2015 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems

# BUNGEE: An Elasticity Benchmark for Self-Adaptive IaaS Cloud Environments

Nikolas Roman Herbst and Samuel Kounev
University of Würzburg
Würzburg, Germany
Email: {firstname.lastname}@uni-wuerzburg.de

Andreas Weber and Henning Groenda
FZI Research Center for Information Technology,
Karlsruhe, Germany
Email: {lastname}@fzi.de

***Abstract*—Today's infrastructure clouds provide resource elasticity (i.e. auto-scaling) mechanisms enabling self-adaptive resource provisioning to reflect variations in the load intensity over time. These mechanisms impact on the application performance, however, their effect in specific situations is hard to quantify and compare. To evaluate the quality of elasticity mechanisms provided by different platforms and configurations, respective metrics and benchmarks are required. Existing metrics for elasticity only consider the time required to provision and deprovision resources or the costs impact of adaptations. Existing benchmarks lack the capability to handle open workloads with realistic load intensity profiles and do not explicitly distinguish between the performance exhibited by the provisioned underlying resources, on the one hand, and the quality of the elasticity mechanisms themselves, on the other hand.**

**In this paper, we propose reliable metrics for quantifying the timing aspects and accuracy of elasticity. Based on these metrics, we propose a novel approach for benchmarking the elasticity of Infrastructure-as-a-Service (IaaS) cloud platforms independent of the performance exhibited by the provisioned underlying resources. We show that the proposed metrics provide consistent ranking of elastic platforms on an ordinal scale. Finally, we present an extensive case study of real-world complexity demonstrating that the proposed approach is applicable in realistic scenarios and can cope with different levels of resource efficiency.**

## I. Introduction

Infrastructure-as-a-Service (IaaS) cloud environments provide the benefit of Utility Computing [1]. Accordingly, many providers offer tools that allow customers to configure automated adaptation processes and thus benefit from the increased flexibility and the ability to react on variations in the load intensity. Predicting and managing the performance impact of such adaptation processes and comparing the elasticity of different approaches is still in its infancy. However, customers require that performance related service level objectives (SLOs) for their applications are continuously met.

Elasticity itself is influenced by these adaptation processes as well as by other factors such as the underlying hardware, the virtualization technology, or the cloud management software. These factors vary across providers and often remain unknown to the cloud customer. Even if they are known, the effect of specific configurations on the performance of an application is hard to quantify and compare. Furthermore, the available adaptation processes are quite different in their methods and complexity as shown in the surveys by Lorido-Botran et al. [2], Galante et al. [3] and of Jennings and Stadler [4].

Previous works on elasticity metrics and benchmarks evaluate this quality attribute only indirectly and to a limited extent: The focus of existing metrics lays either on the technical provisioning time [5], [6], [7], on the response time variability [8], [9], [10], or on the impact on business costs [11], [12], [13], [14], [15]. Existing approaches do not account for differences in the efficiency of the underlying physical resources and employ load profiles that are rarely representative of modern real-life workloads with variable load intensities over time. However, the quality of a mechanism in maintaining SLOs depends on the scenario and workload.

In this paper, we propose: (i) a set of intuitively understandable and hardware-independent metrics for characterizing the elasticity of a self-adaptive platform, and (ii) a novel benchmarking methodology, called BUNGEE, for evaluating the elasticity of IaaS cloud platforms using the proposed metrics. These contributions are partially based on our prior work in [16] and in [17] presenting general discussions about definitions and differentiation of terms, as well as initial ideas and conceptual sketches without any evaluation of specific metrics or benchmarking methodology.

The refined metrics we propose here support evaluating both the accuracy and the timing aspects of elastic behavior. We discuss how the metrics can be aggregated and used to compare the elasticity of cloud platforms. The metrics are designed to support human interpretation and to ease decision making by comparing resource supply and demand curves. The proposed elasticity benchmarking approach BUNGEE[1] supports the use of characteristic open workload intensity profiles tailored to the intended application domain. It employs the LIMBO[2] toolkit [18] with its underlying modeling formalism DLIM for describing load profiles [19]. Different levels of platform efficiency are accounted for by performing an automated calibration phase, the results of which are used to adjust the load profile executed on each platform from the considered platforms under test. In combination with the proposed metrics this allows an independent and quantitative evaluation of the actual achieved platform elasticity.

[1]BUNGEE Cloud Elasticity Benchmark: http://descartes.tools/bungee
[2]LIMBO Load Intensity Modeling: http://descartes.tools/limbo

978-0-7695-5567-6/15 $31.00 © 2015 IEEE
DOI 10.1109/SEAMS.2015.23

In our evaluation, we demonstrate that the proposed metrics provide a consistent ranking of elastic platforms and adaptation process configurations on an ordinal scale. The BUNGEE benchmarking approach is evaluated by applying it to an extensive real-world workload scenario considering deployments on both a private cloud (based on CloudStack) and the public Amazon Web Services (AWS) cloud. The evaluation scenario employs a realistic load profile, consisting of several millions of request submissions, and is conducted using different types of virtual machine instances that differ in terms of their exhibited performance. We demonstrate that the proposed approach is applicable in realistic scenarios and can cope with different levels of resource efficiency.

The remainder of this paper is structured as follows: In Section II, we review related work on metrics and benchmarks. Section III proposes a set of metrics for the quantification of elasticity together with a metric aggregation approach. Section IV explains the benchmarking approach BUNGEE in detail. Section V describes the evaluation and Section VI concludes the paper.

## II. Related Work

In this section, we group existing elasticity metrics and benchmarking approaches according to their perspective and discuss shortcomings.

**Elasticity Metrics:**
Several metrics for elasticity have been proposed so far:
(i) The "scaling latency" metrics in [5], [6] or the "provisioning interval" in [7] capture the time to bring up or drop a resource. This duration is a technical property of an elastic environment independent of the demand changes and the elasticity mechanism itself that decides when to trigger a reconfiguration. Thus, these metrics are insufficient to fully characterize the elasticity of a platform.
(ii) The "elastic speedup" metric proposed by SPEC OSG in [7] relates the processing capability of a system at different scaling levels. This metric does not capture the dynamic aspect of elasticity and is regarded as scalability metric.
(iii) The "reaction time" metric in [20] can only be computed if a unique mapping between resource demand changes and supply changes exists. This assumption does not hold especially for proactive elasticity mechanisms or for mechanisms that have unstable (alternating) states.
(iv) The approaches in [8], [21], [10], [9] characterize elasticity indirectly by analysing response times for significant changes or for SLO compliance. In theory, perfect elasticity would result in constant response times for varying arrival rates. In practice, detailed reasoning about the quality of platform adaptations based on response times alone is hampered due to the lack of relevant information about the platform behavior, e.g., the information about the amount of provisioned surplus resources.
(v) Cost-based metrics are proposed in [11], [12], [13], [14], [15] quantifying the impact of elasticity either by comparing the resulting provisioning costs to the costs for a peak-load static resource assignment or the costs of a hypothetical perfect elastic platform. In both cases, the resulting metrics strongly depend on the underlying cost model, as well as on the assumed penalty for under-provisioning, and thus they do not support fair cross-platform comparisons.
(vi) The integral-based "agility" metric proposed by SPEC OSG in [7] compares the demand and supply over time normalized by the average demand. They state that the metric becomes invalid in cases where SLOs are not met. This metric resembles our previously proposed "precision" metric in [16]. In this paper, we propose a refined version normalized by time (see Section III-A) to capture the accuracy aspect of elastic adaptations, considering also situations when SLOs are not met.

**Benchmarks:**
Existing benchmarking approaches for elasticity [12], [13], [14], [22], [11], [9], [10], [15], [21] account neither for differences in the efficiency of the underlying physical resources, nor for the possibly non-linear scalability of the evaluated platform. As a consequence, elasticity evaluation is not performed in isolation from these related platform attributes, as previously highlighted in [16]. In contrast, the approach proposed in this paper uses the results from an initial scalability and efficiency analysis to adapt the load profile generated for evaluating elasticity, in such a way that differences in the platform efficiency and scalability are factored out. Another limitation of existing elasticity benchmarks is that systems are subjected to load intensity variations that are not representative of real-life workload scenarios. For example, in [9] and [22], of scaling the workload downwards is completely omitted. In [11], sinus like load profiles with plateaus are employed. Real-world load profiles exhibit a mixture of seasonal patterns, trends, bursts and noise. We account for the generic benchmark requirement "representativeness" [23] by employing the load profile modeling formalism DLIM previously presented in [19].

## III. Elasticity Metrics

The elasticity metrics we propose in the following are specified based the definition of elasticity given in [16]:

**Elasticity** is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources *match* the current demand as closely as possible.

The resource *demand* induced by a given load intensity is understood as the minimal amount of resources required for fulfilling a given performance related service level objective (SLO). The metrics we propose are designed to characterize two core aspects of elasticity: *accuracy* and *timing*[3]. For all metrics, the optimal value is zero corresponding to a perfectly elastic platform. The following assumptions must hold for applying the proposed elasticity metrics on a set of evaluated

[3]In [16], these aspects are referred to as *precision* and *speed*.

platforms: the existence of an autonomic adaption process, the scaling of the same resource type, e.g., CPU cores or virtual machines (VMs), and the respective resource type is scalable within the same ranges, e.g., from 1 to 20 resource units.

The metrics evaluate the resulting elastic behavior as observed from the outside and are thus designed in a manner independent of distinct descriptions of the underlying hardware, the virtualization technology, the used cloud management software or the employed elasticity strategy and its configuration. As a consequence, the metrics and the measurement methodology are applicable in situations where not all influencing factors are known. All metrics require two discrete curves as input: The demand curve, which defines how the resource demand varies during the measurement period, and the supply curve, which defines how the amount of resources allocated by the platform actually varies.

The following Section III-A describes the metrics for quantifying the *accuracy* aspect whereas Section III-B presents a set of metrics for quantifying the *timing* aspect. In Section III-C, we outline an approach for the aggregation of the proposed metrics allowing to rank multiple elastic cloud environments and configurations consistently.

### A. Accuracy

The under-provisioning accuracy metric $accuracy_U$, is calculated as the sum of areas $\sum U$ where the resource demand exceeds the supply normalized by the duration of the measurement period $T$, as visualized in Figure 1. Analogously, the over-provisioning accuracy metric $accuracy_O$ is based on the sum of areas $(\sum O)$ where the resource supply exceeds the demand.

$$\text{Under-provisioning: } accuracy_U\ [resource\ units] = \frac{\sum U}{T}$$
$$\text{Over-provisioning: } accuracy_O\ [resource\ units] = \frac{\sum O}{T}$$

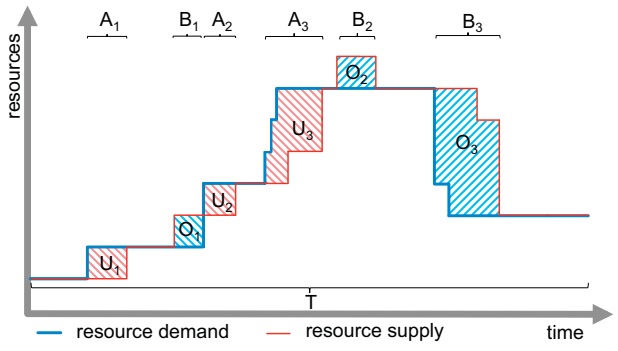



Fig. 1: Illustrating example for accuracy (U, O) and timing (A, B) metrics.

Thus, $accuracy_U$ and $accuracy_O$ measure the average amount of resources that are under/over-provisioned during the measurement period $T$. Since under-provisioning results in violating SLOs, a customer might want to use a platform that does not tend to under-provision at all. Thus, the challenge for providers is to ensure that enough resources are provided at any point in time, but at the same time distinguish themselves from competitors by minimizing the amount of over-provisioned resources. Considering this, the defined separate accuracy metrics for over-provisioning and under-provisioning allow providers to better communicate their elasticity capabilities and customers to select the provider that best matches their needs.

### B. Timing

We characterize the *timing* aspect of elasticity from the viewpoint of the pure *provisioning timeshare*, on the one hand, and from the viewpoint of the induced *jitter* accounting for superfluous or missed adaptations, on the other hand.

*1) Provisioning Timeshare:* The two accuracy metrics allow no reasoning as to whether the average amount of under-/over-provisioned resources results from a few big deviations between demand and supply or if it is rather caused by a constant small deviation. To address this, the following two metrics are designed to provide insights about the ratio of time in which under- or over-provisioning occurs.

As visualized in Figure 1, the following metrics $timeshare_U$ and $timeshare_O$ are computed by summing up the total amount of time spent in an under- $(\sum A)$ or over-provisioned $(\sum B)$ state normalized by the duration of the measurement period. Thus, they measure the overall timeshare spent in under- or over-provisioned states:

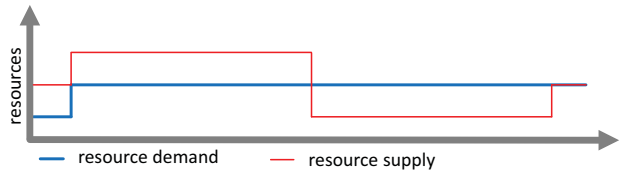
$$\text{Under-provisioning: } timeshare_U = \frac{\sum A}{T}$$
$$\text{Over-provisioning: } timeshare_O = \frac{\sum B}{T}$$

*2) Jitter:* Although the $accuracy$ and $timeshare$ metrics characterize important aspects of elasticity, platforms can still behave very differently while producing the same metric values for $accuracy$ and $timeshare$. An example is shown in Figure 2.

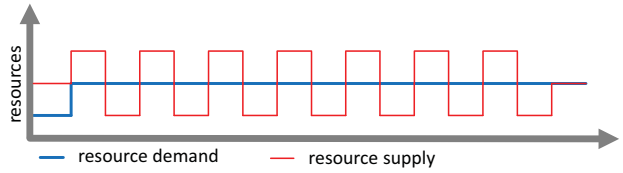
Both Platforms A and B exhibit the same accuracy metrics and spend the same amount of time in under-provisioned and over-provisioned states respectively. However, the behavior of the two platforms differs significantly. Platform B triggers more unnecessary resource supply adaptations than the mechanism on Platform A.

The *jitter* metric addresses this instability and inertia of elasticity mechanisms. A low stability increases adaptation overheads and costs (e.g., in case of instance-hour-based pricing), whereas a high level of inertia results in a decreased SLO compliance.

The $jitter$ metric compares the amount of adaptations in the supply curve $E_S$ with the number of adaptations in the demand curve $E_D$. If a platform de-/allocates more than one resource unit at a time, the adaptations are counted individually




(a) Platform A




(b) Platform B

Fig. 2: Platforms with different elastic behaviors that produce equal results for $accuracy$ and $timeshare$ metrics

per resource unit. The difference is normalized by the length of the measurement period $T$:

$$\text{Jitter metric: } jitter \left[\frac{\#adaptations}{time}\right] = \frac{E_S - E_D}{T}$$

A negative $jitter$ metric indicates that the platform adapts rather sluggish to changes in the demand. A positive $jitter$ metric means that the platform tends to oscillate like Platforms A (little) and B (heavily) as in Figure 2. High absolute values of $jitter$ metrics in general indicate that the platform is not able to react on demand changes appropriately. In contrast to the accuracy and timeshare metrics, a $jitter$ value of zero is a necessary, but not sufficient requirement for a perfect elastic system.

### C. Metric Aggregation

The sections above introduced different metrics for capturing core aspects of elasticity:

- $accuracy_U$ and $accuracy_O$ measure average resource amount deviations
- $timeshare_U$ and $timeshare_O$ measure ratios of time in under- or over-provisioned states
- $jitter$ measures the difference between the frequencies of changes in the demand and supply respectively.

To facilitate direct comparisons between platforms, we propose a way to aggregate the metrics and to build a consistent and fair ranking by selecting an arbitrary baseline platform for normalization. The aggregation and ranking of results is similarly done in established industry benchmarks, e.g., SPEC CPU 2006[4]. Our proposed approach to compute an aggregated *elastic speedup* metric consists of the following three steps:

1) Aggregate the accuracy and timeshare submetrics into a weighted accuracy and a weighted timeshare metric, respectively.
2) Compute elasticity speedups for both of the aggregated metrics using the metric values of a baseline platform.
3) Use the geometric mean to aggregate the speedups for $accuracy$ and $timeshare$ to an $elastic\ speedup$ metric.

The resulting $elastic\ speedup$ metric can be used to compare platforms without having to compare each elasticity metric separately. As a limitation of this approach, the $jitter$ metric is not considered (as it can become zero also in realistic cases of imperfect elasticity). Each of the three steps is explained in the following.

1) The $acc_U$ and $acc_O$ metrics are combined to a weighted accuracy metric $acc_{weighted}$:

$$acc_{weighted} = w_{acc_U} \cdot acc_U + w_{acc_O} \cdot acc_O$$
$$\text{with } w_{acc_U},\ w_{acc_O} \in [0,1], \quad w_{acc_U} + w_{acc_O} = 1$$

In the same way, the $timeshare_U$ and $timeshare_O$ metrics (shortened to $ts$) are combined to a weighted timeshare metric $ts_{weighted}$:

$$ts_{weighted} = w_{ts_U} \cdot ts_U + w_{ts_O} \cdot ts_O$$
$$\text{with } w_{ts_U},\ w_{ts_O} \in [0,1], \quad w_{ts_U} + w_{ts_O} = 1$$

2) Let $x$ be a vector that stores the metric results:

$$x = (x_1, x_2) = (acc_{weighted}, ts_{weighted})$$

For a metric vector $x_{base}$ of a given baseline platform and a metric vector $x_k$ of a benchmarked platform $k$, the speedup vector $s_k$ is computed as follows:

$$s_k = (s_{k_{acc}}, s_{k_{ts}}) = \left(\frac{x_{base_1}}{x_{k_1}}, \frac{x_{base_2}}{x_{k_2}}\right)$$

3) The elements of $s_k$ are aggregated to an $elastic\ speedup_k$ metric using the geometric mean:

$$elastic\ speedup_k = \sqrt{s_{k_{acc}} \cdot s_{k_{ts}}}$$

The geometric mean produces consistent rankings independent of the reference data based on which measurements are normalized; it is generally considered as the only correct mean for normalized measurements [24]. Thus, the ranking of the platforms according to the introduced $elastic\ speedup_k$ is consistent, regardless of the chosen baseline platform. Furthermore, different preferences concerning the two elasticity aspects can be taken into account by using a weighted geometric mean to compute the $elastic\ speedup$ metric:

$$elastic\ speedup_{weighted_k} = {s_{k_{acc}}}^{w_{acc}} \cdot {s_{k_{ts}}}^{w_{ts}}$$
$$\text{with } w_{acc},\ w_{ts} \in [0,1], \quad w_{acc} + w_{ts} = 1$$

**Elasticity Metric Weights:**
The single number elasticity metric shown in the equation above can be adjusted according to the preferences of the target audience by using different weights. For example, the accuracy weights $w_{acc_U} = 0.2,\ \ w_{acc_O} = 0.8$ allow to amplify the influence of the amount of over-provisioned resources compared to the amount of under-provisioned resources. A

[4]SPEC CPU 2006: http://www.spec.org/cpu2006/

reason for doing so could be that over-provisioning leads to additional costs, which mainly depend on the amount of over-provisioned resources. The cost for under-provisioning, in contrast, does not depend that much on the amount of under-provisioned resources, but more on how long resides in an under-provisioned state. This observation can be taken into account by using timeshare weights like: $w_{ts_U} = 0.8,\ w_{ts_O} = 0.2$. Finally, when combining the *accuracy* and *timeshare* metrics, the weights $w_{acc},\ w_{ts}$ can help to prioritize different elasticity aspects. Here, weights like $w_{acc} = \frac{1}{3},\quad w_{ts} = \frac{2}{3}$ for example would double the importance of *short* under- and over-provisioning periods compared to the importance of *small* under- or over-provisioning amounts.

## IV. BUNGEE Elasticity Benchmark

This section presents our the benchmarking approach that is based on the initial sketch in [16] and addresses generic and cloud specific benchmark requirements as stated in Huppler [23], [25] and Folkerts et. al [12]. A general overview of the benchmark components and of the benchmarking workflow is given in this section. The conceptual ideas about the essential benchmark components are discussed in individual subsections. We provide an implementation of the benchmark concept named BUNGEE[5]. Detailed information about its benchmark implementation is provided in [26].

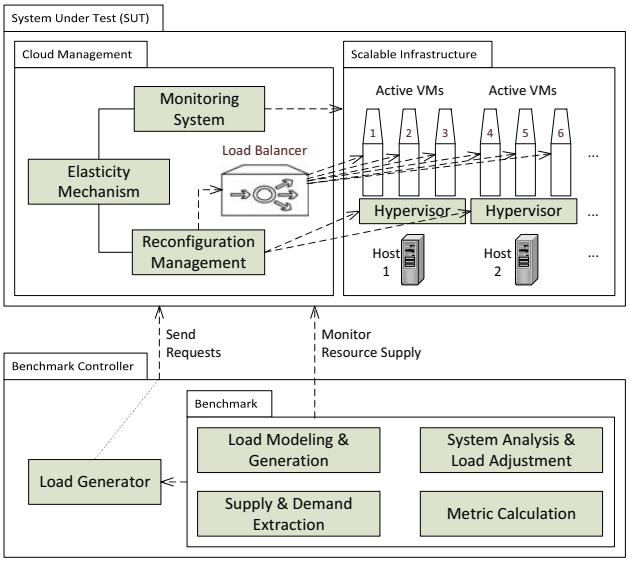



Fig. 3: Blueprint for the SUT and the BUNGEE benchmark controller

Figure 3 illustrates an elastic cloud platform architecture as a blueprint for a infrastructure cloud "system under test" (SUT) together with the *benchmark controller*, which runs the benchmark. The individual benchmark components automate the process for benchmarking resource elasticity in four sequential steps:

[5]BUNGEE Cloud Elasticity Benchmark: http://descartes.tools/bungee

1) **Platform Analysis:** The benchmark analyzes the SUT with respect to the efficiency of its underlying resources and its scaling behavior.
2) **Benchmark Calibration:** The results of the analysis are used to adjust the load intensity profile injected on the SUT in a way that it induces the same resource demand on all compared platforms.
3) **Measurement:** The load generator exposes the SUT to a varying workload according to the adjusted load profile. The benchmark extracts the actual induced resource demand and monitors resource supply changes on the SUT.
4) **Elasticity Evaluation:** The elasticity metrics are computed and used to compare the resource demand and resource supply curves with respect to different elasticity aspects.

The remainder of this section explains the benchmark components according to the following structure: Section IV-A explains how workloads are modeled and executed. Section IV-B explains why analyzing the evaluated platform and calibrating the benchmark accordingly is required and how it is realized. Finally, Section IV-C explains how the resource demand curve and the resource supply curve can be extracted during the measurement.

### A. Load Modeling and Generation

This section covers modeling and executing workloads suitable for elasticity benchmarking.

**Load Profile:** A benchmark should stress the SUT in a representative way. Classical performance benchmarks achieve this by executing a representative mix of different programs. An elasticity benchmark measures how a platform reacts when the demand for specific resources changes. Thus, an elasticity benchmark is required to induce a representative profile of demand changes. Changes in demand and accordingly elastic adaptation of the platform are mainly caused by a varying load intensity. Hence, for elasticity benchmarking, it is important that the variability of the load intensity is directly controlled by generating precisely timed requests based on an open workload profile. Workloads are commonly modeled either as closed workloads or as open workloads [27]. In closed workloads new job arrivals are triggered by job completions, whereas arrivals in open workloads are independent of job completions.

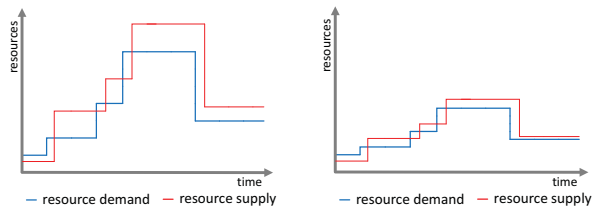
Real-world load profiles are typically composed of trends, seasonal patterns, bursts and noise elements. To address these requirements, BUNGEE employs realistic load intensity profiles to stress the SUT in a representative manner.

V. Kistowski et al. present in [19] a formalism providing means to define load intensity profiles in a flexible manner. The corresponding LIMBO toolkit, described in [18], facilitates the creation of new load profiles that are either automatically extracted from existing load traces or modelled from scratch with desired properties like seasonal patterns or bursts. The usage of this toolkit and the underlying formalism within BUNGEE allows the creation of realistic load profiles that remain configurable and compact.

**Load Generation:** In order to stress an elastic platform in a reproducible manner, it is necessary to inject accurately timed requests into the SUT that mainly stress the resources that are elastically scaled. Depending on the request response time, the handling of consecutive requests (sending of a request and waiting for the corresponding response) overlaps and therefore requires a high level of concurrency. In its load generator, BUNGEE employs the thread pool pattern (also known as replicated worker pattern [28]) with dynamic assignments of requests to threads. When the maximum response time is known - for example when a timeout is defined for the requests - the amount of threads required to handle all requests without delays due to a lack of parallelism can be computed as shown in [26]. These threads can either be located on one node or be split up across several nodes. After each benchmark execution, BUNGEE evaluates the request submission accuracy by checking whether the $95^{th}$ percentile of the deviation between the defined load profile and the actual request submission times remained below a certain threshold.

### B. Analysis and Calibration

The resource demand of a platform for a fixed load intensity depends on two factors: The efficiency of a single underlying resource unit and the overhead caused by combining multiple resource units. Both aspects can vary from platform to platform and define distinct properties namely efficiency and scalability. Elasticity is a different property and should be measured separately. We achieve this by analyzing the load processing capabilities of a platform before evaluating its elasticity. After such an analysis, it is known how many resource units the platform needs for every static load intensity level. The resource demand can then be expressed as a function of the load intensity: $demand(intensity)$. With the help of this mapping function, which is specific for every platform, it is possible to compare the resource demand to the amount of the actually allocated resources. In the following section, we explain how the mapping function can be derived.




(a) Platform A - less efficient resources (b) Platform B - more efficient resources

Fig. 4: Different resource demand and supply curves for the same load profile

Figure 4 illustrates the effect of the same workload on two exemplary platforms. Since the resources of Platform B are more efficient than those of Platform A, Platform B can handle the load with less resources than Platform A. For both platforms, there exist some points in time where the platforms over-provision and other points in time where they under-provision. As the resource demands of the platforms are different, a direct comparison of their elasticity is performed using cost models. (see Section II).

In contrast to related approaches, the idea of our approach introduces an additional benchmark calibration step that adjusts the load profile injected on each tested platform in a way that the induced resource demand changes are identical on all platforms. By doing so, the possibly different levels of efficiency of the underlying resources as well as different scaling behaviors are compensated. With an identical injected resource demands it is now possible to directly compare the quality of the adaptation process and thus evaluate elasticity in a fair way. The realisation consists of two steps, explained in the following two subsections.

*1) Platform Analysis:* The goal of the *Platform Analysis* is to derive a function that maps a given load intensity to the corresponding resource demand. Hereby, the resource demand is the minimum resource amount that is necessary to handle the load intensity without violating a set of given SLOs. Therefore, the analysis assumes a predefined set of SLOs. They have to be chosen according to the targeted domain.

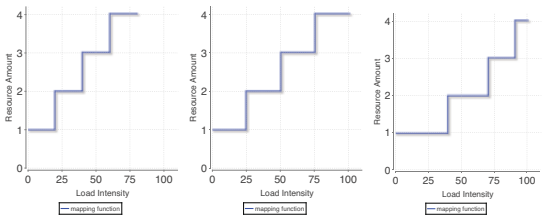
Since this *Platform Analysis* is not intended to evaluate any elastic behavior, scaling is controlled manually by the framework during the analysis phase. The results of three hypothetical analyses are shown in Figure 5. The upper end of the line marks the upper scaling bound and thus the maximal load intensity the platforms can sustain without violating SLOs. This upper bound is either caused by a limited amount of available resources or by limited scalability due to other reasons like limited bandwidth or increased overhead. In the latter case, additional resources are available, but even after adding resources the SLOs cannot be satisfied. Figure 5 shows that the derived mapping function $demand(intensity)$ is a step function. The function is characterized by the intensity levels at which the resource demand increases. Thus, analyzing the platform means finding these levels.

The analysis starts by configuring the SUT to use one resource instance. This includes configuring the load balancer to forward all requests to this instance. Now, the maximum load intensity that the platform can sustain without violating the SLOs is determined as the level at which the resource demand increases. In this paper, the maximum load intensity is also referred to as the load processing capability. Several load picking algorithms that are able to find the load processing capability are discussed in [29]. We apply a binary search algorithm to realize a *searchMaxIntensity*-function. Binary search consumes more time than a model guided search but, in contrast to the latter, it is guaranteed to converge and it is still effective compared to a simple linear search. Since the upper and lower search bounds are not known at the beginning, the binary search is preceded by an exponential increase/decrease of the load intensity to find those bounds. As soon as both bounds are known, a regular binary search is applied. Once the load processing capability for one resource is known, the platform is reconfigured to use an additional resource unit

and the load balancer is reconfigured accordingly. After the reconfiguration, the platform should be able to comply with the SLOs again. The load processing capability is again searched with a binary search algorithm. This process is repeated until either there are no additional resources that can be added to the platform, or a state is reached such that adding further resources does not increase the load processing capability. In both cases, the upper scaling bound is reached and the analysis is finished.

*2) Benchmark Calibration:* The goal of the *Benchmark Calibration* activity is to induce the same demand changes at the same points in time on every platform. To achieve this, BUNGEE adapts the load intensity curve for every platform to compensate for different levels of efficiency of the underlying resources and different scaling behaviors. The mapping function $demand(intensity)$ derived in the previous *Platform Analysis* step, contains information about both. Figure 5 illustrates what impact different levels of efficiency and different scaling behaviors may have on the mapping function. Compared to Platform C, the underlying resources of Platform D are more efficient. Hence, the steps of the mapping function are longer for Platform D than for Platform C. For both platforms, there is no overhead when adding further resources. The resource demand is increasing linearly with the load intensity and the length of the steps for one platform is therefore independent of the load intensity. For Platform E, in contrast, the overhead increases when additional resources are used. As a result of this non-linear increasing resource demand, the length of the steps of the mapping function decreases for increasing load intensity.

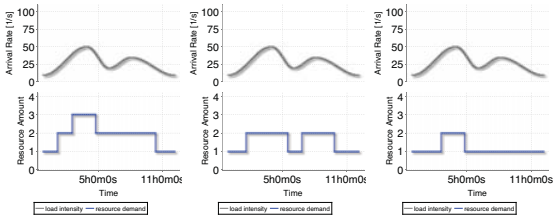
As illustrated in Figure 6, the resource demand variations on Platforms C-E are different when they are exposed to the same load profile, although they offer the same amount of resources. Different scaling behaviors and different levels of efficiency of the underlying resources cause this difference. As a basis for the transformation of load profiles, one platform is selected as baseline to serves as reference for demand changes. The induced resource demands on the compared platforms are the same as for the baseline platform since the respective transformed load profiles are used.




(a) Platform C (b) Platform D (c) Platform E

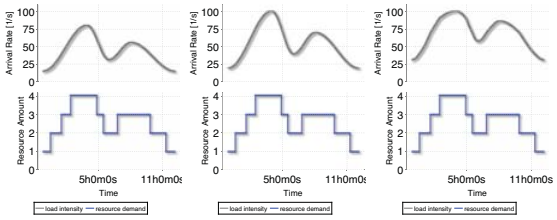
Fig. 5: Different mapping functions

The resource demand of the baseline platform is assumed to increase linearly with load intensity. Thus, the steps of the mapping function are equal in length. Using this assumption,




(a) Platform C (b) Platform D (c) Platform E

Fig. 6: Resource demand for the same load profile




(a) Platform C (b) Platform D (c) Platform E

Fig. 7: Resource demands for platform specific adjusted load profiles

the mapping function $demand_{base}(intensity)$ of the baseline platform can be characterized by two parameters: The number of steps $n_{base}$, which is equal to the assumed amount of available resources, and the maximum load intensity $maxInt_{base}$ that the base platform can sustain using all resources. The first parameter is chosen as the maximum amount of resources that all platforms support. The second parameter should be greater than or equal to the maximum load intensity that occurs in the load profile. Having defined the mapping function $demand_{base}(intensity)$, for every benchmarked platform $k$, the load profile adjustment step finds an adjustment function $adjInt_k(intensity)$ such that the following applies:

$$\forall intensity \in [0, maxInt_{base}]:$$
$$demand_{base}(intensity) = demand_k(adjInt_k(intensity))$$

The adjustment function $adjInt_k(intensity)$ maps the steps from the $demand_{base}(intensity)$ function to steps of the $demand_k(intensity)$ function. The result is a piecewise linear function, whereby every linear section represents the mapping of one step in the $demand_{base}(intensity)$ to the same step in the $demand_k(intensity)$ function. The parameters $m_i$ and $b_i$ that define the linear function $y_{k_i}(x) = m_i * x + b_i$ mapping the intensities belonging to the $i$th step of the $demand_{base}(intensity)$ function to the $i$th step of the $demand_k(intensity)$ function can be calculated as follows:

$$startInt_{k_i} = \max\{intensity | demand_k(intensity) < i\}$$
$$endInt_{k_i} = \max\{intensity | demand_k(intensity) = i\}$$
$$stepLength_{k_i} = endInt_{k_i} - startInt_{k_i}$$
$$m_{k_i} = stepLength_{k_i} / (maxInt_{base} / n_{base})$$
$$b_{k_i} = startInt_{k_i} - stepLength_{k_i} * (i - 1)$$

and thus the function $adjInt_k(intensity)$ can be expressed as:

$$adjInt_k(x) = \begin{cases} y_{k_1} & \text{when } startInt_{k_1} < x \leq endInt_{k_1} \\ \vdots & \\ y_{k_n} & \text{when } startInt_{k_n} < x \leq endInt_{k_n} \end{cases} \quad (1)$$

Having calculated the adjustment function $adjInt_k(intensity)$, this function can be applied to the original load profile $intensity(t) = lp(t)$ in order to retrieve a platform specific adjusted load profile $lp_k(t)$. This adjusted load profile can then be used in the actual benchmark run.

$$lp_k(t) = adjInt_k(lp(t))$$

Figure 7 shows the induced load demand for Platforms C, D and E using the adjusted load profiles. Although the platforms have underlying resources with different levels of efficiency and different scaling behaviors, the induced resource demand variations are now equal for all compared platforms.

### C. Measurement and Metric Calculation

After having conducted the *Platform Analysis* and *Calibration* steps, BUNGEE executes the measurements with a reasonable warm-up at the beginning. Afterwards, the measurement run is validated using the sanity check explained at the end of Section IV-A. For the computation of the accuracy, provisioning timeshare and jitter metrics, the resource demand and supply changes are required as input: The demand changes can be directly derived from the adapted load profile using the platform specific mapping function. The supply changes need to be extracted either from cloud management logs or by polling during the measurement. In both cases, a resource is defined to be supplied during the time spans when it is ready to process or actively processing requests. The time spans a resource is billed may differ significantly.

## V. Evaluation

This section presents the experiment setup and demonstrates the benchmarking capabilities of the BUNGEE approach with a realistic load profile and a realistic amount of dynamically scaled resources. The benchmark is applied to a private, CloudStack-based cloud, as well as a public, AWS-based cloud (Amazon EC2).

### A. Experiment Setup

As previously illustrated in Figure 3, the experiment setup consists of three parts: The infrastructure nodes, the management and load balancing nodes and the benchmark controller nodes. The first two parts form the benchmarked SUT. In the private cloud setup, the infrastructure provides the physical resources of four 12 core AMD Opteron 6174 CPUs at 2.2 GHz and 256 GB RAM as fully virtualized resources using XenServer 6.2 as hypervisor. The management node (Intel Core i7 860 with 8 cores at 2.8 GHz and 8 GB RAM) runs the cloud management software (CloudStack 4.2) and the load balancer (Citrix Netscaler 10.1 VPX 1000) in separate Linux-based VMs. The benchmark controller node runs the Java-based benchmark harness and the load driver (JMeter) on a Windows Vista desktop (Dell Precision T3400 (4 x 2.5 GHz) and 8 GB RAM). The three nodes are physically connected over a 1 GBit Ethernet network. Clock synchronization is ensured by using a Stratum 3 NTP server located in the same network. The template for virtual machines used by CloudStack bases on CentOS 5.6 as operating system with java run-time environment and SNMP service installed. SNMP provides access to resource utilization information required for the elasticity mechanism. Experiments on the private CloudStack-based environment employ VMs with 1 core at 2.2. GHz, 1 GB RAM and local storage is available. Experiments on the AWS EC2 public cloud employ the general purpose *m1.small* instance type.

TABLE I: Elasticity default parameters of private cloud setup

| Name | Default Val. (CS — AWS) | Description |
|---|---|---|
| evalInterval | 5s | Frequency for scaling rule evaluation |
| quietTime | 300s | Period without scaling rule evaluation after supply change |
| destroyVmGracePer | 30s | Time for connection closing before a VM is destroyed |
| condTrueDurUp | 30s — 60s | Duration of condition true before a scale up triggered |
| counterUp | CPU util. | Monitored metric for *thresholdUp* |
| operatorUp | $>$ | Operator comparing *counterUp* and *thresholdUp* |
| thresholdUp | 90% | Threshold for *counterUp* |
| condTrueDurDown | 30s — 60s | Duration of condition true before scale down triggered |
| counterDown | CPU util. | Monitored metric for *thresholdDown* |
| operatorDown | $<$ | Operator comparing *counterDown* and *thresholdDown* |
| thresholdDown | 50% | Threshold for *counterDown* |
| responseTimeout | 1s — 2s | Period of time within that a response is expected from healthy instances |
| healthCheckInterval | 5s | Time between two health checks |
| healthyThreshold | 1 — 2 | Number of subsequent health checks before instance is declared healthy |
| unhealthyThreshold | 4 — 2 | Number of subsequent health checks before instance is declared unhealthy |

**Elasticity Parameters:** CloudStack (CS) and AWS allow to configure a rule based elasticity mechanism with the set of parameters in Table I. The listed default values are used if not mentioned otherwise.

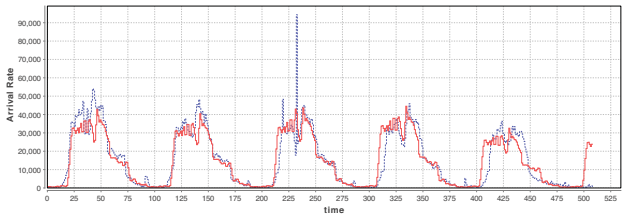
**Benchmark Controller Configuration:** The benchmark controller offers several configuration options that allow to configure it according to the targeted domain. Table II shows the different parameters and default values. The amount of work executed within each request is defined by a $size$ parameter. It is set to $50000$ for the evaluation meaning that each request issues a randomized calculation of the 50000th element of the fibonacci series. During the calibration phase, the benchmark needs a specified SLO in order to perform the *Platform Analysis*. Additionally, the benchmark has a $requestTimeout$ parameter defining how long the benchmark waits for a response before the connection is closed.

TABLE II: Benchmark harness parameters

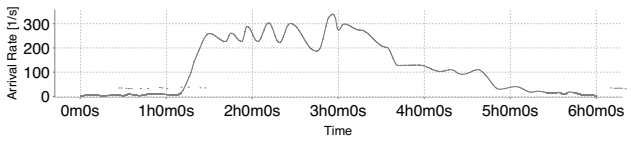
| Name | Default |
|---|---|
| $requestSize$ | 50000 |
| $requestTimeout$ | 1000ms |
| $SLO$ | 95% of all requests must be processed successfully within a maximum response time of 500ms. |
| $warmup_{calibration}$ | 180s |
| $warmup_{measurement}$ | 300s |

### B. Benchmark Evaluation

This section demonstrates the benchmarking capabilities for a realistic load profile and a representative amount of resources on the private and on a public cloud as described in Section V-A. The load profile (see Figure 8) used for this




(a) Original (blue) and modeled (red) intensity [requests/15min] trace




(b) Compacted load profile of a single day adapted for one-core VMs in private cloud setup

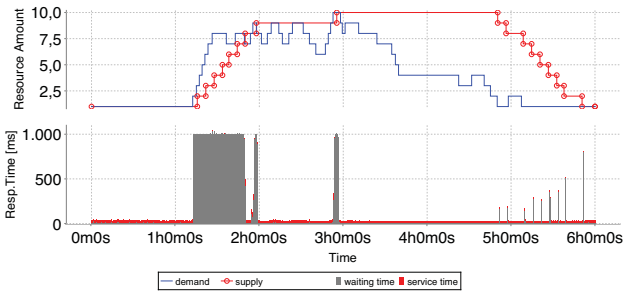
Fig. 8: One day load profile derived from a real five day transaction trace

scenario is derived from a real intensity trace (see Figure 8(a)) previously used in [30]. The trace features the amount of transactions on an IBM z196 Mainframe during February 2011 with a quarter-hourly resolution. To reduce the experiment time, the first day is selected as the load profile and has been compacted from 24 hours to 6 hours. The load intensity within this load profile varies between two and 339 requests per second and contains about 2.7 million request timestamps, when adapted for a maximum of 10 one-core VMs in the private cloud setup. As a single AWS $m1.small$ instance is capable of processing 71 requests per second, instead of 34 for the private cloud VMs, the load profile for public cloud experiments (see top of Figure 10(a)) is adapted to vary between five and 710 requests per second. This timestamp file contains about 5.6 million individual request submission timestamps and induces the same resource demand changes as the load profile for the private cloud setup. The *Platform Analysis* step has been evaluated for its reproducibility separately: After eight repetitions, the maximal deviation from the average processing capability per scaling stage was 6.8% for the public cloud and even lower for the private cloud setup.

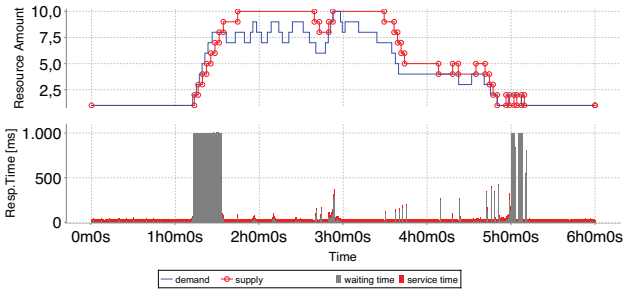
**Platform Configurations:** The resource elasticity of the cloud platforms is evaluated for different elasticity rule parameter settings as shown in Table III: Configuration A serves as a baseline configuration for elasticity comparisons, as it is expected to exhibit the lowest degree of elasticity. Which one of the Configurations B, C and D shows the highest degree of elasticity is not directly visible from the parameters.

TABLE III: Elasticity parameter configurations on private (A,B) and public (C,D) clouds

| Config-uration | quiet-Time | condTrue-DurUp | condTrue-DurDown | thresh-oldUp | thresh-oldDown |
|---|---|---|---|---|---|
| A | 240s | 120s | 120s | 90% | 10% |
| B | 120s | 30s | 30s | 65% | 50% |
| C | 120s | 60s | 60s | 65% | 50% |
| D | 60s | 60s | 60s | 65% | 40% |



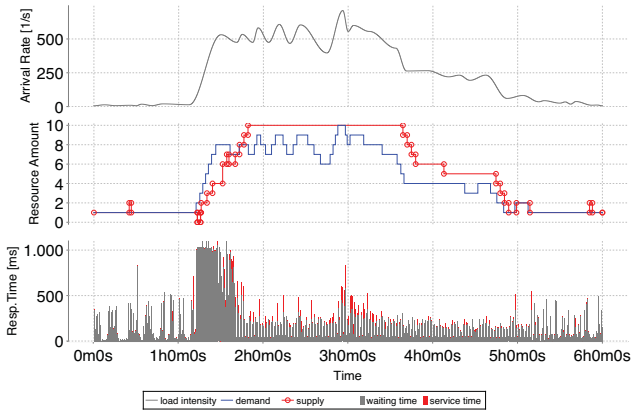

(a) Configuration A: Slow resource increase, very slow resource decrease



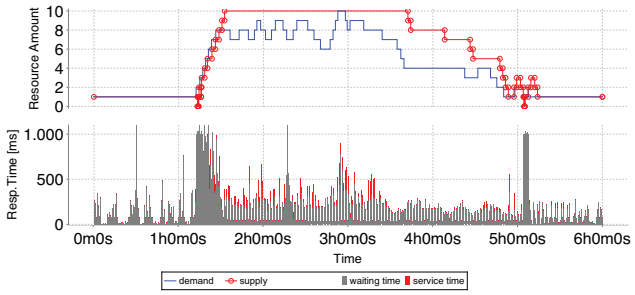

(b) Configuration B: Faster resource increase and decrease than Config. A

Fig. 9: Elastic behavior for two elasticity rule parameter settings on CloudStack

**Results:** Figure 9 and Figure 10 illustrate the exhibited elasticity of the four different Configurations A, B on the private cloud and C, D on the public cloud. Configuration D reacts fastest on the step increase starting at 90 min. The shape of resource allocations of Configuration B fits best to the demand changes. At the step parts of the profile, when the platforms are in under-provisioned state, the request response time rises to the defined timeout of 1 second with the request violating the specified SLO. The response time graphs show




(a) Configuration C on public cloud with adapted load profile




(b) Configuration D: Even faster resource in- and decrease compared to Configurations C

Fig. 10: Elastic behavior for different elasticity rule parameter settings on a public cloud (1)

TABLE IV: Metric results for evaluated configurations on private (A, B) and on public (C, D) clouds

| Configuration | $acc_O$ [#res.] | $acc_U$ [#res.] | $ts_O$ [%] | $ts_U$ [%] | $jitter$ $\left[\frac{\#adap.}{min}\right]$ |
|---|---|---|---|---|---|
| A | 2.425 | 0.264 | 60.1 | 11.7 | -0.067 |
| B | 0.815 | 0.080 | 48.7 | 6.5 | -0.028 |
| C | 1.053 | 0.180 | 51.9 | 8.1 | -0.033 |
| D | 1.442 | 0.049 | 57.6 | 4.7 | -0.017 |

TABLE V: *Elastic speedup* with equal weights: $w_{acc_U} = w_{acc_O} = w_{ts_U} = w_{ts_O} = w_{acc} = w_{ts} = 0.5$

| Configuration | *accuracy speedup* | *timeshare speedup* | **elastic speedup** | SLO viol. [%] |
|---|---|---|---|---|
| B | 3.004 | 1.301 | 1.977 | 8.4 |
| C | 2.181 | 1.197 | 1.615 | 9.1 |
| D | 1.803 | 1.152 | 1.442 | 5.0 |
| A | 1.000 | 1.000 | 1.000 | 20.3 |

higher and more variable response times for the public cloud experiments (C, D) compared to the private cloud experiments (A, B). Possible reasons are a higher performance variability due to imperfect isolation or overbooking on the public cloud.

Table IV contains the metric results for the four configurations. Configuration B exhibits the lowest $accuracy_O$ metric value, whereas Configuration D achieves the lowest $accuracy_U$ result and is less than 5% of the experiment time in an underprovisioned state. All configurations exhibit a negative $jitter$ value as there are more demand changes than supply changes.

The aggregated metric values (see Section III-C) for equal weights are shown in Table V. As expected from the visual representation, Configuration B achieves the highest elastic speedup result, with C, D, and A following Note that the percentage of SLO violating requests is not correlated with the elastic speedup metric, as the former depends on the $timeshare_U$ value and the amount of requests sent during under-provisioned states.

**Summary:** The BUNGEE elasticity benchmark approach was applied to a complex realistic load scenario and ranked different platforms and configurations according to the exhibited degree of elasticity. Both the *Platform Analysis* and the overall benchmarking methodology delivered reproducible and meaningful results.

## VI. Conclusions

This paper presented a set of metrics for capturing the accuracy and timing aspects of elastic platforms. It provided a metric aggregation method and showed how it can be adapted to personal preferences. It described the novel elasticity benchmarking approach BUNGEE and showed its evaluation in a scenario of realistic complexity. It was shown that the proposed metrics and benchmarking methodology are able to rank elastic platforms and configurations on an ordinal scale according to their exhibited degree of elasticity for a given domain-specific and realistic load profile.

As part of future work, we plan to support other resource types besides virtual machines (scale out), e.g., CPU cores (scale up) or main memory. Furthermore, we plan to quantify the impact of proactive/predictive elasticity mechanisms on the elasticity metric values.

## VII. Acknowledgments

The European Union's Seventh Framework Programme funded parts of this research under grant agreement 610711. This research has been supported by the Research Group[6] of the Standard Performance Evaluation Corporation (SPEC)[7]. Amazon Web Services supported this research by a Research Grant award.

## References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[6]SPEC Research: `http://research.spec.org`
[7]SPEC: `http://www.spec.org`

[2] T. Lorido-Botran, J. Miguel-Alonso, and J. 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. [Online]. Available: http://dx.doi.org/10.1007/s10723-014-9314-7

[3] G. Galante and L. C. E. d. Bona, "A Survey on Cloud Computing Elasticity," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 263–270. [Online]. Available: http://dx.doi.org/10.1109/UCC.2012.30

[4] B. Jennings and R. Stadler, "Resource Management in Clouds: Survey and Research Challenges," *Journal of Network and Systems Management*, pp. 1–53, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10922-014-9307-7

[5] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879143

[6] Z. Li, L. O'Brien, H. Zhang, and R. Cai, "On a Catalogue of Metrics for Evaluating Commercial Cloud Services," in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, Sept 2012, pp. 164–173. [Online]. Available: http://dx.doi.org/10.1109/Grid.2012.15

[7] D. Chandler, N. Coskun, S. Baset, E. Nahum, S. R. M. Khandker, T. Daly, N. W. I. Paul, L. Barton, M. Wagner, R. Hariharan, and Y. seng Chao, "Report on Cloud Computing to the OSG Steering Committee," Tech. Rep., Apr. 2012. [Online]. Available: http://www.spec.org/osgcloud/docs/osgcloudwgreport20120410.pdf

[8] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the Weather Tomorrow?: Towards a Benchmark for the Cloud," in *Proceedings of the Second International Workshop on Testing Database Systems*, ser. DBTest '09. New York, NY, USA: ACM, 2009, pp. 9:1–9:6. [Online]. Available: http://doi.acm.org/10.1145/1594156.1594168

[9] T. Dory, B. Mejías, P. V. Roy, and N.-L. Tran, "Measuring Elasticity for Cloud Databases," in *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011. [Online]. Available: http://www.info.ucl.ac.be/~pvr/CC2011elasticityCRfinal.pdf

[10] R. F. Almeida, F. R. Sousa, S. Lifschitz, and J. C. Machado, "On Defining Metrics for Elasticity of Cloud Databases," in *Proceedings of the 28th Brazilian Symposium on Databases*, 2013. [Online]. Available: http://sbbd2013.cin.ufpe.br/Proceedings/artigos/sbbd_shp_12.html

[11] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a Consumer Can Measure Elasticity for Cloud Platforms," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/2188286.2188301

[12] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun, "Benchmarking in the Cloud: What It Should, Can, and Cannot Be," in *Selected Topics in Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin Heidelberg, 2012, vol. 7755, pp. 173–188. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36727-4_12

[13] B. Suleiman, "Elasticity Economics of Cloud-Based Applications," in *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing*, ser. SCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 694–695. [Online]. Available: http://dx.doi.org/10.1109/SCC.2012.65

[14] J. Weinman, "Time is Money: The Value of "On-Demand"," 2011, (accessed July 9, 2014). [Online]. Available: http://www.joeweinman.com/resources/Joe_Weinman_Time_Is_Money.pdf

[15] C. Tinnefeld, D. Taschik, and H. Plattner, "Quantifying the Elasticity of a Database Management System," in *DBKDA 2014, The Sixth International Conference on Advances in Databases, Knowledge, and Data Applications*, 2014, pp. 125–131. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=dbkda_2014_5_30_50076

[16] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What it is, and What it is Not (short paper)," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013)*. USENIX, June 2013. [Online]. Available: https://www.usenix.org/conference/icac13/elasticity-cloud-computing-what-it-and-what-it-not

[17] A. Weber, N. R. Herbst, H. Groenda, and S. Kounev, "Towards a Resource Elasticity Benchmark for Cloud Environments," in *Proceedings of the 2nd International Workshop on Hot Topics in Cloud Service Scalability (HotTopiCS 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, March 2014. [Online]. Available: http://sdq.ipd.kit.edu/research/publications/#WeHeGrKo2014-HotTopicsWS-ElaBench

[18] J. G. von Kistowski, N. R. Herbst, and S. Kounev, "LIMBO: A Tool For Modeling Variable Load Intensities (Demo Paper)," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, March 2014. [Online]. Available: http://dx.doi.org/10.1145/2568088.2576092

[19] ——, "Modeling Variations in Load Intensity over Time," in *Proceedings of the 3rd International Workshop on Large-Scale Testing (LT 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. ACM, March 2014. [Online]. Available: http://dx.doi.org/10.1145/2577036.2577037

[20] M. Kuperberg, N. R. Herbst, J. G. von Kistowski, and R. Reussner, "Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms," Karlsruhe Institute of Technology (KIT), Tech. Rep., 2011. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000023476

[21] 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*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[22] D. Shawky and A. Ali, "Defining a Measure of Cloud Computing Elasticity," in *Systems and Computer Science (ICSCS), 2012 1st International Conference on*, Aug 2012, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1109/IConSCS.2012.6502449

[23] K. Huppler, "Performance Evaluation and Benchmarking," R. Nambiar and M. Poess, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, ch. The Art of Building a Good Benchmark, pp. 18–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10424-4_3

[24] P. J. Fleming and J. J. Wallace, "How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results," *Commun. ACM*, vol. 29, no. 3, pp. 218–221, Mar. 1986. [Online]. Available: http://doi.acm.org/10.1145/5666.5673

[25] K. Huppler, "Benchmarking with Your Head in the Cloud," in *Topics in Performance Evaluation, Measurement and Characterization*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin Heidelberg, 2012, vol. 7144, pp. 97–110. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32627-1_7

[26] A. Weber, "Resource Elasticity Benchmarking in Cloud Environments," Master's Thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2014. [Online]. Available: http://sdqweb.ipd.kit.edu/publications/pdfs/Weber2014.pdf

[27] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open Versus Closed: A Cautionary Tale," in *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*, ser. NSDI'06. Berkeley, CA, USA: USENIX Association, 2006, pp. 18–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267680.1267698

[28] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces Principles, Patterns, and Practice*, ser. Java series. Addison-Wesley, 1999.

[29] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu, "Cutting Corners: Workbench Automation for Server Benchmarking," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 241–254. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404032

[30] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive Workload Classification and Forecasting for Proactive Resource Provisioning," *Concurrency and Computation: Practice and Experience*, 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3224