700, 2016.

[4] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: a survey," *Future generation computer systems Journal*, vol. 29, no. 1, pp. 84–106, 2013.

[5] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, "Mobile cloud computing: a survey, state of art and future directions," *Mobile Networks and Applications Journal*, vol. 19, no. 2, pp. 133–143, 2014.

[6] Y. Xu and S. Mao, "A survey of mobile cloud computing for rich media applications," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 46–53, 2013.

[7] Y. Wang, R. Chen, and D.-C. Wang, "A survey of mobile cloud computing applications: perspectives and challenges," *Wireless Personal Communications Journal*, vol. 80, no. 4, pp. 1607–1623, 2015.

[8] P. R. Palos-Sanchez, F. J. Arenas-Marquez, and M. Aguayo-Camacho, "Cloud computing (SaaS) adoption as a strategic technology: results of an empirical study," *Mobile Information Systems Journal*, vol. 2017, Article ID 2536040, 20 pages, 2017.

[9] M. N. Sadiku, S. M. Musa, and O. D. Momoh, "Cloud computing: opportunities and challenges," *IEEE Potentials*, vol. 33, no. 1, pp. 34–36, 2014.

[10] A. A. Ibrahim, D. Kliazovich, and P. Bouvry, "On service level agreement assurance in cloud computing data centers," in *Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing*, pp. 921–926, San Francisco, CA, USA, June-July 2016.

[11] S. A. Baset, "Cloud SLAs: present and future," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 57–66, 2012.

[12] L. Sun, J. Singh, and O. K. Hussain, "Service level agreement (SLA) assurance for cloud services: a survey from a transactional risk perspective," in *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, pp. 263–266, Bali, Indonesia, December 2012.

[13] C. Di Martino, S. Sarkar, R. Ganesan, Z. T. Kalbarczyk, and R. K. Iyer, "Analysis and diagnosis of SLA violations in a production SaaS cloud," *IEEE Transactions on Reliability*, vol. 66, no. 1, pp. 54–75, 2017.

[14] A. A. Ibrahim, S. Varrette, and P. Bouvry, "PRESENCE: toward a novel approach for performance evaluation of mobile cloud SaaS web services," in *Proceedings of the 32nd IEEE International Conference on Information Networking (ICOIN 2018)*, Chiang Mai, Thailand, January 2018.

[15] V. Stantchev, "Performance evaluation of cloud computing offerings," in *Proceedings of the 3rd International Conference on Advanced Engineering Computing and Applications in Sciences, ADVCOMP 2009*, pp. 187–192, Sliema, Malta, October 2009.

[16] J. Y. Lee, J. W. Lee, D. W. Cheun, and S. D. Kim, "A quality model for evaluating software-as-a-service in cloud computing," in *Proceedings of the 2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications*, pp. 261–266, Haikou, China, 2009.

[17] C. J. Gao, K. Manjula, P. Roopa et al., "A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation," in *Proceedings of the CloudCom 2012-Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, pp. 464–471, Taipei, Taiwan, December 2012.

[18] P. X. Wen and L. Dong, "Quality model for evaluating SaaS service," in *Proceedings of the 4th International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2013*, pp. 83–87, Xi'an, China, September 2013.

[19] G. Cicotti, S. D'Antonio, R. Cristaldi, and A. Sergio, "How to monitor QoS in cloud infrastructures: the QoSMONaaS approach," *Studies in Computational Intelligence*, vol. 446, pp. 253–262, 2013.

[20] G. Cicotti, L. Coppolino, S. D'Antonio, and L. Romano, "How to monitor QoS in cloud infrastructures: the QoSMONaaS approach," *International Journal of Computational Science and Engineering*, vol. 11, no. 1, pp. 29–45, 2015.

[21] A. A. Z. A. Ibrahim, D. Kliazovich, and P. Bouvry, "Service level agreement assurance between cloud services providers and cloud customers," in *Proceedings–2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2016*, pp. 588–591, Cartagena, Colombia, May 2016.

[22] A. M. Hammadi and O. Hussain, "A framework for SLA assurance in cloud computing," in *Proceedings–26th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2012*, pp. 393–398, Fukuoka, Japan, March 2012.

[23] S. S. Wagle, M. Guzek, and P. Bouvry, "Cloud service providers ranking based on service delivery and consumer experience," in *Proceedings of the 2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pp. 209–212, Niagara Falls, ON, Canada, October 2015.

[24] S. S. Wagle, M. Guzek, and P. Bouvry, "Service performance pattern analysis and prediction of commercially available cloud providers," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, pp. 26–34, Hong Kong, China, December 2017.

[25] M. Guzek, S. Varrette, V. Plugaru, J. E. Pecero, and P. Bouvry, "A holistic model of the performance and the energy-efficiency of hypervisors in an HPC environment," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 15, pp. 2569–2590, 2014.

[26] M. Bader-El-Den and R. Poli, "Generating sat local-search heuristics using a gp hyper-heuristic framework," in *Artificial Evolution*, N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, Eds., pp. 37–49, Springer, Berlin, Heidelberg, Germany, 2008.

[27] J. H. Drake, E. Özcan, and E. K. Burke, "A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem," *Evolutionary Computation*, vol. 24, no. 1, pp. 113–141, 2016.

[28] A. Shrestha and A. Mahmood, "Improving genetic algorithm with fine-tuned crossover and scaled architecture," *Journal of Mathematics*, vol. 2016, Article ID 4015845, 10 pages, 2016.

[29] T. T. Allen, *Introduction to ARENA Software*, Springer, London, UK, 2011.

[30] R. C. Blair and J. J. Higgins, "A comparison of the power of Wilcoxon's rank-sum statistic to that of Student's t statistic under various nonnormal distributions," *Journal of Educational Statistics*, vol. 5, no. 4, pp. 309–335, 1980.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing SoCC' 10*, Indianapolis, IN, USA, June 2010.

[32] Redis Labs, *memtier_benchmark: A High-Throughput Benchmarking Tool for Redis & Memcached*, Redis Labs, Mountain View, CA, USA, https://redislabs.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis- memcached/, 2013.

[33] Redis Labs, *How Fast is Redis?*, Redis Labs, Mountain View, CA, USA, https://redis.io/topics/benchmarks, 2018.

[34] Twitter, *rpc-perf–RPC Performance Testing*, Twitter, San Francisco, CA, USA, https://github.com/AbdallahCoptan/rpc-perf, 2018.

[35] Postgresql, "pgbench—run a benchmark test on PostgreSQL," https://www.postgresql.org/docs/9.4/static/pgbench.html, 2018.

[36] A. Labs, "HTTP_LOAD: multiprocessing http test client," https://github.com/AbdallahCoptan/HTTP_LOAD, 2018.

[37] Apache, "ab—Apache HTTP server benchmarking tool," https://httpd.apache.org/docs/2.4/programs/ab.html, 2018.

[38] The Regents of the University of California, "iPerf—the ultimate speed test tool for TCP, UDP and SCTP," https://iperf.fr/, 2018.

[39] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an academic HPC cluster: the UL experience," in *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014)*, pp. 959–967, Bologna, Italy, July 2014.