Quantifying cloud performance and dependability

Herbst, Nikolas; Bauer, André; Kounev, Samuel; Oikonomou, Giorgos; Van Eyk, Erwin; Kousiouris, George; Evangelinou, Athanasia; Krebs, Rouven; Brecht, Tim; Abad, Cristina L.; Iosup, Alexandru

published in
ACM Transactions on Modeling and Performance Evaluation of Computing Systems
2018

DOI (link to publisher)
10.1145/3236332

Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges

NIKOLAS HERBST, ANDRÉ BAUER, and SAMUEL KOUNEV, Universität Würzburg, Germany
GIORGOS OIKONOMOU and ERWIN VAN EYK, TU Delft, the Netherlands
GEORGE KOUSIOURIS and ATHANASIA EVANGELINOU, Nat'l. Tech. U. of Athens, Greece
ROUVEN KREBS, SAP SE, Germany
TIM BRECHT, University of Waterloo, Canada
CRISTINA L. ABAD, Escuela Superior Politécnica del Litoral, Ecuador
ALEXANDRU IOSUP, Vrije Universiteit Amsterdam and TU Delft, the Netherlands

In only a decade, cloud computing has emerged from a pursuit for a service-driven information and communication technology (ICT), becoming a significant fraction of the ICT market. Responding to the growth of the market, many alternative cloud services and their underlying systems are currently vying for the attention of cloud users and providers. To make informed choices between competing cloud service providers, permit the cost-benefit analysis of cloud-based systems, and enable system DevOps to evaluate and tune the performance of these complex ecosystems, appropriate performance metrics, benchmarks, tools, and methodologies are necessary. This requires re-examining old system properties and considering new system properties, possibly leading to the re-design of classic benchmarking metrics such as expressing performance as throughput and latency (response time). In this work, we address these requirements by focusing on four system properties: (i) *elasticity* of the cloud service, to accommodate large variations in the amount of service requested, (ii) *performance isolation* between the tenants of shared cloud systems and resulting *performance variability*, (iii) *availability* of cloud services and systems, and (iv) the *operational risk* of running a production system in a cloud environment. Focusing on key metrics for each of these properties, we review the state-of-the-art, then select or propose new metrics together with measurement approaches. We see the presented metrics as a foundation toward upcoming, future industry-standard cloud benchmarks.

19

This work was co-funded by the German Research Foundation (DFG) under Grant No. KO 3445/11-1, and by the Netherlands NWO Grants MagnaData and COMMIT; it has been supported by the CloudPerfect project and received funding from the EU H2020 research and innovation programme, topic ICT-06-2016: Cloud Computing, under Grant Agreement No. 73225. This research has been supported by the Research Group of the Standard Performance Evaluation Corporation (SPEC). The authors thank for reviewing Kai Sachs (SAP), Klaus-Dieter Lange (HPE), and Manoj K. Nambiar (Tata Consulting).
Authors' addresses: N. Herbst (corresponding author), A. Bauer, and S. Kounev, Informatik II, Universität Würzburg, Am Hubland, D-97074 Würzburg, Germany; emails: {nikolas.herbst, andre.bauer, samuel.kounev}@uni-wuerzburg.de; G. Oikonomou and E. van Eyk, EEMCS, Distributed Systems, P.O. Box 5031, 2600 GA Delft, The Netherlands; emails: {giorgoikonomou, erwinvaneyk}@gmail.com; G. Kousiouris and A. Evangelinou, National Technical University of Athens, School of Electrical and Computer Engineering, 9 Iroon Polytechniou St, 157 80 Athens, Greece; emails: {gkousiou, aevang}@mail.ntua.gr; R. Krebs, SAP Deutschland SE & Co. KG, Hasso-Plattner-Ring 7, D-69190 Walldorf, Germany; email: rouven.krebs@sap.com; T. Brecht, Davis Center, David R. Cheriton School of Computer Science, University of Waterloo, 200 University Ave. West, Waterloo, ON N2L 3G1, Canada; email: brecht@uwaterloo.ca; C. L. Abad, Escuela Superior Politecnica del Litoral, Campus Gustavo Galindo, Vía Perimetral 5, Guayaquil, Ecuador; email: cabad@fiec.espol.edu.ec; A. Iosup, Vrije Universiteit Amsterdam, De Boelelaan 1081, W&N Building, The Netherlands; email: A.Iosup@vu.nl.

https://doi.org/10.1145/3236332

CCS Concepts: • **General and reference** → **Cross-computing tools and techniques**; • **Software and its engineering** → **Virtual machines**; **Cloud computing**; • **Computer systems organization** → *Self-organizing autonomic computing*;

Additional Key Words and Phrases: Metrics, cloud, benchmarking, elasticity, performance isolation, performance variability, availability, operational risk

ACM Reference format:
Nikolas Herbst, André Bauer, Samuel Kounev, Giorgos Oikonomou, Erwin van Eyk, George Kousiouris, Athanasia Evangelinou, Rouven Krebs, Tim Brecht, Cristina L. Abad, and Alexandru Iosup. 2018. Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 4, Article 19 (August 2018), 36 pages.
https://doi.org/10.1145/3236332

1 INTRODUCTION

Cloud computing is a paradigm under which ICT services are offered on demand "as a service," where resources providing the service are dynamically adjusted to meet the needs of a varying workload. Over the past decade, cloud computing has become increasingly important for the information and communication technology (ICT) industry. Cloud applications already represent a significant share of the entire ICT market in Europe [20] (with similar fractions expected for North America, Middle East, and Asia). By 2025, over three-quarters of global business and personal data may reside in the cloud, according to a recent IDC report [36]. This promising growth trend makes clouds an interesting new target for benchmarking, with the goal of comparing, tuning, and improving the increasingly large set of cloud-based systems and applications, and the cloud fabric itself. However, traditional approaches to benchmarking my not be well suited to cloud computing environments. In classical benchmarking, system performance metrics are measured on systems-under-test (SUTs) that are well-defined, well-behaved, and often operating on a fixed or at least pre-defined set of resources. In contrast, cloud-based systems add new, different challenges to benchmarking, because they can be built using a rich yet volatile combination of infrastructure, platforms, and entire software stacks, which recursively can be built out of cloud systems and offered as cloud services. For example, to allow its subscribers to browse the offered and then to watch the selected videos on TVs, smart-phones, and other devices, Netflix utilizes a cloud service that provides its front-end services, operates its own cloud services to generate different bit-rate and device-specific encoding, and leverages many other cloud services from monitoring to payment. A key to benchmarking the rich service and resource tapestry that characterizes many cloud services and their underlying ecosystems is the re-definition of traditional benchmarking metrics for cloud settings, and the definition of new metrics that are unique to cloud computing. This is the focus of our work, and the main contribution of this article.

Academic studies, concerned public reports, and even company white papers indicate that a variety of new operational and user-centric *properties of system quality* (i.e., *non-functional properties*) are important in cloud settings. We consider four such properties. First, cloud systems are expected to deliver an illusion of infinite capacity and capability, yet to appear perfectly *elastic* to offer important economies of scale. Second, cloud services and systems have been shown to exhibit high-performance variability [38], against which modern cloud users now expect protection (*performance isolation*). Third, also because the recursive nature of cloud services can lead to cascading failures across multiple clouds when even one fails, increasingly more demanding users expect that the *availability* of cloud services is nearly perfect, and that even a few unavailability events will cause significant reputation and pecuniary damage to a cloud provider. Fourth, as the risks of not meeting implicit user-expectations and explicit service contracts (service level agreements, SLAs)

are increasing with the scale of cloud operations, cloud providers have become increasingly more interested in quantifying, reducing, and possibly reporting their *operational risk*.

With the market growing and maturing, many cloud services are now competing to retain existing customers and to attract new customers. Consequently, being able to benchmark, quantify, and compare the capabilities of competing systems is increasingly important. We first examine the research question: *For the four properties of cloud services we consider, can existing metrics be applied to cloud computing environments and be used to compare services?* Responding to this question, our survey of the state-of-the-art (see Section 9) indicates that the existing body of work on (cloud) performance measurement and assessment, albeit valuable, does not address satisfactorily the question, and in particular the existing metrics leave important conceptual and practical gaps in quantifying elasticity, performance isolation and variability, availability, and operational risk for cloud services. Therefore, we propose the main research question investigated in this work:

Q: Which new metrics can be useful to measure, examine, and compare cloud-based systems, for the four properties we consider in this work?

Both designing new metrics and proving their usefulness (possibly even in comparison with previous metrics) is non-trivial. Generally, metrics are designed to quantify a specific property of interest, and become useful only when shared by a diverse group of stakeholders. Metrics must address their design goals, which include domain-specific aspects, such as providing unique insights into the performance of cloud settings, and general aspects, such as the characteristics expected of a good metric we explain in Section 7. As with many other designed artifacts, proving that one metric is correct, or that one metric is better than another are not merely deductive or quantitative processes. Design often *satisfies* a need, rather than solving exactly (i.e., correctly) a problem. Different metrics are designed for different aspects and use cases; for example, the commonly used temperature metrics Celsius (interval scale) and Kelvin (ratio scale) metrics cannot be considered as one better than the other. Even when quantifying the same coarse-grained aspect, metrics can differ in some essential, albeit fine-grained, detail; for example, even the traditional utilization metric can offer different facets when considering heterogeneous resources and services.

Although elasticity, performance isolation and variability, availability, and operational risk are already perceived as important aspects in academia and by the industry, they have not yet been thoroughly defined and surveyed. As we show in this work, their meaning may be different for different stakeholders, and in some cases existing definitions are inconsistent or even contrary to each other. Toward answering the main research question, our main contribution is four-fold. Each contribution is focusing on defining the foundations of benchmarking one property of system quality in cloud settings and on showing exemplary applications to realistic cloud settings:

(1) *Elasticity*, addressed in Section 3. Elasticity offers the opportunity to automatically adapt the resource supply to a changing demand. Observing high response times during resource congestion or a low resource utilization during excess use allows only for an indirect, narrow understanding of the accuracy and timeliness of elastic adaptations. We present an extended set of metrics and methods for combining them to capture the accuracy and timing aspects of elastic platforms explicitly. We refine the elasticity metrics from previous work [32, 33, 37], introduce a well-defined uniform instability metric as well as the elastic deviation.

(2) *Performance isolation & variability*, addressed in Section 4. The underlying cloud infrastructure should isolate different customers sharing the same hardware from each other with regards to the performance they observe. We summarize how the impact of disruptive workloads onto specified performance guarantees connected with a cloud service

offering can be quantified as discussed in previous work [45]. In addition, we propose a new metric capturing performance variability seen as an effect of imperfect isolation.

(3) *Availability*, addressed in Section 5. We analyze the availability definitions used by various cloud providers, and quantify the availability of their business critical cloud applications and compare them for different contexts. We then define a metric of SLA adherence that enables direct comparisons between providers with otherwise different definitions of availability.

(4) *Operational risk-related*, addressed in Section 6. And on a more general level than the other features, we also focus in this work on estimating different types of operational risks that are connected with running software in the cloud. We define here various relevant metrics, and a measurement methodology that addresses them.

Because the practicality and utility of new metrics will be determined by whether or not they are adopted and how widely they are used, with this work we aim not only to answer the main research question but also to: (i) initiate discussions and provide a foundation regarding possible performance metrics, evaluation methodologies, and challenges that arise in cloud computing environments; (ii) examine methods for comparing the performance, dependability, and other non-functional properties of competing cloud offerings and technologies; (iii) influence emerging standardization efforts; and (iv) encourage providers of cloud computing services to support and make available monitoring of such non-functional properties, as a critical piece of infrastructure that is often lacking in current practice. Toward these goals (i–iv) and in addition to the four main contributions (1–4) mentioned above, we propose (I) in Section 2, a taxonomy of useful cloud metrics, (II) in Section 8, an in-depth discussion of open challenges that arise mainly in the context of containerization and micro-services as new technologies in the cloud computing domain. (III) Last, we review related work on cloud metrics in Section 9.

2 A TAXONOMY FOR CLOUD METRICS

In this section, we propose a cloud-metric hierarchy that reflects four different levels of abstraction in cloud performance measurement and assessment. Whereas in mathematics, the term metric is explicitly distinguished from the term measure (the former referring to a *distance function*), in computer science and engineering, the terms metric and measure overlap in meaning and are often used interchangeably. One way to distinguish between them is to look at metrics as values that can be derived from some fundamental measurements comprising one or more measures. In cloud settings, only a subset of the metrics can be measured directly at runtime by an end-user. Other metrics require detailed knowledge and controlled experiments, and thus need to be reported and measured using standards shared by cloud providers and infrastructure customers. Because there can be many or even endless derivations, and because such standards do not yet exist, it is useful to design a taxonomy of useful cloud metrics. We design such a taxonomy and depict it in Figure 1. We now address each layer, in turn.

The first layer includes *Traditional Metrics* for performance and dependability, such as end-to-end response time, average resource utilization, and throughput rates, are key instruments in the performance evaluation of cloud computing services. As we discuss in Section 8.1, to ensure well-defined and reproducible measurements, addressing the challenge of transparent placement of measurement sensors for these metrics has become more important in a cloud computing context.

We propose to group on the next-higher level the set of *Cloud Infrastructure Metrics* that cover various aspects of non-functional quality with a focus on the operation of the infrastructure, such as energy efficiency, or the capability of the infrastructure to isolate the performance hosted virtual machines and provide a certain degree of performance stability. This layer provides metrics useful in defining service level objectives, such as performance isolation and elasticity, efficiency,

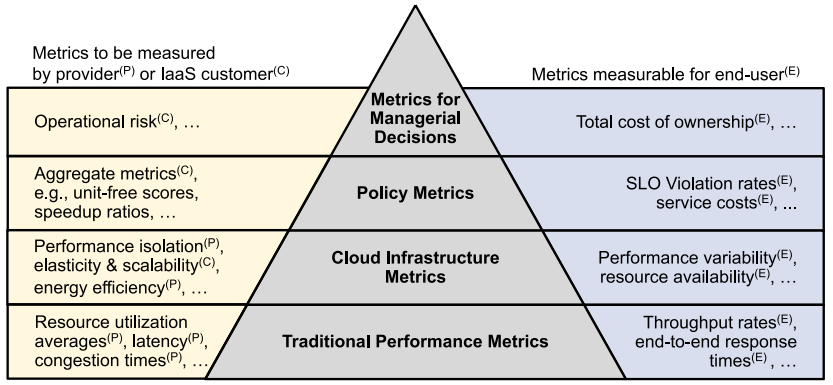


Fig. 1. A hierarchical taxonomy of cloud metrics.

and operational risk. These non-functional properties of a cloud environment or cloud service deployment are derived measures based on the Traditional Metrics. For example, Traditional Metrics characterizing a system using a response time distribution can be used to derive Cloud Infrastructure Metrics that inform about the quality of isolation between tenants, or if the system's resource management follows the demand for resources well-enough for efficient usage.

The group of *Policy Metrics*, the third layer, targets quantities relevant to an end-user. Examples include assessing the fulfillment of SLOs as experienced by users; and ranking and selecting cloud providers using unit-free scores aggregating multiple Cloud Infrastructure Metrics.

The topmost layer is comprised of *Metrics for Managerial Decisions*, which uses deep aggregates to quantify for example the risk and cost of running arbitrary cloud services in arbitrary cloud environments.

3 ELASTICITY

Elasticity has originally been defined in physics as a material property capturing the capability of returning to its original state after a deformation. In economics, elasticity captures the effect of change in one variable to another dependent variable. In both cases, elasticity is an intuitive concept and can be precisely described using mathematical formulas.

The concept of elasticity has been transferred to the context of clouds and is commonly considered a central attribute of the cloud paradigm [60]. The term is heavily used in providers' advertisements and even in the naming of specific products or services. Even though tremendous efforts are invested to enable cloud systems to behave in an elastic manner, no common and precise understanding of this term in the context of cloud computing has been established so far, and no industry-standard ways have been proposed to quantify and compare elastic behavior.

3.1 Prerequisites

The scalability of a system including all hardware, virtualization, and software layers within its boundaries is a prerequisite for speaking of elasticity. Scalability is the ability of a system to sustain increasing workloads with adequate performance, provided that hardware resources are added. In the context of distributed systems, it has been defined by Jogalekar and Woodside [42], as well as in the works of Duboc et al. [19], where also a measurement methodology is proposed.

The existence of at least one adaptation process is typically assumed. The process is normally automated, but it could contain manual steps. Without a defined adaptation process, a scalable system cannot scale in an elastic manner, as scalability on its own does not include temporal aspects.

When evaluating elasticity, the following points need to be checked beforehand:

- *Automated Scaling:* What adaptation process is used for automated scaling?
- *Elasticity Dimensions:* What is the set of resource types scaled as part of the adaptation process?
- *Resource Scaling Units:* For each resource type, in what unit is the amount of allocated resources varied?
- *Scalability Bounds:* For each resource type, what is the upper bound on the amount of resources that can be allocated?

3.2 Definition

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.

Dimensions and Core Aspects. Any given adaptation process is defined in the context of at least one or possibly multiple types of resources that can be scaled up or down as part of the adaptation. Each resource type can be seen as a separate dimension of the adaptation process with its own elasticity properties. If a resource type consists of other resources types, like in the case of a virtual machine having assigned CPU cores and memory, then elasticity can be considered at multiple levels. Normally, resources of a given resource type can only be provisioned in discrete units like CPU cores, VMs, or physical nodes. For each dimension of the adaptation process with respect to a specific resource type, elasticity captures the following core aspects of the adaptation:

Timing The timing aspect is captured by the percentages a system is in an under-provisioned, over-provisioned or perfect state and by the oscillation of adaptations.

Accuracy The accuracy of scaling is defined as the average relative deviation of the current amount of allocated resources from the actual resource demand.

A direct comparison between two systems in terms of elasticity is only possible if the same resource types (measured in identical units) are scaled. To evaluate the actual elasticity in a given scenario, one must define the criterion through which the amount of provisioned resources is considered to *match* the actual demand needed to satisfy the system's given performance requirements. Based on such a matching criterion, specific metrics that quantify the above mentioned core aspects, as discussed in more detail in Section 3.5, can be defined to quantify the practically achieved elasticity in comparison to the hypothetical *optimal elasticity*. The latter corresponds to the hypothetical case where the system is scalable with respect to all considered elasticity dimensions without any upper bounds on the amount of resources that can be provisioned and where resources are provisioned and de-provisioned immediately as they are needed exactly matching the actual demand at any point in time. *Optimal elasticity*, as defined here, would only be limited by the granularity of resource scaling units.

Differentiation. This paragraph discusses the conceptual differences between elasticity and the related terms, scalability and efficiency.

Scalability is a prerequisite for elasticity, but it does not consider temporal aspects of how fast, how often, and at what granularity scaling actions can be performed. Scalability is the ability of the system to sustain increasing workloads by making use of additional resources, and therefore, in contrast to elasticity, it is not directly related to how well the actual resource demands are matched by the provisioned resources at any point in time.

Efficiency expresses the amount of resources consumed for processing a given amount of work. In contrast to elasticity, efficiency is directly linked to resource types that are scaled as part of the system's adaptation mechanisms. Normally, better elasticity results in higher efficiency. This implication does not apply in the other direction, as efficiency can be influenced by other factors (e.g., different implementations of the same operation).

3.3 Related Elasticity Metrics and Measurement Methodologies

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

3.3.1 Related Elasticity Metrics. Several metrics for elasticity have been proposed so in literature:

The "scaling latency" metrics [49, 50] or the "provisioning interval" [12] capture the pure time to bring up or drop a resource. This duration is a technical property of an elastic environment independent of the timeliness and accuracy of demand changes, thus independent the elasticity mechanism itself that decides when to trigger a reconfiguration. We consider these metrics as insufficient to fully characterize the elasticity of a platform.

The "reaction time" metric we proposed at an early stage of our research already in 2011 in Reference [46] 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.

The "elastic speedup" metric proposed by SPEC OSG in Reference [12] relates the processing capability of a system at different scaling levels. This metric—contrary to intuition—does not capture the dynamic aspects of elasticity and is regarded as scalability metric.

The integral-based "agility" metric also proposed by SPEC OSG [12] compares the demand and supply over time normalized by the average demand. They state that the metric becomes invalid in cases where service level objectives (SLOs) are not met. This "agility" metric has not been included as part of the SPEC Cloud IaaS 2016 benchmark.[1] This metric resembles an early version of our proposed "precision" metric [32]. We propose a refined version normalized by time to capture the accuracy aspect of elastic adaptations, considering also situations when SLOs are not met.

The approaches in the work of Binning et al., Cooper et al., Almeida et al., and Dory et al. [3, 10, 15, 18] characterize elasticity indirectly by analyzing 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.

Becker et al. [8] introduced in 2015 the "mean-time-to-repair" in the context of elasticity as the time the systems needs on average to step out of an imperfectly provisioned state. This "mean-time-to-repair" is estimated indirectly based on analyzing for how long the request response times violate a given SLO or a systems runs below a target utilization—in other words, without knowing the exact demand for resources. Furthermore, the "mean-time-to-repair" metric comes with the assumption that for each defect a unique repair is given. In practice, based on experimental experience [33, 37, 48], this assumption does not hold. It is a common case that within one repair phase multiple further defects and/or repair actions can occur as illustrated in Figure 2. In these cases, this metric is not completely defined with the result that metric values might be computed based on different interpretations and thus can be misleading.

[1] SPEC Cloud IaaS 2016: https://www.spec.org/cloud_iaas2016/.

Numerous cost-based elasticity metrics have been proposed so far [26, 40, 66, 68, 73]. They quantify 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 *and* on the assumed penalty for under-provisioning, and thus do not support fair cross-platform comparisons.

We propose a set of elasticity metrics explicitly covering the timeliness, accuracy and stability aspects of a deployed elasticity mechanism (a.k.a. auto-scaler) together with three different way to aggregate in a possibly weighted manner. This set of core elasticity metrics can of course be complemented by user-oriented measures like response-time distributions, percentage of SLO violations, a cost measure by applying a cost model, or accounted and charged resource instance times.

3.3.2 Elasticity Measurement Approaches. As a high number of auto-scaling mechanisms has been proposed in literature over the course of the last decades, many of them come with individual and tailored evaluation approaches. We observe that in most cases an experimental comparison against other state-of-the-art mechanisms is omitted and the mechanism's effectiveness is justified by show-cases of improvements in service level compliance. The field of resource management algorithms has also been flooded by approaches only evaluated in simulation frameworks. A broader simulated analysis of auto-scalers can be found in the work of Papadopoulos et al. [57]. A number of approaches for elasticity evaluation—in some cases only on an abstract level—can be found in literature [3, 15, 18, 26, 40, 63, 66, 68, 73]:

In the work of Folkerts et al. [26], the requirements for a cloud benchmark are described on an abstract level. The work of Suleiman [66], Weinman [73], Shawky and Ali [63] as well as the work of Islam et al. [40] have in common their end-user perspective on elasticity and quantify costs or service level compliance. These are important end-user metrics, but we argue that good results in these metrics are indirectly affected by elasticity mechanisms. Timeliness, accuracy and stability of resource adaptations are not considered although those are the core aspect of an elastic behavior. Furthermore, these approaches account neither for differences in the performance 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. In contrast, the approach proposed in this work uses the results from an initial systematic scalability and performance analysis to adapt the generated load profile for evaluating elasticity, in such a way that differences in the platform performance and scalability are factored out.

Another major limitation of existing elasticity evaluation approaches is that systems are subjected to load intensity variations that are not representative for real-life workload scenarios. For example, in the work of Dory et al. [18] and of Shawky and Ali [63], the direction of scaling the workload downwards is completely omitted. In the work of Islam et al. [40], 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 "representative" [34] by employing the load profile modeling formalism DLIM as presented in the article [71].

3.4 Scalability Analysis as Preprocessing Step

To capture the curve of demanded resource units over an intensity-varying workload, the scalability of the system needs to be analyzed as a preprocessing step. As implemented in the BUNGEE Cloud Elasticity Benchmark [33], a scalability analysis results in a discrete function mapping the minimal amount of resources to a workload intensity while satisfying a given SLOs. The workload intensity can be specified either as the number of workload units (e.g., user requests) present at

the system at the same time (concurrency level) or as the number of workload units that arrive per unit of time (arrival rate). For the derivation of the mapping step function, it is proposed to conduct a binary search for finding the maximum sustainable load intensity per configuration. The characterization of the scalability per scaling dimension limits the evaluation of elasticity to one dimension at time.

3.5 Proposed Elasticity Metrics

The *demanded resource units* of a certain load intensity is the minimal amount of resources required for fulfilling a given performance-related SLO. The demanded resource units mapping for different load intensity levels can be derived with the methodology proposed in Section 3.4. The following metrics are designed to characterize two core aspects of elasticity: *accuracy* and *timing*.[2] As all metrics calculate their observed elasticity aspect based on the supply and the demanded resource units, each metric describes how the supply behavior differs from the demanded resource units; i.e., all metrics can be seen as deviation from the perfect behavior. Hence, the optimal value is zero and defines the perfect elastic system; the higher the metrics, the worse is the observed elasticity aspect. For a valid comparison of elasticity based on the proposed set of metrics, the platforms (i) require the existence of an autonomic adaption process (e.g., an auto-scaler), (ii) the scaling of the same resource type, e.g., CPU cores or VMs, and (iii) within the same ranges, e.g., 1 to 20 resource units.

The proposed metrics evaluate the elastic behavior and are not designed for distinct descriptions of the underlying hardware, the virtualization technology, the cloud management software or the elasticity strategy and its configuration in isolation. As a consequence, the metric and measurement methodology are applicable when not all influencing factors are known. The metrics require two discrete curves as input: (i) the curve of demanded resource units, which defines how the demanded resource units vary during the measurement period, where d_t is the demanded resource units at time $t \in [0, T]$; and (ii) the supply curve, which defines how the amount of used resources vary during the measurement, where s_t is the supplied resource units at time $t \in [0, T]$

In Section 3.5.1, we describe the *accuracy* metrics, and in Section 3.5.2, we present a set of metrics for the quantification of *timing* aspects. In Section 3.6, we outline two approaches for the aggregation of the proposed elasticity metrics enabling to compute a consistent ranking between multiple elastic cloud environments and configurations. Afterwards, we discuss challenges of elasticity in multi-tier applications. Finally, we conclude with a scaling behavior example and its associated metrics.

3.5.1 Accuracy. In contrast to the definition in the previous publication [32], the provisioning accuracy is extended so that it describes the relative amount of resources that are wrongly provisioned. Furthermore, we propose two different ways to calculate this metric: (i) the provisioning accuracy related to the current demanded resource units or (ii) the provisioning accuracy normalized by the maximum number of resources that are available for the experiment.

While using the first approach and taking Figure 2 into account, the under-provisioning accuracy θ_U is the sum of the areas U_t divided by the current demanded resource units d_t and, finally, normalized by the measurement period T. Similarly, the over-provisioning accuracy θ_O represents the relative amount of resources that are over-provisioned during the measurement interval:

$$\theta_U[\%] = \frac{100}{T} \cdot \sum_{t=1}^{T} \frac{max(d_t - s_t, 0)}{d_t} \Delta t, \qquad \theta_O[\%] = \frac{100}{T} \cdot \sum_{t=1}^{T} \frac{max(s_t - d_t, 0)}{d_t} \Delta t. \tag{1}$$

[2] In the work of Herbst et al. [32], these aspects are referred to as *precision* and *speed*.

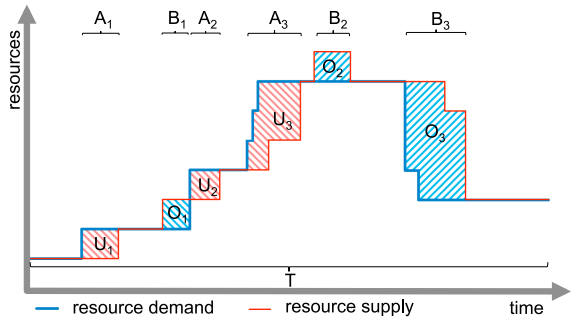


Fig. 2. Illustration for the definition of accuracy and provisioning timeshare metrics.

While using the second approach and considering Figure 2, θ'_U is the sum of the areas U_t normalized by the measurement period T and the maximum available resources units. Similarly, θ'_O is the sum of the of the areas O_t that is normalized by the time and the maximum resource units:

$$\theta'_U[\%] = \frac{100}{T} \cdot \sum_{t=1}^{T} \frac{max(d_t - s_t, 0)}{max_{t \in T}(d_t)} \Delta t, \qquad \theta'_O[\%] = \frac{100}{T} \cdot \sum_{t=1}^{T} \frac{max(s_t - d_t, 0)}{max_{t \in T}(d_t)} \Delta t. \tag{2}$$

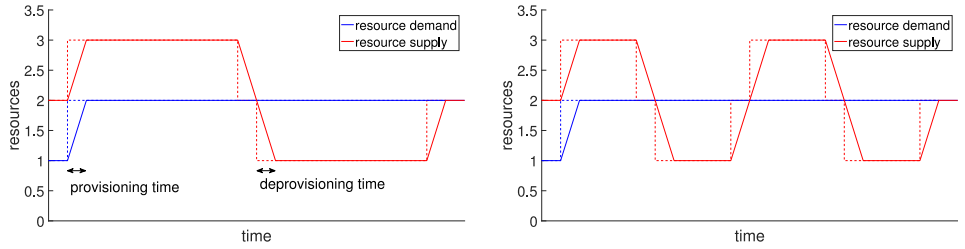
The advantage of the first approach is that the metric describes the relative deviations between the allocated resource units and their respective demanded resource units. However, the range of this metric is $[0;\infty)$, while the second approach gives values between 0 and 100 (as it describes the deviation related to the maximum available resource units). Herein, we use the term provisioning accuracy for both approaches as they are exchangeable for the aggregated elasticity metrics.

3.5.2 Timing. We highlight the *timing* aspect of elasticity from the viewpoints of the pure *wrong provisioning timeshare* and the *instability* accounting the degree of oscillation.

Wrong Provisioning Timeshare. The accuracy metrics allow no reasoning on whether the average amount of under-/over-provisioned resource units results from a few big deviations between demanded resource units and supply, or if it is caused by a constant small deviation. The following two metrics address this by giving more insights about the ratio of time in which under- or over-provisioning occurs. The metrics capture the time in percentage in which the system is under/over-provisioned during the experiment. In other words, The under-provisioning timeshare τ_U and over-provisioning timeshare τ_O are computed by summing up the total amount of time spend in an under- or over-provisioned state normalized by the duration of the measurement period (see Figure 2). Thus, τ_U and τ_O measure the overall timeshare spent in under- or over-provisioned states:

$$\tau_U[\%] = \frac{100}{T} \cdot \sum_{t=1}^{T} max(sgn(d_t - s_t), 0) \Delta t, \qquad \tau_O[\%] = \frac{100}{T} \cdot \sum_{t=1}^{T} max(sgn(s_t - d_t), 0) \Delta t. \tag{3}$$

Instability. Although the *accuracy* and *timeshare* metrics measure important aspects of elasticity, platforms can behave differently while producing the same metric values for *accuracy* and *timeshare* metrics. An example is shown in Figure 3: Platforms A and B exhibit the same accuracy and spend the same amount of time in the under-provisioned and over-provisioned states. However, B triggers three unnecessary resource supply adaptations, whereas A triggers seven. We propose to capture this with a further metric called *instability* to support reasoning for instance-time-based pricing models as well as for the operators view on estimating resource adaptation overheads.



(a) Platform A (b) Platform B

Fig. 3. Platforms with different elastic behaviors that produce equal results for *accuracy* and *timeshare*.

The *instability* metric υ is the time in percentage in which the curves of supplied and demanded resource units are not parallel. As a requirement for the calculation of this metric, the average provisioning and deprovisioning time has to be determined experimentally. The step-functions of demanded resource units and supply are transformed to ramps based on the average deprovisioning and provisioning time as depicted in Figure 3(a). Without this transformation, the resulting value would depend on how Δt is chosen. In summary, υ measures the fraction of time in which the demanded resource units and the supplied units do not change with same direction:

$$\upsilon = \frac{100}{T-1} \cdot \sum_{t=2}^{T} min(|sgn(\Delta s_t) - sgn(\Delta d_t)|, 1)\Delta t. \tag{4}$$

An *instability* value close to zero indicates that the platform adapts closely to a change in demand, like in Figure 3(a). A value close to 100 means that the platform oscillates heavily and does not converge to the demand, like in Figure 3(b). In contrast to the accuracy and timeshare metrics, a υ value of zero is a necessary but not sufficient requirement for a perfect elastic system.

3.6 Metric Aggregation

For a direct comparison of platforms, we propose two ways to aggregate the set of elasticity metrics and to build a consistent and fair ranking: (i) the *elastic deviation* and (ii) the *elastic speedup score*.

Elastic Deviation. To quantify the *elastic deviation* σ of an elasticity measurement k, we propose to calculate the deviation of the observed behavior to the theoretically optimal behavior (i.e., the curves of the supplied and demanded resource units are identical). A system with optimal elastic behavior in theory is assumed to have a value of 0 for this metric and, the higher the value, the worse the behavior. For the calculation of the deviation, we propose to compute the *Minkowski distance* d_p. As the theoretically optimal elastic behavior is assumed to have for each individual metric the value 0, the d_p-metric is reduced to the L_p-norm and is defined as

$$\sigma[\%] := \|x\|_5 = \left(\theta_U^5 + \theta_O^5 + \tau_U^5 + \tau_O^5 + \upsilon^5\right)^{\frac{1}{5}}. \tag{5}$$

Please note that user-defined weights could easily be added to set a preference for either over- or under-provisioning.

Elastic Speedup Score. The *elastic speedup score* ϵ is computed similar to the aggregation and ranking of results in established benchmarks, e.g., SPEC CPU2006.[3] Here, the use of the geometric mean to aggregate speedups in relation to a defined baseline scenario is a common approach.

[3]SPEC CPU2006: http://www.spec.org/cpu2006/.

The geometric mean produces consistent rankings and is suitable for normalized measurements [25]. The resulting *elastic speedup score* allows to compare elasticity performance without having to compare each elasticity metric separately, and a later point in time add a new result to the ranking (in contrast to a closed set in, e.g., a pair-wise competition).

A drawback of the elastic speedup score is its high sensitivity to values close to zero and becoming undefined if one or more of the metrics are zero. To minimize the probability of zero-valued metrics, we propose to aggregate the accuracy and timeshare sub metrics into a weighted accuracy and a weighted wrong provisioning timeshare metric, respectively:

$$\theta[\%] = w_{\theta_U} \cdot \theta_U + w_{\theta_O} \cdot \theta_O, \tag{6}$$
$$\tau[\%] = w_{\tau_U} \cdot \tau_U + w_{\tau_O} \cdot \tau_O, \qquad \text{with } w_{i_j} \in [0,1], w_{\theta_U} + w_{\theta_O} = 1 \text{ and } w_{\tau_U} + w_{\tau_O} = 1.$$

This way, θ and τ become zero only for the theoretical optimal auto-scaler. Thus, we compute the metrics of a baseline platform *base* and for elasticity measurement k as follows:

$$\epsilon_k = \left(\frac{\theta_{base}}{\theta_k}\right)^{w_\theta} \cdot \left(\frac{\tau_{base}}{\tau_k}\right)^{w_\tau} \cdot \left(\frac{\upsilon_{base}}{\upsilon_k}\right)^{w_\upsilon}, \qquad \text{with } w_\theta, w_\tau, w_\upsilon \in [0,1], w_\theta + w_\tau + w_\upsilon = 1. \tag{7}$$

The weights can be used (a) to realize user-defined preferences, e.g., for either putting more weight on under- or overprovisioning and (b) to adjust the impact of accuracy and timeshare metrics compared to instability if desired.

3.7 Elasticity Measurement Approach

This paragraph sketches an elasticity benchmarking concept that we propose as described in Reference [33], together with its implementation called BUNGEE.[4] Generic and cloud-specific benchmark requirements as stated in the work of Huppler et al. [34, 35] and Folkerts et al. [26] are considered in this approach. The four main steps in the measurement process can be summarized as follows:

(1) **Platform Analysis:** The benchmark analyzes the system under test (SUT) with respect to the performance 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 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:** Elasticity metrics are computed and used to compare the resource demand and resource supply curves with respect to different elasticity aspects.

3.8 Exemplary Experimental Elasticity Metric Results

This section presents an extract of evaluation from the work of Bauer et al. [7]. In this work, we measure the elasticity of a state-of-the-art auto-scaler depending on different inputs (Input I: estimated CPU utilization based on service demand law; Input II: measured CPU utilization using TOP command). The results of this measurement are depicted in Figure 4 and listed in Table 1, where each column lists an elasticity metric and each row represents a measurement. In the figure, the black curve represents the demanded VMs, which were determined by BUNGEE (Section 3.7), the blue dashed curve visualizes the auto-scaling behavior with the Input I, and the red dotted

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

Table 1. Elasticity Metrics

Metric	Input I	Input II
θ_U	6.49%	15.14%
θ_O	13.54%	7.05%
τ_U	18.87%	45.62%
τ_O	58.28%	31.72%
υ	24.34%	20.78%
σ	58.47%	47.21%
ϵ_k	2.61	2.64

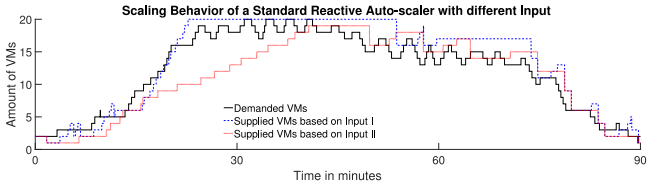


Fig. 4. Illustration of an auto-scaling example.

curve shows the scaling behavior based on the Input II. While observing the two different scaling behaviors, it is hard to decide which input results in a better performance as the different input leads to contrary performance: (i) While the auto-scaler with Input I almost fits the demand during the increasing load in the first 25min, the system is in an over-provision state during the remaining time. (ii) While the auto-scaler with Input II under-provisions the system until minute 35, it almost fits the demand during the remaining time. This is reflected in the elasticity metrics: The auto-scaler with Input I has significant better values for under-provisioning accuracy and time share, but has significant worse values for the over-provision accuracy and time-share. As the difference is quite similar for each compared metric, the elastic speed-up score is almost the same. A difference in performance can be seen in the auto-scaling deviation. Here, the auto-scaling behavior based on Input II achieves a better value in the context of elasticity. Note that the elasticity describes the system behavior and does not take user experience or efficient resource usage directly into account.

3.9 Discussion

Besides the publication mentioned in Section 3.8, the elasticity metrics are applied to describe auto-scaler behavior [37, 48] or are used to measure the elasticity of a system [33]. However, the proposed metrics are designed to quantify the elasticity on systems that host single-tier applications running on one group of resources. Hence, with the investigation of multi-tier application and multiple auto-scalers in place for sub-groups of resources or one multi-scaler some challenges occur: (i) While the search of the demand-intensity curve on a single-tier applications takes n steps (n is the number of resource units), the amount of steps for a naive search for a multi-tier application is $n_1 \cdot \ldots \cdot n_m$, where m is the number of tiers and n_i the number of resources of tier i. (ii) As the demanded resource units for a given intensity for a single tier application is distinct, the demand derivation for multi-tier application becomes more complex, because for a constant load intensity there could be more than one optimal resource configuration. (iii) While investigating each tier separately, the proposed metrics remain fully applicable; however, while investigating the metrics along all tiers, the proposed metrics are not completely sufficient. For instance, in the first tier, there is one resource too less and in the second tier, there is one resource too much. In sum, the number of resources are equal to the number of demanded resource units. Hence, the proposed elasticity measurement methodology and the metrics need extensions to handle such scenarios.

4 PERFORMANCE ISOLATION AND VARIABILITY

Cloud resources are shared among several customers on various layers like IaaS, PaaS, or SaaS. Thus, performance isolation of customers poses a major challenge. In this section, we sketch a way for quantifying performance isolation that has previously been proposed by Krebs et al. [45] and propose a new metric to quantify the performance variability of a cloud service offering.

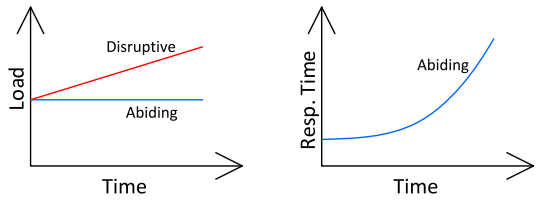


Fig. 5. Influence of the disruptive tenant onto the abiding.

Performance isolation in connection with a stable performance are important aspects for various stakeholders. When a platform developer has to develop a mechanism to ensure performance isolation between customers, he needs to validate the effectiveness of his approach to ensure the quality of the product. Furthermore, to improve an existing mechanism, he needs an isolation metric to compare different variants of the solution. When a system owner has to decide for one particular deployment in a virtual environment, his decision might be strongly influenced also by the quality of the non-functional performance isolation and stability properties.

The degree of performance isolation is perceived as a static quality of a given platform or environment, whereas performance variability is a dynamic property describing fluctuations in performance caused by imperfect isolation. To quantify performance isolation, full control of the cloud environment is required to distinguish between and physically collocate abiding and disruptive cloud tenants in a given experiment. Having an industry-standard benchmark for performance isolation, cloud providers could conduct trusted experiments and advertise their results. In contrast, performance variability can be quantified in a black-box manner by conducting a number of small performance measurements. Repeating performance measurements in randomized time intervals would enable the reporting of the recently experienced variability, as this quantity may change over time and depend on the time of the day.

A system is considered performance-isolated if, for customers working within their quotas, the performance is not negatively affected when other customers sharing the physical hardware exceed their quotas. A decreasing performance for the customers exceeding their quotas is accepted. A quota might be implicitly given by the specification of a service offering in combination with the current overbooking factor. Thus, a cloud customer must not explicitly be aware of his quota.

The isolation metrics [45] distinguish between groups of *disruptive* and *abiding* customers. The latter work within their given quota (e.g., defined number of requests/s) the former exceed their quota. Isolation metrics are based on the influence of the disruptive customers on the abiding customers. Thus, we have two groups and observe the performance of one group as a function of the workload of the other group (cf. Figure 5).

4.1 Isolation Metrics based on QoS impact

For the definition of the isolation metric, a set of symbols is specified in Table 2. These metrics depend on at least two measurements. First, the observed QoS results for every $t \in A$ at a reference workload W_{ref} where all customers stress the system at their quotas (or slightly less). Second, the results for every $t \in A$ at a workload W_{disr} when a subset of the customers have increased their load to challenge the system's isolation mechanisms. As previously defined, W_{ref} and W_{disr} are composed of the workload of the same set of customers, which is the union of A and D. At W_{disr} the workload of the disruptive customers is increased.

We consider the relative difference of the QoS (Δz_A) for abiding customers at the reference workload compared to the disruptive workload. Additionally, we consider the relative difference

Table 2. Overview of Variables and Symbols

Symbol	Meaning
t	A customer in the system.
D	Set of disruptive customers exceeding their quotas (e.g., contains customers inducing more than the allowed requests per second). $\|D\| > 0$
A	Set of abiding customers not exceeding their quotas (e.g., contains customers inducing less than the allowed requests per second). $\|A\| > 0$
w_t	Workload caused by customer t represented as numeric value $\in \mathbb{R}_0^+$. The workload is considered to increase with higher values (e.g., request rate and job size). $w_t \in W$
W	The total system workload as a set of the workloads induced by all individual customers. Thus, the load of the disruptive and abiding ones.
$z_t(W)$	A numeric value describing the QoS provided to customer t. The individual QoS a customer observes depends on the composed workload of all customer W. We consider QoS metrics where lower values of $z_t(W)$ correspond to better qualities (e.g., response time) and $z_t(W) \in \mathbb{R}_0^+$
I	The degree of isolation provided by the system. An index is added to distinguish different types of isolation metrics. The various indices are introduced later.

of the load induced by the two workloads (Δw):

$$\Delta z_A = \frac{\sum_{t \in A}[z_t(W_{disr}) - z_t(W_{ref})]}{\sum_{t \in A} z_t(W_{ref})}, \qquad \Delta w = \frac{\sum_{w_t \in W_{disr}} w_t - \sum_{w_t \in W_{ref}} w_t}{\sum_{w_t \in W_{ref}} w_t}. \tag{8}$$

Based on these two differences, the influence of the increased workload on the QoS of the abiding tenants is expressed as follows:

$$I_{QoS} = \frac{\Delta z_A}{\Delta w}. \tag{9}$$

A low value of this metric represents a good isolation as the difference of the QoS in relation to the increased workload is low. Accordingly, a high value expresses a bad isolation of the system.

Further isolation metrics can be found in the work of Krebs et al. [45], e.g., considering a measurement approach where the QoS of the abiding customer is kept constant by synthetically reducing its load. The amount of required reduction is taken as basis to compute isolation metrics.

4.2 Exemplary Experimental Isolation Metric Results

This section presents results based on the work of Krebs et al. [45] assessing different Xen hypervisor configurations (CPU pinning vs. credit scheduler) and their effects on the performance isolation I_{QoS} when executing the TPC-W benchmark for transactional web servers. Two different scenarios are investigated in this case study. In the *pinned* scenario, the server with eight physical cores hosts four guest systems (dom1, dom2, dom3, dom4) and dom0. Every domU has a fixed memory allocation of 3096MB and hosts a MySQL 5.0 database and a Tomcat web server. The various domains are exclusively pinned to the existing cores. Thus, no competition for the same CPU resources is possible. Based on this run-time environment, four separate instances of the TPC-W bookshop application are deployed. In the *unpinned* scenario, all domU and the dom0 are not pinned to a specific CPU and free to use any existing hardware resource. Xen's credit scheduler is chosen to allocate the domains to the various resources.

Table 3 shows the values used to define the reference and disruptive workloads, whereas EBs stands for the number of emulated browsers in the workload driver.

Table 3. Performance Results for the Scenario Setup and Configuration

Scenario	EBs per domU	Total throughput	Throughput per domU	Avg. response time	Max. load disruptive
Pinned	3,000	1,195req/s	299req/s	1,104ms	15,000
Unpinned	1,500	721req/s	180req/s	842ms	13,500

Table 4. Results of I_{QoS} in the Various Scenarios

Scenario	Disruptive load	Response time	Δw	Δz	I_{QoS}
Pinned	15,000	1,317ms	4.00	0.19	0.05
Unpinned	3,200	927ms	0.33	0.10	0.30
	4,800	942ms	0.60	0.12	0.20
	7,500	914ms	1.05	0.09	0.08
	10,000	1,173ms	1.47	0.39	0.27

The highest difference in throughput for one domain compared to the mean is around 4.5% and the highest difference of the response times around 6.5% in the pinned scenario. In the unpinned case, we observed 2.2% difference for the throughput. The difference of the response times was at 8.2%. As a consequence of these observations p_{end} is set to 15,000 for the pinned scenario and to 13,500 for the unpinned. It is worth to mention that in the unpinned scenario p_{end} is very close to nine times the load of the maximum throughput for one domain. Table 4 contains the values of I_{QoS} for the two scenarios. The first column of Table 4 identifies the scenario, the second the amount of users for the disruptive domain, the third column the average response time of all abiding domains followed by the results for Δw, Δz, I_{QoS}. To ensure a level playing field of comparison, we select measurements where Δw is the same in the relevant scenarios.

The pinned scenario presents a nearly perfect isolation throughout the whole range. The I_{QoS} presented in Table 4 at a disruptive load of 15,000 users was below 0.05.

For the isolation metrics based on QoS impact, we determine the isolation at various disruptive workloads shown in Table 4. We observe two significant characteristics. The first is the increasing response time when the disruptive load is set to 3,200 users. The second is the increasing response time at 10,000 users. Accordingly, the isolation becomes better between 3,200 and 10,000 users. This is because of the widely constant response times by increasing load changing the ratio of $\Delta z/\Delta w$.

4.3 Performance Variability of Virtual Cores

Typically, performance of virtualized computing offerings in the IaaS domain is indirectly expressed through the use of sizing parameters (such as RAM size, number of cores, etc.), with some providers using their own metrics (e.g., Amazon Compute Units). In the case of nominal sizing parameters, these fail to capture the actual performance of the resources when executing a given task, which may also depend on other parameters (e.g., sharing of some level of cache with other processors, bus speed, etc.). Furthermore, performance may fluctuate due to a number of other parameters (such as collocation of virtual resources in multi-tenant infrastructures, noisy neighbor effect, etc.). Thus, a more accurate representation of these computational capabilities could be based on a number of benchmark tests (potentially diverse, to capture different application types and ways to utilize the underlying hardware resources). It is not the purpose of this work to go into details on which these benchmarks should be. The main goal is to indicate the fluctuation aspects obtained by the benchmark scores, due to this dynamic and on demand nature of cloud services usage. This fluctuation should include no absolute values, but percentages of deviation from the average

Table 5. Performance of Virtual Cores Metrics

Cloud Provider	Instance Type	PDVC (%)	Avg. Bench. Score	Serv. Eff. Score
Flexiant	2Gb-2CPU	11.2	4,423.2	0.226
Flexiant	4Gb-3CPU	31.4	10,237.0	0.169
Flexiant	4Gb-4CPU	15.2	3,052.6	0.172
Microsoft Azure	A2	10.0	5,501.3	0.191
Microsoft Azure	A1	7.9	9,974.3	0.213
Amazon EC2	m1.large	20.0	5,070.7	0.173
Amazon EC2	m1.medium	11.1	6,692.2	0.217
Amazon EC2	m1.small	3.7	9,381.3	0.364

Table 6. Ranking Based on Performance of Virtual Cores Metrics

Ranks by PDVC	Ranks by Avg. Bench. Score	Ranks by Serv. Eff. Score
Amazon EC2 m1.small (best)	Flexiant 4Gb-4CPU (best)	Amazon EC2 m1.small (best)
Microsoft Azure A1	Flexiant 2Gb-2CPU	Flexiant 2Gb-2CPU
Microsoft Azure A2	Amazon EC2 m1.large	Amazon EC2 m1.medium
Amazon EC2 m1.medium	Microsoft Azure A2	Microsoft Azure A1
Flexiant 2Gb-2CPU	Amazon EC2 m1.medium	Microsoft Azure A2
Flexiant 4Gb-4CPU	Amazon EC2 m1.small	Amazon EC2 m1.large
Amazon EC2 m1.large	Microsoft Azure A1	Flexiant 4Gb-4CPU
Flexiant 4Gb-3CPU (worst)	Flexiant 4Gb-3CPU (worst)	Flexiant 4Gb-3CPU (worst)

values, according to the "performance of virtual core" metrics given in Table 5. From the two, we consider mainly the PVC (case 1), described below, which is a combination of the generic mean absolute deviation and the mean absolute percentage error formulas, if error in this case is considered the deviation from the mean value of the measurements. The metric (we denote it as PDVC, Percentage Deviation of Virtual Cores), calculates the average percentage deviation of the results from the mean value for the same benchmark, the same workload and the same size of VM. This metric, if also guaranteed by the respective provider, may indicate a promised limit C across all measurements, where n the observation set and C the guaranteed percentage by the provider (if any):

$$PVDC = \frac{\sum_{i=1}^{n} \frac{|x_i - \overline{x}|}{|\overline{x}|}}{n} < C. \tag{10}$$

Applied to the measurements of Evangelinou et al. [21], the specific metric portrays in Table 6 the following ratings for the various service types on the specific examined benchmarks.

Based on the specific rating, and compared to the original average benchmark values obtained, a clear difference may be observed in the values obtained for the PDVC metric, indicating an enhanced stability of the offering of Amazon small, which is ranked sixth in the absolute comparison based on the average benchmark score. Comparing to the service efficiency column (based on the service efficiency metric, a weighted sum of performance and cost as defined and used the work of Kousiouris et al. [44]), which is more balanced, since it includes an aspect of the offering's cost, the top rated one is again EC2 small; however, there are considerable differences in the following ones, with Azure offerings being evaluated with lower SE, while indicated as more stable from the PDVC. Flexiant 2GB-2CPU, however, appears as one of the best offerings in terms of absolute or relative (to cost) performance, while in the PDVC case is ranked in a lower position.

5 AVAILABILITY

We study critical aspects of measuring availability of cloud services, existing metrics and their relation to the de facto standards, and conditions that apply to the public cloud market. Relevance to cloud environments is high, given that availability is one of the strong arguments for using cloud services, together with the elastic nature of the resources and adaptable utilization. Furthermore, availability is a key performance indicator (KPI) included in most public cloud service level agreements (SLAs).

The goal is to identify metrics that can be used for provider and service comparisons or for incorporation into trust and reputation mechanisms. Furthermore, we intend to highlight aspects that are needed for an availability benchmark, which in contrast to most cases of benchmarking, does not refer to the creation of an elementary computational pattern that may be created to measure a system's performance. Instead, we refer mainly to a daemon-like monitoring client for auditing provider SLAs and extracting the relevant statistics.

5.1 Prerequisites and Context

SLAs are contracts between a service provider and a potential client, that govern or guarantee a specific aspect of the offered service. Relevant metrics may include availability, performance aspects, QoS, and so on, as defined service level objectives (SLOs). Research efforts have investigated advanced aspects like direct SLA negotiation and runtime renegotiation [11], and real-time QoS aspects [43]; however, in main public cloud environments they appear as static agreements [4, 30, 55], prepared in advance by the respective provider, and not available for adaptation to each specific user. Thus, a cloud user has no choice but accepting de facto the terms and conditions offered.

5.2 Relevant Metrics Definition

Operational Availability. For real-time monitoring of services availability, one very interesting approach is provided by CloudSleuth [14], which has an extensive network of deployed applications in numerous providers and locations and continuously monitors their performance with regard to response times and availability metrics. CloudSleuth's measurement way is not adapted to each provider's SLA definition, so it cannot be used to claim compensation but it relates mainly to the definition of operational availability. Furthermore, it checks the response of a web server (status 200 return type on a GET request). Thus, it cannot distinguish between a failure due to an unavailable VM (case of provider liability) or due to an application server crash (customer liability in case of IaaS deployment, provider liability in case of PaaS) or pure application fault (customer liability). However, it follows a normal availability definition that makes it feasible to compare services from different providers, a process that cannot be performed while following their specific SLA definitions, since they have differences in the way they define availability. Equation (11) contains CloudSleuth's formula for availability A_{CS}, with N as the overall number of samples S_i and M as the overall number of failed samples:

$$A_{CS} = \frac{\sum_{i=1}^{N} S_i - \sum_{i=1}^{M} S_i}{\sum_{i=1}^{N} S_i}. \tag{11}$$

Availability has also been defined using the Mean Time Between Failures [12], to avoid cases where small uptime is combined with a small downtime (and thus result in a high availability measure). While this is reasonable, commercial cloud providers tend to consider as uptime the entire month duration rather than only the actual uptime the services were used by the end user. Aceto et al. [2] conducted a thorough analysis of monitoring tools. While numerous tools exist for focusing on service availability (and other nonfunctional properties), it is questionable whether the way they

are calculating the specific metrics is compliant to the relevant definitions in commercial clouds and their respective SLAs. In the work of Li et al. [50], an investigation of new metrics and relevant measurement tools expands across different areas such as network, computation, memory and storage. However, availability is again not considered against the actual SLA definitions.

De Facto Industrial SLAs Examination. Commercial providers provide their own definition of availability in their respective SLAs. In many cases this implies a different calculation based on the same samples and a set of preconditions that are necessary for a specific service deployment to be included under the offered SLA. To investigate the ability to provide an abstraction layer for this type of services and thus create a generic and abstracted benchmarking client, one needs to identify common concepts in the way the SLAs are defined for the various providers (we considered Amazon EC2, Google Compute and Windows Azure). The first and foremost point of attention is the generic definition of availability A_g, which is included in Equation (12). Here, P_i is each elementary defined time interval (can be minutes, hours, etc., depending on the examined services) and DP_i is a failed examined (downtime) period. This may seem the same as the one defined in Equation (11); however, the providers typically define the downtime period as a step function (x in minutes). C_{min} is currently defined as 1min for the public cloud providers Google, Amazon, and Microsoft:

$$A_g = \frac{\sum_{i=1}^{N} P_i - \sum_{i=1}^{N} DP_i}{\sum_{i=1}^{N} P_i} \qquad DP(x) = \begin{cases} 0 & if\ x < C_{min} \\ x & if\ x > C_{min} \end{cases}. \tag{12}$$

Furthermore, one other key point is the existence of preconditions needed for an SLA to be applicable for a specific group of services used (e.g., need to have multiple VMs deployed in different availability zones). There are variations between provider definitions with regard to this aspect, with relation to where the VMs are deployed or if further action is necessary after the realization that a virtual resource is unavailable (e.g., restart of the resource). In a nutshell, the similarities and differences of the three examined providers are the following:

(1) Common concepts: the common concepts include the Quantum of downtime period, i.e. the fact that providers do not accept that a downtime period is valid, unless it is higher than a specific quantum of time. Furthermore, discount formats is the common compensation for downtime with relation to the monthly bill, while the calculation cycle is typically set to 1 month. Also, typically more than one VM need to be used to accept SLA applicability (distributed across at least two availability zones) and they need to be simultaneously unavailable for an overall sample to be considered as unavailable.
(2) Differences: differences include the quantum size (minor, since the format is the same but also because these differences tend to extinct with the typical example of Google Compute changing from 5 to 1min at the end of 2016). The number of discount levels is also different (again minor, since the format is the same). For the Azure Compute case, it seems that more than one instances for the same template ID must be deployed. However, it seems also that the overall time interval refers only to the time a VM was actually active. Other functional differences include the restart from the template, which is necessary in Google App Engine before validating an unavailable sample.

5.3 Abstracted and Comparable Metrics

It is not feasible to directly compare provider-defined availability metrics between providers, as these differ in definition. It is more meaningful to follow a more generic definition (as in Reference [14]), or abstract to the more generic concept of SLA adherence level, which can be defined as the

ratio of violated SLAs over the overall SLAs. Since SLA period is set to monthly cycles, this may be the minimum granularity of observation.

SLA Adherence Levels S_a. Special attention must be given for cases that sampling is not continuous, indicating that the client did not have running services for a given period, applicable for an SLA. These cases must be removed from such a ratio, especially for the cases that no violations are examined in the limited sampling period, given that no actual testing has been performed. If a violation is observed even for a limited period, then this may be included. Furthermore, SLA adherence may be grouped according to the complexity of the examined service

SLA Strictness Levels S_s. Besides SLA adherence, other metrics may be defined to depict the strictness of an SLA. As a prerequisite, we assume that the metric must follow a "higher is stricter" approach. Stricter implies that it is more difficult for a provider to maintain this SLA. To define such a metric, one needs to identify what are the critical factors that may affect strictness levels. Factors should be normalized to a specific interval (e.g., 0 to 1) and appropriate levels for them may be defined. Indicative factors may include:

(1) Size of the minimum continuous downtime period (quantum of downtime period q). A higher size means that the SLA is more relaxed, giving the ability to the provider to hide outages if they are less than the defined interval. The effect of such a factor may be of a linear fashion (e.g., $1 - q$). Necessary edges of the original interval (before the normalization process) may be defined per case, based, e.g., on existing examples of SLAs.
(2) Ability to use the running time of the services and not the overall monthly time, denoted by a Boolean variable t (0 false, 1 true). This would be stricter in the sense that we are not considering the time the service is not running as available.
(3) Percentage of availability that is guaranteed. Again this may be of a linear fashion, with logical intervals defined by the examined SLAs.
(4) Existence of performance metrics (e.g., response timing constraints). This may be a boolean feature x, whose values may be set to higher levels (0 or 5). The importance of this will be explained briefly.

Such metric may be useful for deploying applications with potentially different characteristics and requirements. For example, having *soft real-time applications* would imply that we need to have feature 4. Other less demanding applications may be accommodated by services whose SLAs are less strict. Thus, suitable value intervals may be adjusted for each feature. If we use a value of 5 for the true case of feature 4, and all the other features are linked in such a manner that their cumulative score is not higher than 5, then by indicating a necessary strictness level of 5 implies on a numerical level that feature 4 needs definitely to be existent in the selected cloud service.

Depending on the application types and their requirements and based on the metric definition, one can define categories of strictness based on the metric values that correspond to according levels (e.g., medium strictness needs a score from 2 to 3, etc.). It is evident that such a metric is based only on the SLA analysis and is static, if the SLA definition is not changed. The indicative formula for the case of equal importance to all parameters appears in Equation (13):

$$S = t + (1 - s_1 q) + s_2 p + x, \tag{13}$$
$$\text{where } s_i \text{ is a normalization factor for the continuous variables} \in [0, 1],$$
$$\text{so that } (s_1 \cdot q) \in [0, 1] \text{ and } (s_2 \cdot q) \cdot t \in \{0, 1\}, x \in \{0, 1\}.$$

For the normalization intervals, for p we have used 99% and 100% as the edges, given that these were the ranges encountered in the examined SLAs. For q we have used 0 and 10min as the edges;

Table 7. Example Application of the SLA Strictness Metric on Existing Public Cloud SLAs

Provider/Service	t	q	q'	p	x	S	S'
Google Compute	0	1 (norm: 0.1)	1 (norm: 0.0167)	99.95 (norm: 0.5)	0	1.4	1.4833
Amazon EC2	0	1 (norm: 0.1)	1 (norm: 0.0167)	99.95 (norm: 0.5)	0	1.4	1.4833
Microsoft Azure	1	1 (norm: 0.1)	1 (norm: 0.0167)	99.95 (norm: 0.5)	0	2.4	2.4833

0 indicates the case where no minimum interval is defined (thus abiding by the formula in Equation (11)) and 10 is the maximum interval in examined compute level SLAs. However, there are larger intervals (e.g., 60min) in terms of other layer SLAs (Azure Storage). The limit to 60 has been tried out in the q' case that is included in Table 7, along with the example of the other factors and the overall values of the SLA strictness metrics in the three examined public SLAs.

For this parameter, while in the past differences existed (Google had a 5min value), currently they are uniform to 1min. An example of x not being 0 would be the Azure Storage SLA, where unavailability is also determined by response time limits to a variety of service calls.

One interesting aspect in this case refers to the usage model of cloud services that may influence one or more parameters of the selection. As an example, if an entity uses cloud services as their main and continuous operation platform, then the concern of whether the actual usage time or the overall monthly time is used as the calculation interval becomes irrelevant (and, thus, the t parameter may be omitted, resulting in equivalent rankings for the providers of Table 7). If the usage model considers bursting scenarios (temporary usage of external cloud services to meet peaks in demand), then the respective t parameter gains significant importance in the decision process.

5.4 Measurement Methodology

To create an availability benchmark for the aforementioned SLAs, the following requirements/steps in the methodology must be undertaken:

(1) Complete alignment to each provider definition to achieve **non-repudiation**, including:
   (a) Necessary **preconditions** checking in terms of number and type
   (b) Availability **calculation** formula
   (c) **Dynamic** consideration of user actions (e.g., starting/stopping of a VM) that may influence SLA applicability
   (d) Downtime due to **maintenance**
(2) General assurance mechanisms in terms of faults that may be accredited to 3rd party services/equipment:
   (a) For example testing of general Internet connectivity on the client side
   (b) Front-end API availability of provider
   (c) Monitoring daemon not running for an interval at the client side. This can be covered by appropriate assumptions, e.g., if logs are not available for an interval then the services are considered as available
(3) Logging mechanisms that persist the necessary range and type of information

System Setup. System setup should include a running cloud service. Furthermore, it should include the benchmark/auditing code (according to the aforementioned requirements taken under consideration) that is typically running externally to the cloud infrastructure, to cover the case that connectivity exists internally in the provider but not toward the outside world.

Workload. Given that this is a daemon-like benchmark, the concept of workload is not directly applicable. The only aspect of workload that would apply would be for the number of cloud service

instances to be monitored and the only constraint is that these cover the preconditions of the SLA. However, an interesting consideration in this case may be the differentiation based on the complexity of the observed service (in terms of number of virtual resources used), given that this would influence the probabilities of not having a violation.

If we consider the case of a typical cloud deployment at the IaaS level, then we may use N availability zones (AZ), in which M virtual machines are deployed. An availability zone is typically defined as a part of the data center that shares the same networking and/or power supply. Thus, the usage of multiple AZs eliminates the existence of a single point of failure. For simplicity purposes, we assume that the number of VMs is the same across all AZs. The probability of a technical failure in the AZ P_T depends on the probability of power supply failure P_P and the probability of network failure P_N. The probability that a VM is unavailable P_U contains P_H the risk of the physical host in which a VM is running to fail and P_V the risk of the VM to fail. Assuming that these probabilities are mutually exclusive and depending on different factors, the overall probability of failure for a deployment in one AZ P_A is given by Equation (14). The service is deemed as unavailable in one specific AZ if power or network connectivity is lost across the AZ or if all VMs in that AZ are at the same time unavailable:

$$P_A = P_T + \prod_{i=1}^{M} P_{U_i} = P_T + P_U^M, \quad \text{with} \quad P_T = P_P + P_N \quad \text{and} \quad P_U = P_H + P_V. \tag{14}$$

If M VMs are deployed in each one of the N AZs, then the overall failure probability P_O is given by Equation (15), assuming that the various AZs have similar power and network probability failures and given that we are not aware of the affinity of VM placement across physical nodes, thus we can assume that different physical hosts are used for each VM:

$$P_O = \prod_{i=1}^{N} P_{A_i} = (P_T + P_U^M)^N = \sum_{k=0}^{N} \frac{N!}{k!(N-k)!} P_T^k P_U^{M(N-k)}. \tag{15}$$

The significant factors that indicate the complexity (M and N) can be used as a generic metric of "workload." In addition, they can be used to classify results according to service complexity.

5.5 Discussion

While availability has been defined in the literature in various ways, existing mainstream public clouds such as Amazon, Google and Azure have separate definitions, which may be similar but not identical even to each other. Thus, direct comparison of providers based on these metrics cannot be achieved and especially benchmarked against the guarantees they advertise in their respective SLAs. In this section, an analysis is performed regarding the similarities of provider definitions and how these can lead to guidelines regarding the implementation of benchmarking clients for identifying provider fulfillment of the issued service level agreements toward their customers.

Furthermore, we define a simple yet directly comparable (between providers) metric of SLA adherence. Classes of workload can be identified based on the size and deployment characteristics of the measured service thus further refining the aforementioned comparison.

6 OPERATIONAL RISK

In this section, we present cloud metrics laid on the top level of the pyramid in Figure 1. These metrics may provide comparable information about the overall status of cloud systems and can further be used for managerial decisions by operators and stakeholders. We define the Operational Risk (OR), a family of metrics determining the risk of production systems running in cloud environments, and provide metrics and measurement approaches to quantify it.

6.1 Goal and Relevance

Risk management is defined as a decision paradigm in grids [17] and used for the selection of an infrastructure to host an application. Similarly, in clouds we use risk as a quality aspect reflecting the impact of running an application in cloud infrastructures.

The operational risk derives from the risk aspect and depicts the performance impact on services when running in cloud systems. Operational risk evaluates whether cloud services run in the system as expected and, if not, the severity of the deviations. Given that cloud users want their applications to run with minimum performance degradation, providers may use metrics such the operational risk to publicly present the capabilities of their systems.

In addition, operational risk helps managers summarize and report system status in management analysis. For instance, cloud architects need metrics to depict the overall system status and, if possible, the future impact of cloud solutions on system performance. However, operators monitoring the cloud infrastructures use different metrics for in-depth performance analysis.

The operational risk provides an overview of the system and contributes to a higher-level approach toward the performance evaluation of complex systems.

6.2 Prerequisites and Context

In this section, we describe the operational risk of cloud services deriving from their performance levels when running in cloud infrastructures. The term *risk* is also used in literature for describing security issues of cloud but the notion of security is out of scope of the presented metric.

The *service performance* in clouds refers to multiple kinds of performance that can be measured in cloud services. A possible distinction among these performance types is the level of service (e.g., IaaS, PaaS, SaaS) in the cloud system. The service level indicates the service performance for that level and also the suitable metrics measuring the performance.

The definition of operational risk refers to *expected* performance levels that cloud services should run at to avoid degradation. The expected performance varies depending on needs. We describe the most common benchmark objectives from which the expected performance levels derive.

First, the performance can be compared to an optimal level that bounds theoretically the measurements. The implied expected performance to compare results is the optimal level. An example is the total availability of cloud infrastructures with no downtime, as described in Section 5.

A second benchmark result can be the comparison between the performance level of a service running alone on a dedicated environment and the performance of the same service running on an environment where multiple services share the resources. This is a more realistic approach to benchmark and to compare performances, since optimal performances may never be reached.

A third result in cloud benchmarks can be the success of the system in achieving the guaranteed performance levels described in SLOs. Providers may use benchmarking to test systems against possible SLO violations and create new SLA levels according to the results. Therefore, we set as expected performance levels of services those defined in SLOs. In this way, we can test and tune systems according to required performance bounds. Having SLA levels as a benchmark goal requires the definition of such SLOs, which are general and applicable for many cloud-based systems.

As a result of the many variations of performance levels set as benchmark goals, we can map these levels to the expected performance levels in operational risk. In this way, operational risk can evaluate the severity of the deviations between monitored results and given expected levels.

6.3 Proposed Operational Risk Metrics

The proposed operational risk metric adapts the notion of the expected levels from the second comparison type described in Section 6.2.

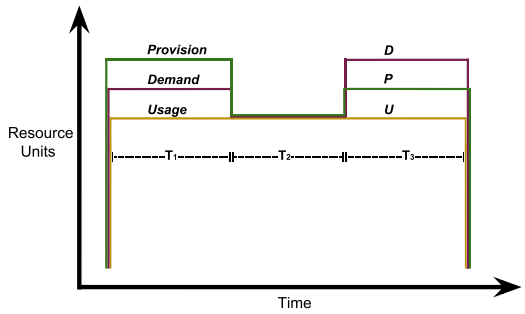


Fig. 6. Metrics composing the Operational Risk metric for cloud service.

The metric quantifies the variation of service performance between the service running in a dedicated environment and the performance when running in another, non-dedicated environment. The former provides an isolated environment where resources are always available to the service under test. In contrast, due to concurrent operations of services in a non-dedicated environment, services share the resources resulting in performance interference among the services [77]. This interference affects the performance delivered in services. Hence, this performance may deviate from the service performance in the dedicated environments implying degradation issues.

Since we use the notion of the expected levels for testing performance, by *risk*, we imply the likelihood that the service performance in the cloud will deviate from the demanded performance. The expected service performance is reflected by the performance in the dedicated system.

We focus on the service performance at the IaaS level, thus the operational risk utilizes IaaS-level metrics and refers to performance levels that reflect the resource utilization by cloud services.

6.3.1 Related Metrics. The risk metric reveals an additional aspect of the service performance in the cloud, which is the degree to which service performance is degraded. Figure 6 shows the three metrics referring to resource type of a cloud service: the current usage (U) of the resource, the demanded (D) amount the service requires, and the provisioned (P) amount of resource, which is the upper limit of resource the service is able to consume and is given by the resource manager.

As Figure 6 shows, the D line can be higher than the U line as a service may not receive what it requires due to contention issues in the system, e.g., resource overbooking. We consider that usage U cannot be higher than the demanded amount D ($U \not> D$), because the real usage of resource cannot exceed the requirements of service in resources. Similarly, $U \not> P$ as the actual resource consumption can only reach up to the resource limit having been set by the resource manager.

Relative Levels of Metrics. Considering the three mentioned metrics, we show the possible cases about the relative levels of the metrics. In an ideally auto-scaling and isolated environment, the service is provisioned and consumes the demanded amount of resource ($P = D = U$), as in period T_2 in Figure 6. The resource manager provides the demanded resources ($P = D$) and the service utilizes all the demanded resources ($D = U$) without interfering with other co-hosted services.

However, the case of the three coinciding lines is not applicable in real environments. An auto-scaling mechanism is not perfectly accurate and creates incompatibilities between the provisioned and the demanded resources. When a service demands fewer resources than it has been provisioned ($D < P$), the system hosts an over-provisioned service. The system has under-provisioned resources to service when $D > P$. The two mentioned cases are depicted in periods T_1, T_3 in Figure 6.

Finally, the real usage U cannot be higher than the demanded usage D; however, the lines of U and D do not always coincide due to contention issues in the system. We call the deviation between D and U, which is always non-negative ($D \geq U$), as *contention-metric* and it presents the gap between the demanded amount of resource and the real consumption of resource at specific time. The higher the value of the contention-metric, the more severe is the resource-contention that is experienced by the service; hence degrading the service performance. For instance, in Figure 6, the service experiences more contention in its resources during period T_3 than during period T_1 as the gap between the resource lines D, U is bigger in the former period.

6.3.2 Definition of Metric. The operational risk metric presented in this section incorporates the three resource metrics mentioned before and splits the metric into two partial metrics, the *provision risk* (r_p) and the *contention risk* (r_c). We split the risk metric into two so that each metric evaluates different risk aspects when service performances are degraded. The provision risk evaluates the severity of performance degradation affected by inaccurate provisioned resources to services, and the contention risk, the severity affected by the resource contention when resources are shared among services. The combination of the two metrics, will let cloud providers have a more comprehensive view of the system performance and adjust performance issues toward their needs.

Provision Risk. We define the *provision risk* (r_p) as the degree to which the demanded resources meet the provisioned resources:

$$r_p = \frac{1}{T} \int_T \frac{P_t - D_t}{P_t} dt, \qquad [-1, 1], \tag{16}$$

where P_t and D_t are the provisioned and the demanded resources respectively at time t. T is the time period in which we measure the two metrics. The integral value of r_p measures the relative squashed area enclosed by the resource values of P and D for time period T. The value of r_p ranges between $[-1, 1]$, indicating an under-provisioning situation when $r_p \in [-1, 0)$ and an over-provisioning case when $r_p \in (0, 1]$. The zero value indicates an accurate provision of resources according to the demanded resources. The closer the value of r_p is to zero, the less risk is indicated for the service.

Contention Risk. Similarly, we define the *contention risk* (r_c). This metric utilizes the resource values of demanded and used resources over time T. The metrics D_t and U_t indicate the values of the respective metrics at time t:

$$r_c = \frac{1}{T} \int_T \frac{D_t - U_t}{D_t} dt, \qquad [0, 1]. \tag{17}$$

The r_c value is non-negative as the amount of used resources cannot exceed the amount of demanded resources. The higher the value of r_c, the more risk is estimated for a service to not receive the demanded performance due to resource contention.

Service Risk. The risk for a cloud service should be composed of the combination of the two aforementioned risk metrics to have better overview on the service status regarding risk levels.

We define the *risk of service* (r_e) as the degree to which a cloud service performs as expected, which derives from the performance of the provisioning capabilities of the system and from the system's capacity to keep low resource contention among hosted services. Both expected performance levels contribute to evaluate if a system supplies the service adequately with enough

resources:

$$r_e = w_p \cdot |r_p| + w_c \cdot r_c = \frac{1}{T}\left(w_p \cdot \int_T \frac{|P_t - D_t|}{P_t} dt + w_c \cdot \int_T \frac{D_t - U_t}{D_t} dt\right), \; [0,1], \quad \text{with } w_p, w_c \in [0,1], w_p + w_c = 1. \tag{18}$$

For the service risk metric r_e in Equation (18), the difference between P_t and D_t is an absolute value, because we do not focus on the provisioning type of risk but only for the level of severity that the difference between the two metrics reflects. Factors $w_p, w_c \in [0,1]$ are used to weight the operational risk value according to user needs.

System Risk. The operational risk of a cloud system (r_s) should be an overview of the risk values of the constituent services in the system. The aggregation method of service risks, which calculates the system risk value, deviates according to user needs. The method will result in a variability metric that combines values according to different purposes. For example, providers who want to test the risk levels of the system may use quantile values to depict the risk level of a proportion of services. When cloud environments are assessed for stable risk levels, a variability metric, like the inter-quartile range (*IQR*), will show the dispersion of service risks in the systems and thus, the performance variability in the systems.

6.4 Measurement Methodology

To measure effectively the operational risk of cloud systems, we have to determine which monitored data is the respective metrics that operational risk is built upon. Additionally, the resource type has to be decided in order for the appropriate resource metrics to be declared. We propose the measurement of multiple resource types, since objectives in resource management by cloud operators need to consider the joint management of different resources [41]. We focus on the most common resource types that are currently available to clouds: CPU, memory, network, and storage.

Metrics P & U. The metric definitions of provision (P) and usage (U) are straight-forward and one can easily monitor the respective values. Metric P refers to the capacity that resource manager has provisioned to service and metric U is the actual resource capacity that service uses. In any resource type, the corresponding metrics of P and U can be readily monitored.

Metric D. Although it may be confusing how the demanded amount of the aforementioned resource types is estimated, there is currently enough research on that topic. For the resource type of CPU, prediction models have been introduced in Reference [39] and contemporary cloud monitors provide metrics about the demanded amount of CPU. For the memory resource, the demanded amount can also be estimated as in Reference [76] and used as the possible memory capacity that service needs at specific time. For the resource types of network and storage, the metric D is simpler to calculate, because the demanded capacities are either the size or the number of requests of that resource received by the resource manager. The resource manager, after collecting the requests, handles a subset of these requests (i.e., metric U) due to the system load.

Weights w_p, w_c. The values of the two weights w_p, w_c represent the importance of the two metrics r_p, r_c, respectively. One has to consider the purpose of benchmarking the cloud system to define similarly the weights. The *contention risk* may be more important for testing the impact of co-location of cloud services ($w_p < w_c$), whereas the *provision risk* represents better the operational risk when selecting the most promising elastic policy for a system.

Table 8. Metric Results for a Simulated Cloud Environment ($w_c = 1$)

T [#months]	W [#VMs]	W [#VMs/service]	$\overline{r_e}$	r_s [Q_2]
3	1,800	11	0.05	0.001
3	3,400	61	0.09	0.007

6.5 Measurement Approach

The approach of evaluating the operational risk from Section 6.3 can be summarized as follows:

(1) **Measurement:** The benchmark exposes the system-under-test to a given workload W for specific time period T. Given the workload resource demands, the resource allocations for the workload by the system and the contention levels in the system, the benchmark estimates the current resource usage U and the provision amount P across period T.
(2) **Risk Evaluation:** Operational risk metrics are computed and used to compare the risk in different time periods or different cloud environments.

We present an example of measuring operational risk using simulation of cloud services running in a multi-datacenter multi-cluster system. To simplify the process, we focus on the contention risk of a service ($r_e = r_c$) and for one resource type (CPU). The workload trace has a long time period of three months, because short-term performance degradation of services do not show representative overview of their severity. Monitoring and testing long workload traces is the main reason of having simulation results for our measurements. The computed risk metrics are shown in Table 8.

Each table row represents an experimental configuration with the second service load stressing higher the system. The different operational risk values in the configurations reflect the different impact of the CPU workloads in the system. By measuring the risk, providers may focus on the most contended services, therefore services with risk values higher than the system risk ($r_e > r_s$).

6.6 Related Work for Operational Risk Metrics

Risk management and resource contention are not new subjects in cloud research. Risk management has been defined as a decision paradigm in Grids [17], where selection among multiple infrastructures should be taken into account to host an application. Historical records about SLO violations can be used to assess the system risk for an incoming application to fulfill the agreed objectives [17, 24]. The risk levels are also evaluated according to the provisioned mismatches of the elasticity mechanism. Our approach of operational risk considers the elastic aspect of the system and incorporates it with the risk of performance interference of services.

Persico et al. [58, 59] studied the intra-network performance in AWS EC2 and Microsoft Azure, suggesting ways to properly characterize the maximum achievable throughput; studies like these are necessary to correctly define the baseline-level prior to a performance degradation analysis. Tang et al. [67] investigated the interference of services in memory. The presented results on performance degradation use as baseline-level the performance of a service running alone in the system under test. Zhuravlev et al. [77] used a similar approach measuring the performance degradation, with a more explicit definition of the degradation. The authors define the relative performance degradation of memory components according to the performance level of the service running alone. We extend this related work considering the impact of inaccurately provisioned resources in the hosted services. The degradation of performance due to resource contention is affected by the elastic mechanisms in clouds, thus the need to incorporate the two notions.

6.7 Discussion

In this section, we present the feature of operational risk in clouds and introduce a representative metric evaluating the risk in cloud services and systems not to perform as expected.

The notion of risk differentiates from the elasticity feature in cloud, because elasticity takes into account the provisioned and the demanded values of service resource to cope with the service load. In contrast, both performance isolation and availability utilize demand and usage metrics for different reasons. Performance isolation concerns about the contention that a service may experience and evaluates the severity of the interference among services while availability evaluates the periods where demand is present but the usage of resources cannot be achieved.

The operational risk incorporates the three resource-level metrics and complements the other metrics described in previous sections to assess the severity of performance degradation regarding the mentioned cloud features.

Usability. The operational risk metric can be used as evaluation of cloud services and systems to assess whether the performance guarantees are met.

The measurement of service-performance in the cloud is important for both customer and cloud provider. The customer wants to maximize profit by delivering good QoS to clients. Therefore, the service performance is utilized by the customer to check the progress of service-runtime as well as to compare and select the cloud environment that meets the service needs the most.

For cloud providers, they are interested in the levels of service performance, because they are burdened with financial fees when SLO violations occur. Moreover, there is a reputation cost for cloud providers when not delivering good performance results to customers, thus, providers also utilize service-performance metrics to maintain competition in the cloud market.

7 METRIC DISCUSSION

In the previous sections, we defined a number of cloud relevant metrics for different non-functional quality aspects, including ways to aggregate them. In Table 9, we provide an overview on the presented metrics, including their value ranges, optimal points and references to show cases. In general, it is impossible to prove correctness or superiority of a metric; a metric is a common agreement on how to quantify a given property. One can discuss to what degree metrics fulfill characteristics of a good, well-defined, and intuitive metric and additionally demonstrate their usefulness by meaningful results in rows of experiments or adoption in different communities. Let us go in the following step-by-step over a list of metric characteristics:

Definition. A metric should come along with a precise, clear mathematical (symbolic) expression and a defined unit of measure, to assure consistent application and interpretation. We are compliant with this requirement, as all of our proposed metrics come with a mathematical expression, a unit, or are simply time-based ratios.

Interpretation. A metric should be intuitively understandable. We address this by keeping the metrics simplistic and describe the meaning of each in a compact sentence. Furthermore, it is important to specify (i) if a metric has a physical unit or is unite-free, (ii) if it is normalized and if yes, how, (iii) if it is directly time-dependent or can only be computed ex-post after a completed measurement, and (iv) clear information on the value range and the optimal point. Aggregate metrics should keep generality and fairness, combined with a way to customize by agreement on a weight vector.

Reliability. A metric is considered reliable if it ranks experiment results consistently with respect to the property that is subject of evaluation. In other words, if System A performs better than System B with respect to the property under evaluation, then the values of the metric for the two

Table 9. Metric Summary

Quality Attribute	Metric		Value Range	Unit	How to Measure	Show Case
Elasticity	Accuracy	θ_U	$[0;\infty)$, opt: 0	%	ex post, calibration required	✓ Section 3.8 [7, 37]
		θ_O	$[0;\infty)$, opt: 0	%		
		θ'_U	$[0;100]$, opt: 0	%		
		θ'_O	$[0;100]$, opt: 0	%		
	Timeshare	τ_U	$[0;\infty)$, opt: 0	%		
		τ_O	$[0;\infty)$, opt: 0	%		
	Instability	υ	$[0;100]$, opt: 0	%		
	Deviation	σ	$[0;\infty)$, opt: 0	%	ex post	
	Speedup	ϵ_k	$[0;\infty)$, opt: ∞	None	ex post	
Isolation	QoS	I_{QoS}	$[0;\infty]$, opt: 0	None	ex post	✓ Section 4.2 [45]
Variability	Deviation	$PVDC$	$[0;100]$, opt: 0	%	ex post	✓ Section 4.3 [21]
Availability	Adherence	S_a	$[0;100]$, opt: 100	%	ex post	✓ Section 5.3 appl. on pub. clouds[5]
	Strictness	S_s	$[0;\infty]$, opt: ∞	%	der. from SLO def.	
Operational Risk	Provision	r_p	$[-1;1]$, opt: 0	None	ex post	✓ Section 6.5
	Contention	r_c	$[0;1]$, opt: 0	None		
	Service	r_e	$[0;1]$, opt: 0	None		
	System	r_s	$[0;1]$, opt: 0	None		

systems should consistently indicate this (e.g., higher value meaning better score). In the intuitive optimal case, a metric is considered linear if its value is linearly proportional to the degree to which the system under test exhibits the property under evaluation. For example, if a performance metric is linear, then twice as high value of the metric would indicate twice as good performance. Linear metrics are intuitively appealing, since humans typically tend to think in linear terms. For the aggregate metrics, linearity might not be given. In the long run, the metric's distributions in reality should be analyzed to improve the reliability of their aggregation. For our proposed metrics, we demonstrate consistent rankings in rows of experiments.

Further Quality Aspect of Metrics in Combination with a Standardized Measurement. A transparently defined and consistently applied measurement procedure is important for reliable measurements of a metric. For example, it is important to state where the sensors need to be placed (to have an unambiguous view on the respective resource), the frequency of sampling idle/busy counters and the intervals for reporting averaged percentages. The easier a metric is to measure, the more likely it is that it will be used in practice and that its value will be correctly determined. For all proposed metrics, we define or refer to a measurement methodology (a.k.a. benchmarking approach) and demonstrate their applicability. It is out of scope of this work and part of our future agenda to support the development of standardized measurement approaches to render the metric independent and repeatably measurable. A metric's measurement approach is independent if its definition complete and behavior cannot be influenced by proprietary interests of different vendors or manufacturers aiming to gain competitive advantage by measuring the metric in a way that favors their products or services. As our proposed metrics come with a mathematical definition and a measurement methodology, we claim at this point of research, that it should be possible to verify that a measurement was conducted according to given run-rules. Still, there could be ways to manipulate the results in a way we are not aware of yet. Repeatability of metric measurements

[5]https://aws.amazon.com/ec2/sla/, https://cloud.google.com/compute/sla, and https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_6/.

implies that if the metric is measured multiple times using the same procedure, the same value is measured. In practice, small differences are usually acceptable; however, ideally, a metric's measurement approach should be deterministic when measured multiple times. In this work, we can only partially demonstrate that repeatability is possible in a controlled experiment environment and to a certain degree in the public cloud domain.

8 CHALLENGES

In this section, we first discuss general challenges of conducting performance experiments in cloud environments, before we focus on the emerging challenges for metrics and measurement methodologies in the context of cloud microservice technologies.

8.1 On Challenges of Performance Experiments in Clouds

Obtaining performance metrics from cloud computing environments requires running experiments. Unfortunately, measured performance can vary significantly during the execution of an individual experiment and from one execution to the next [38, 47, 62]. These variations may be due to the use of different physical machines with different performance characteristics (e.g., faster processors) or to interference caused by other VMs that may be using the same shared resources (e.g., disks, processors, or networks). As a result, significant care is required to obtain valid and meaningful measures of performance. Previous work [1] has shown that commonly used approaches to conducting experiments in clouds can lead to unfair comparisons and invalid conclusions. They propose the use of a methodology called randomized multiple interleaved trails (RMIT) in which competing alternatives that are being compared are interleaved in a round robin fashion. For example, the first round might run alternatives A, B, and then C and then the order is re-randomized for each subsequent round so the second round might run B, C, and then A. Clearly, because of variability, multiple executions will be required to obtain metrics and at least some basic statistical analysis, such as computing and reporting means and confidence intervals, is required.

The reliability of the resulting measures, when based on system-level metrics like average resource utilization, pose another challenge for conducting performance experiments. The dynamic mapping relationships of resource activities between the virtual and the physical layers makes the profiling of system level metrics difficult. This is exacerbated by cross-resource utilization relationships (e.g., due to I/O scheduling on hypervisor level) and resource multiplexing across different VMs running on the same, possibly overbooked hardware [52]. For comparisons, the position and type of measurement sensors need to be well-defined and consistent across experiments.

8.2 Emerging Challenges

In the recent years, the industry has increasingly been moving toward microservice, and to serverless or Function-as-a-Service (FaaS) architectures, aiming to take advantage of the flexibility, scalability, and time-to-market in cloud environments [5, 69]. This growing popularity of these microservice and FaaS architectures is introducing new challenges and opportunities to the field of non-functional performance evaluation in cloud computing environments. In this section, potential extensions, assumptions, and changes in applicability of the cloud metrics are discussed to address the concerns and challenges specific to these new types of architectures.

Adaptations for Resource Nesting. The increasing popularity of microservices coincides with technological advances in containerization. In contrast to traditional VM-based virtualization, containerization technology, such as LXC[6] or Docker,[7] offer reduced performance overhead

[6] https://linuxcontainers.org/.
[7] https://www.docker.com/.

by executing applications as isolated processes, commonly referred to as containers, directly on the host operating system [23]. As a result, systems are increasingly deployed as a set of smaller microservices in a container cloud. However, due to the security concerns associated with this relatively immature technology, it is currently a common practice to deploy microservice containers on top of a VM. Moreover, to take advantage of the container-based, environment-agnostic cloud infrastructure, FaaS functions are typically deployed on top of containers [61]. This *resource nesting*, consisting of FaaS functions, containers, VMs, and physical resources, introduces challenges for many existing metrics. Where previously metrics for the evaluation of cloud environments focused on a single virtual layer, VMs, on top of physical hardware, adaptations will need to be considered to ensure quality aspects consider multiple virtual layers. One implication of this increasing resource nesting is that metrics concerned with elasticity typically assume that underlying resources are available and can be requested with a consistent provisioning delay. However, this assumption no longer holds true, scaling higher-level resources depends on the scaling capabilities of the underlying, lower-level resources. To scale up a FaaS function, in an optimistic scenario, the underlying VM and container are already provisioned, leading to a very short provisioning time. However, in alternative scenario's, the VM and the container might need to be scaled first before having sufficient resources for the function to scale. Therefore, performance metrics of systems comprised of these nested resources should not only account for their own performance but also of that of the underlying resources.

Adaptations for Resource Prioritization. One of the important advantages to cloud computing is the notion of "on-demand" resources. This powerful notion has allowed cloud users to avoid large upfront costs and static maintenance costs by only paying the cloud provider for the resources it uses. By effectively managing and multiplexing its resources a cloud provider can make a profit, while reducing the costs for the cloud users. However, offering the same priority for every application might not be the most effective. A background application collecting log data would most likely not have the same urgency or priority as a business-critical web server. Resource cost instead of absolute, immediate performance is the most important factor for a background application, while in the business-critical web server the performance is required at any expense. This scenario is especially true, with regards to microservices, where the diverse set of microservices all have different performance/cost trade-offs. Large providers are experimenting with the notion of spot markets, where cloud users can get resources with fewer performance guarantees at a reduced price. For example, Amazon Spot Instances allows you to bid on spare capacity at their data-centers and in return allow Amazon to pre-empt the resources when the capacity is needed [75]. However, Microsoft Azure and Google Cloud offer a substantial, fixed discount on pre-emptable VMs compared to regular VMs. Yet current metrics focus on optimizing performance, elasticity and other like-wise metrics without a regard for the costs. Though metrics for absolute, functional characteristics, such as performance, will continue to be of value, adaptions will be needed to reflect how well systems can match the expected functional and non-functional objectives. Others have started to investigate this performance/cost trade-off [64], but more research is needed on how to adapt existing metrics and methodologies to take the costs of performance into account.

Adaptations for Resource Fragmentation. The increased adoption of the cloud by various industries comes with a wide variety of different types of workloads, which also introduce more and specific demand for various types of resources [9]. High-performance resources, such as VMs with large memory allocation and CPU shares, or specific resources, such as machines with GPUs, generally cost more relative to less performant machines. This diverse demand requires providers to be flexible in the resource options that they offer. However, this diverse demand does lead to physical resources being sub-optimally utilized, as it leads to *resource fragmentation*. For

example, a single application might take up most of the CPU while under-utilizing the memory. Due to the inability to change resource characteristics on-demand, this prevents any other application being able to make use of the resource. In the context of microservices this resource fragmentation would increase even more as microservices are generally optimized to focus on a similar set of related responsibilities, leading to microservice being a CPU-heavy, I/O-heavy, or network-heavy specific resource consumer. Metrics for elasticity and performance typically consider only a single resource. When they do consider multiple resources, they fail to take into account over-provisioning in multiple resource dimensions. Therefore, adaptations should become part of performance evaluations, to put more emphasis on how effectively multiple resource dimensions are taken into account.

Adaptations for Dynamic Workloads. Current approaches in performance evaluation of cloud applications assume a relatively static workload. However, with the rising popularity of DevOps practices, including automated, frequent deployment of new versions of services is increasingly becoming an industry standard, which is referred to as Continuous Deployment (CD) [28]. According to the 2015 State of DevOps Report [27], a significant portion of organizations already conduct multiple software deployments per day. For example, the retailer Otto conducts more than 500 deployments per week [31]. These rapid changes in the behaviour and resource consumption of these services cause variations in the workload that affect the entire system under test, making it more difficult to predict workloads based on historical data. Accordingly, the methodology will need to be adapted to focus more on evaluating the initial "warm-up time" of mechanisms. The warm-up time is the time needed to obtain a workload characterization needed by a mechanism or policy to perform close to optimally. For example, in auto-scaling research, many policies make use of historical data to evaluate scaling decisions using time series analysis [51]. These policies need historical data to function optimally. With the frequent deployments and changing configurations of services, the size of the workload that needs to be recorded to have a policy function near-optimally becomes a concern. Yet, this concern is not yet represented in existing metrics for cloud environments.

8.3 Discussion

The industry is rapidly transforming to an increasingly dynamic, on-demand model. This trend raises a number of challenges regarding the current cloud metrics and methodology. First, adaptions to existing metrics will need to be considered to address increasing resource nesting. Second, there is an increased focus on applications being cost-effective, meaning that the operational costs should match the desired performance. Besides focusing performance of applications, there should be research done into adaptations to the metrics to consider the performance/cost ratio. Third, adaptations to metrics regarding resource fragmentation should be considered. Finally, these dynamic systems are accompanied by dynamic workloads; workloads of which the characteristics change significantly over time. The methodology of how to evaluate policies, which may depend on long-running static workloads for profiling purposes, will need to reevaluated.

9 RELATED WORK

We identify two distinct and large bodies of related work: one, focusing on the four categories of metrics we address in Sections 3 through 6; the other, general work on cloud performance measurement and assessment. For studies focusing on the four categories of metrics addressed in this work, we have already discussed the similarities and differences with our metrics in the corresponding sections: elasticity (Section 3), performance isolation (Section 4), availability (Section 5), and operational risk (Section 6). Overall, this work summarizes and extends the body of knowledge available for these metrics.

The general work includes much innovative work during the early years, followed since around 2010 by synoptic material. From the early work, innovation has focused on metrics for quantifying overheads [6, 13, 54], basic operational characteristics [56, 74], specific functionality such as scheduling [16, 53, 70] and (emerging topic in clouds) auto-scaling [37], and frameworks for comparing clouds across multiple metrics [72]. We review in the following some important results from the synoptic material, which our study complements with new metrics, new synoptic material, and guidelines of how to measure specific metrics in practice.

Iosup et al. [38] propose a framework and the tools to compare clouds for HPC and many-task workloads. The framework considers metrics for basic operational characteristics, overheads, traditional performance, scalability, and variability, and has been applied to four commercial clouds for a long period. Our study greatly extends the set of earlier metrics.

Garg et al. [29] propose a framework to monitor and rank cloud computing services based on a set of given metrics. Their goal is to allow customers to evaluate and rank cloud offerings based on the provider's ability to meet the user's essential and non-essential requirements, based on metrics for QoS attributes in the SMI framework of the now defunct cloud Service Measurement Index Consoritium (CSMIC) [65]. They frame the problem using multiple criteria decision making (MCDM) and propose a ranking approach based on an Analytic Hierarchy Process (AHP), which reduces bias in decision making. The problem of ranking providers based on multiple metrics is orthogonal to the choice of metrics to use for each non-functional quality aspect of the environment.

Li et al. [50] collect metrics used in prior cloud projects and construct a metrics catalogue. We revise the compiled list and contrast them with our proposed metrics, in Sections 3 through 6.

Becker et al. [8] use the goal question metric (GQM) method to derive metrics for scalability, elasticity and efficiency, with the goal of using these metrics for requirements/SLO engineering. In contrast, our goal is to make various cloud offerings and technologies comparable to each other, and provide a common understanding among all cloud stakeholders.

Aceto et al. and Fatema et al. [2, 22] survey existing cloud monitoring tools, identifying requirements, strengths and weaknesses of these tools. Our study extends this direction by analyzing how to measure specific metrics, thus providing the conceptual tools to assess whether the metrics reported by existing tools and benchmarks are appropriate and complete. Moreover, the metrics we propose in this work could be added to the existing tools in the future.

10 CONCLUSION

Because cloud computing services, and the systems underlying them, already account for a large fraction of the information and communication technology (ICT) market, understanding their non-functional properties is increasingly important. Building standardized cloud benchmarks for these tasks could lead to better system design, tuning opportunities, and eased cloud service selection. Responding to this need, we highlight the relevance of non-functional system properties emerging in the context of cloud computing, namely elasticity, performance isolation, availability, and operational risk. We discuss these properties in depth and describe existing or new metrics that can quantify these properties. Thus, for these four properties, we lay a foundation for benchmarking cloud computing settings. Furthermore, we propose a hierarchical taxonomy for cloud-relevant metrics and discuss emerging challenges.

As future activities, we plan to conduct comprehensive real-world experiments that underline the applicability and usefulness of the proposed metrics, also refining and supporting the standardization the corresponding measurement approaches. As a next step, we are working on an extensive review of existing cloud-relevant benchmarks and connected domains like big data, web services and graph processing.

REFERENCES

[1] Ali Abedi and Tim Brecht. 2017. Conducting repeatable experiments in highly variable cloud computing environments. In *Proceedings of ACM/SPEC ICPE*.

[2] Giuseppe Aceto et al. 2013. Cloud monitoring: A survey. *Comput. Netw.* 57, 9 (2013).

[3] Rodrigo F. Almeida, Flávio R. C. Sousa, Sérgio Lifschitz, and Javam C. Machado. 2013. On defining metrics for elasticity of cloud databases. In *Proceedings of the SBBD*. Retrieved from http://sbbd2013.cin.ufpe.br/Proceedings/artigos/sbbd_shp_12.html.

[4] Amazon. 2017. EC2 Compute SLA. Retrieved from http://aws.amazon.com/ec2/sla/.

[5] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Softw.* 33, 3 (2016), 42–52.

[6] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *Proceedings of SOSP*. 164–177.

[7] André Bauer, Johannes Grohmann, Nikolas Herbst, and Samuel Kounev. 2018. On the value of service demand estimation for auto-scaling. In *Proceedings of GI/ITG MMB*. Springer.

[8] Matthias Becker et al. 2015. Systematically deriving quality metrics for cloud computing systems. In *Proceedings of ACM/SPEC ICPE*.

[9] David Bernstein. 2014. Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Comput.* 1, 3 (2014), 81–84.

[10] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. 2009. How is the weather tomorrow?: Towards a benchmark for the cloud. In *Proceedings of DBTest*. ACM, New York, NY. DOI: http://dx.doi.org/10.1145/1594156.1594168

[11] M. Boniface, B. Nasser, et al. 2010. Platform-as-a-service architecture for real-time quality of service management in clouds. In *Proceedings of ICIW*. 155–160. DOI: http://dx.doi.org/10.1109/ICIW.2010.91

[12] Dean Chandler et al. 2012. *Report on Cloud Computing to the OSG Steering Committee*. Technical Report. Retrieved from http://www.spec.org/osgcloud/docs/osgcloudwgreport20120410.pdf.

[13] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. 2004. Xen and the art of repeated research. In *Proceedings of USENIX ATC*. 135–144.

[14] CloudSleuth. 2017. CloudSleuth monitoring network. Retrieved from https://cloud.spring.io/spring-cloud-sleuth/.

[15] Brian F. Cooper, Adam, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of ACM SoCC*. ACM, New York, NY, 143–154. DOI: http://dx.doi.org/10.1145/1807128.1807152

[16] Marcos Dias de Assuncao, Alexandre di Costanzo, and Rajkumar Buyya. 2009. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In *Proceedings of HPDC*. ACM, New York, NY, 141–150.

[17] Karim Djemame et al. 2006. Introducing risk management into the grid. In *Proceedings of 2nd IEEE International Conference on e-Science and Grid Computing*. IEEE, 28–28.

[18] Thibault Dory et al. 2011. Measuring elasticity for cloud databases. In *Proceedings of ACM/IEEE CCGrid*. Retrieved from http://www.info.ucl.ac.be/ pvr/CC2011elasticityCRfinal.pdf.

[19] Leticia Duboc, David Rosenblum, and Tony Wicks. 2007. A framework for characterization and analysis of software system scalability. In *Proceedings of ACM SIGSOFT ESEC-FSE*. ACM, 375–384. DOI: http://dx.doi.org/10.1145/1287624.1287679

[20] European Commission. 2014. Uptake of Cloud in Europe. Final Report. Digital Agenda for Europe report. Publications Office of the European Union, Luxembourg.

[21] Athanasia Evangelinou, Michele Ciavotta, Danilo Ardagna, Aliki Kopaneli, George Kousiouris, and Theodora Varvarigou. 2016. Enterprise applications cloud rightsizing through a joint benchmarking and optimization approach. *Elsevier Future Generation Computer Systems* 78 (2018), 102–114. DOI: http://dx.doi.org/10.1016/j.future.2016.11.002

[22] Kaniz Fatema et al. 2014. A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives. *J. Parallel Distrib. Comput.* 74, 10 (2014).

[23] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. An updated performance comparison of virtual machines and linux containers. In *Proceedings of IEEE ISPASS*. IEEE, 171–172.

[24] Ana Juan Ferrer, Francisco Hernández, et al. 2012. OPTIMIS: A holistic approach to cloud service provisioning. *Elsevier Future Generation Computer Systems* 28, 1 (2012), 66–77.

[25] Philip J. Fleming and John J. Wallace. 1986. How not to lie with statistics: The correct way to summarize benchmark results. *Commun. ACM* 29, 3 (Mar. 1986), 218–221. DOI: http://dx.doi.org/10.1145/5666.5673

[26] Enno Folkerts, Alexander Alexandrov, Kai Sachs, Alexandru Iosup, Volker Markl, and Cafer Tosun. 2012. Benchmarking in the cloud: What it should, can, and cannot be. In *Selected Topics in Performance Evaluation and Benchmarking*. LNCS, Vol. 7755.

[27] N. Forsgren Velasquez et al. 2015. State of DevOps report 2015. *Puppet Labs IT Revolut.* (2015). https://puppet.com/resources/whitepaper/2015-state-devops-report.
[28] Martin Fowler. 2013. Continuous delivery. Retrieved from https://martinfowler.com/bliki/ContinuousDelivery.html.
[29] Saurabh Kumar Garg et al. 2013. A framework for ranking of cloud computing services. *Elsevier FGCS* 29, 4 (2013).
[30] Google. Compute level SLA. Retrieved from https://cloud.google.com/compute/sla.
[31] W. Hasselbring and G. Steinacker. 2017. Microservice architectures for scalability, agility and reliability in E-commerce. In *Proceedings of IEEE ICSA Workshops*. IEEE, 243–246.
[32] Nikolas Herbst, Samuel Kounev, and Ralf Reussner. 2013. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of USENIX ICAC*. USENIX. Retrieved from https://www.usenix.org/conference/icac13/elasticity-cloud-computing-what-it-and-what-it-not.
[33] Nikolas Herbst, Samuel Kounev, Andreas Weber, and Henning Groenda. 2015. BUNGEE: An elasticity benchmark for self-adaptive IaaS cloud environments. In *Proceedings of SEAMS*. IEEE Press, Piscataway, NJ, 46–56. Retrieved from http://dl.acm.org/citation.cfm?id=2821357.2821366.
[34] Karl Huppler. 2009. *Performance Evaluation and Benchmarking*. Springer-Verlag, Berlin, 18–30. DOI: http://dx.doi.org/10.1007/978-3-642-10424-4_3
[35] Karl Huppler. 2012. Benchmarking with your head in the cloud. In *Topics in Performance Evaluation, Measurement and Characterization*, Raghunath Nambiar and Meikel Poess (Eds.). Lecture Notes in Computer Science, Vol. 7144. Springer, Berlin, 97–110. DOI: http://dx.doi.org/10.1007/978-3-642-32627-1_7
[36] IDC. 2016. *Worldwide and Regional Public IT Cloud Services: 2016–2020 Forecast*. IDC Tech Report. Retrieved from http://www.idc.com/getdoc.jsp?containerId=US40739016.
[37] Alexey Ilyushkin et al. 2017. An experimental performance evaluation of autoscaling policies for complex workflows. In *Proceedings of ACM/SPEC ICPE*. ACM, 75–86.
[38] Alexandru Iosup, Simon Ostermann, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick H. J. Epema. 2011. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE TPDS* 22, 6 (2011), 931–945. DOI: http://dx.doi.org/10.1109/TPDS.2011.66
[39] C. Isci, J. E. Hanson, I. Whalley, M. Steinder, and J. O. Kephart. 2010. Runtime demand estimation for effective dynamic resource management. In *Proceedings of IEEE Network Operations and Management Symposium (NOMS'10)*. IEEE, 381–388. DOI: http://dx.doi.org/10.1109/NOMS.2010.5488495
[40] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. 2012. How a consumer can measure elasticity for cloud platforms. In *Proceedings of ACM/SPEC ICPE 2012*. ACM, New York, NY, 85–96. DOI: http://dx.doi.org/10.1145/2188286.2188301
[41] Brendan Jennings and Rolf Stadler. 2015. Resource management in clouds: Survey and research challenges. *Springer Journal of Network and Systems Management* 23, 3 (2015), 567–619. DOI: http://dx.doi.org/10.1007/s10922-014-9307-7
[42] Prasad Jogalekar and Murray Woodside. 2000. Evaluating the scalability of distributed systems. *IEEE TPDS* 11 (2000), 589–603.
[43] G. Kousiouris, D. Kyriazis, S. Gogouvitis, A. Menychtas, K. Konstanteli, and T. Varvarigou. 2011. Translation of application-level terms to resource-level attributes across the cloud stack layers. In *Proceedings of IEEE ISCC*. 153–160. DOI: http://dx.doi.org/10.1109/ISCC.2011.5984009
[44] George Kousiouris et al. 2014. A multi-cloud framework for measuring and describing performance aspects of cloud services across different application types. In *Proceedings of MultiCloud*.
[45] Rouven Krebs, Christof Momm, and Samuel Kounev. 2014. Metrics and techniques for quantifying performance isolation in cloud environments. *Elsevier SciCo* Vol. 90, Part B (2014), 116–134.
[46] Michael Kuperberg et al. 2011. *Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms*. Technical Report. KIT, Germany. Retrieved from http://digbib.ubka.uni-karlsruhe.de/volltexte/1000023476.
[47] Philipp Leitner and Jürgen Cito. 2016. Patterns in the chaos—A study of performance variation and predictability in public IaaS clouds. *ACM Trans. Internet Technol.* 16, 3 (2016).
[48] Veronika Lesch, André Bauer, Nikolas Herbst, and Samuel Kounev. 2018. FOX: Cost-awareness for autonomic resource management in public clouds. In *Proceedings of ACM/SPEC ICPE*. ACM, New York, NY, 12. DOI: http://dx.doi.org/10.1145/3184407.3184415
[49] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing public cloud providers. In *Proceedings of ACM SIGCOMM IMC*. ACM, New York, NY, 1–14. DOI: http://dx.doi.org/10.1145/1879141.1879143
[50] Zheng Li, L. O'Brien, He Zhang, and R. Cai. 2012. On a catalogue of metrics for evaluating commercial cloud services. In *Proceedings of ACM/IEEE CCGrid*. 164–173. DOI: http://dx.doi.org/10.1109/Grid.2012.15
[51] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *J. Grid Comput.* 12, 4 (2014), 559–592.
[52] Lei Lu et al. Untangling mixed information to calibrate resource utilization in virtual machines. In *Proceedings of ACM ICAC*. ACM, New York, NY, 151–160. DOI: http://dx.doi.org/10.1145/1998582.1998606
[53] Ming Mao, Jie Li, and M. Humphrey. 2010. Cloud auto-scaling with deadline and budget constraints. In *Proceedings of ACM/IEEE CCGrid*.

[54] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. John Janakiraman, and Willy Zwaenepoel. 2005. Diagnosing performance overheads in the Xen virtual machine environment. In *Proceedings of VEE*. 13–23.
[55] Microsoft. 2017. Azure compute level SLA. Retrieved from https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_2/.
[56] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. 2008. Amazon S3 for science grids: A viable solution? In *Proceedings of DADC*. 55–64. DOI: http://dx.doi.org/10.1145/1383519.1383526
[57] Alessandro Vittorio Papadopoulos et al. 2016. PEAS: A performance evaluation framework for auto-scaling strategies in cloud applications. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 1, 4, Article 15 (Aug. 2016). DOI: http://dx.doi.org/10.1145/2930659
[58] Valerio Persico, Pietro Marchetta, Alessio Botta, and Antonio Pescapé. 2015. Measuring network throughput in the cloud: The case of Amazon EC2. *Comput. Netw.* 93 (2015).
[59] V. Persico, P. Marchetta, A. Botta, and A. Pescape. 2015. On network throughput variability in Microsoft azure cloud. In *Proceedings of IEEE GLOBECOM*.
[60] D. C. Plummer et al. 2009. *Study: Five Refining Attributes of Public and Private Cloud Computing*. Technical Report. Gartner.
[61] Mike Roberts. 2016. Serverless architectures. Retrieved from https://martinfowler.com/articles/serverless.html.
[62] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. 2010. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endow.* 3, 1–2 (2010), 460–471.
[63] D. M. Shawky and A. F. Ali. 2012. Defining a measure of cloud computing elasticity. In *Proceedings of ICSCS*. 1–5. DOI: http://dx.doi.org/10.1109/IConSCS.2012.6502449
[64] Siqi Shen, Alexandru Iosup, et al. 2015. An availability-on-demand mechanism for datacenters. In *Proceedings of IEEE/ACM CCGrid*. 495–504.
[65] J. Siegel and J. Perdue. 2012. Cloud services measures for global use: The service measurement index (SMI). In *Proceedings of Annual SRII Global Conference*.
[66] Basem Suleiman. 2012. Elasticity economics of cloud-based applications. In *Proceedings of IEEE SCC*. IEEE Computer Society, Washington, DC, 694–695. DOI: http://dx.doi.org/10.1109/SCC.2012.65
[67] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. 2011. The impact of memory subsystem resource sharing on datacenter applications. *ACM SIGARCH Comput. Architect. News* 39, 3 (2011), 283–294.
[68] Christian Tinnefeld, Daniel Taschik, and Hasso Plattner. 2014. Quantifying the elasticity of a database management system. In *Proceedings of DBKDA*. 125–131. Retrieved from http://www.thinkmind.org/index.php?view=article&articleid=dbkda_2014_5_30_50076.
[69] Erwin van Eyk, Alexandru Iosup, Simon Seif, and Markus Thömmes. 2017. The SPEC cloud group's research vision on FaaS and serverless architectures. In *Proceedings of International Workshop on Serverless Computing*. ACM, 1–4.
[70] David Villegas, Athanasios Antoniou, Seyed Masoud Sadjadi, and Alexandru Iosup. 2012. An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In *Proceedings of ACM/IEEE CCGrid*. 612–619.
[71] Jóakim von Kistowski, Nikolas Herbst, Samuel Kounev, Henning Groenda, Christian Stier, and Sebastian Lehrig. 2017. Modeling and extracting load intensity profiles. *ACM TAAS* 11, 4, Article 23 (Jan. 2017). DOI: http://dx.doi.org/10.1145/3019596
[72] Lei Wang, Jianfeng Zhan, Weisong Shi, Yi Liang, and Lin Yuan. 2009. In cloud, do MTC or HTC service providers benefit from the economies of scale? In *Proceedings of MTAGS, SC Workshops*.
[73] Joe Weinman. 2011. Time is Money: The Value of "On-Demand." Retrieved from http://www.joeweinman.com/resources/Joe_Weinman_Time_Is_Money.pdf.
[74] Nezih Yigitbasi, Alexandru Iosup, Dick H. J. Epema, and Simon Ostermann. 2009. C-meter: A framework for performance analysis of computing clouds. In *Proceedings of ACM/IEEE CCGrid*. 472–477.
[75] Qi Zhang, Quanyan Zhu, and Raouf Boutaba. 2011. Dynamic resource allocation for spot markets in cloud computing environments. In *Proceedings of IEEE UCC*. IEEE, 178–185.
[76] Pin Zhou, Vivek Pandey, Jagadeesan Sundaresan, Anand Raghuraman, Yuanyuan Zhou, and Sanjeev Kumar. 2004. Dynamic tracking of page miss ratio curve for memory management. *ACM SIGOPS Operat. Syst. Rev.* 38, 5 (2004), 177–188.
[77] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. 2010. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the 15th Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems (ASPLOS'10)*. ACM, New York, NY, USA, 129–142. DOI: http://dx.doi.org/10.1145/1736020.1736036

Received September 2017; revised June 2018; accepted July 2018