A

PEAS: A Performance Evaluation Framework for Auto-Scaling Strategies in Cloud Applications

ALESSANDRO VITTORIO PAPADOPOULOS, Lund University
AHMED ALI-ELDIN, Umeå University
KARL-ERIK ÅRZÉN, Lund University
JOHAN TORDSSON, Umeå University
ERIK ELMROTH, Umeå University

Numerous auto-scaling strategies have been proposed in the last few years for improving various Quality of Service (QoS) indicators of cloud applications, e.g., response time and throughput, by adapting the amount of resources assigned to the application to meet the workload demand. However, the evaluation of a proposed auto-scaler is usually achieved through experiments under specific conditions, and seldom includes extensive testing to account for uncertainties in the workloads, and unexpected behaviors of the system. These tests by no means can provide guarantees about the behavior of the system in general conditions. In this paper, we present PEAS, a Performance Evaluation framework for Auto-Scaling strategies in the presence of uncertainties. The evaluation is formulated as a *chance constrained optimization problem*, which is solved using *scenario theory*. The adoption of such a technique allows one to give probabilistic guarantees of the obtainable performance. Six different auto-scaling strategies have been selected from the literature for extensive test evaluation, and compared using the proposed framework. We build a discrete event simulator and parameterize it based on real experiments. Using the simulator, each auto-scaler's performance is evaluated using 796 distinct real workload traces from projects hosted on the Wikimedia foundations' servers, and their performance is compared using PEAS. The evaluation is carried out using different performance metrics, highlighting the flexibility of the framework, while providing probabilistic bounds on the evaluation and the performance of the algorithms. Our results highlight the problem of generalizing the conclusions of the original published studies and show that based on the evaluation criteria, a controller can be shown to be better than other controllers.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: distributed systems; D.2.8 [**Software Engineering**]: Metrics—*performance measures*; G.1.6 [**Mathematics of computing**]: Optimization—*stochastic programming*; H.4 [**Information Systems Applications**]: miscellaneous; K.6.4 [**Management of Computing and Information Systems**]: System Management

General Terms: Performance, Reliability, Theory

Additional Key Words and Phrases: Performance evaluation, Auto-scaling, Randomized optimization, Elasticity, Cloud Computing

1. INTRODUCTION

Elasticity can be defined as the property of a cloud infrastructure (datacenter) or a cloud application to dynamically adjust the amount of allocated resources to meet

This work was partially supported by the Swedish Research Council (VR) for the project "Cloud Control", by the Swedish Government's strategic effort eSSENCE, the European Union's Seventh Framework Programme under grant agreement 610711 for the project CACTOS, and through the LCCC Linnaeus and ELLIIT Excellence Centers.
Author's address: A.V. Papadopoulos, Lund University, Department of Automatic Control, Ole Römers väg 1, 22363 Lund, Sweden; email: alessandro.papadopoulos@control.lth.se; Ahmed Ali-Eldin, Umeå University, Umeå, Sweden; email:ahmeda@cs.umu.se.

DOI: 0000001.0000001

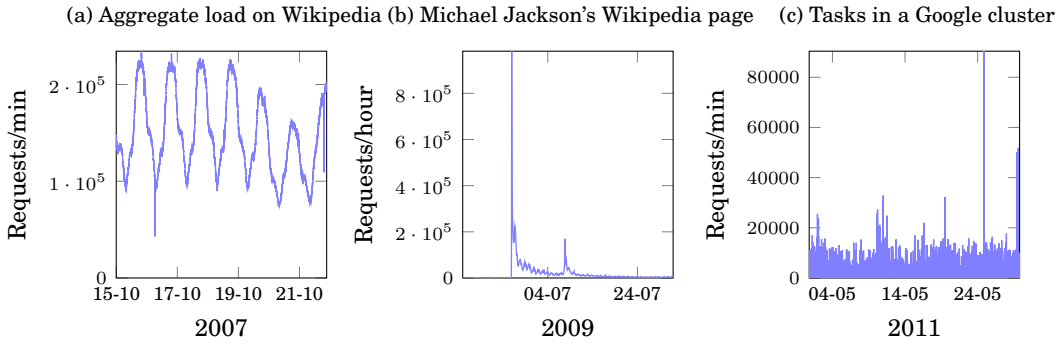

Fig. 1. Different workloads have different burstiness. Plot (a) shows a workload with little burstiness and strong periodicity. Plot (b) shows sudden spikes in a mostly periodic workload due to an event. Plot (c) shows a bursty workload.

changes in workload demands. In principle, resources allocated to a running service should be varied such that the Quality of Service (QoS) requirements are preserved at minimum cost. A large number of auto-scalers have been designed for this purpose, using different approaches such as control theory [Lim et al. 2009], neural networks [Islam et al. 2012], second order regression [Iqbal et al. 2011], histograms [Urgaonkar et al. 2008], time-series models [Herbst et al. 2013], the secant method [Meng et al. 2010], and look-ahead control [Roy et al. 2011]. While very different in nature, all of these controllers have one thing in common: They are designed to provision resources according to the changing workloads of the applications deployed in a cloud.

Cloud (and datacenter) workloads may have very different characteristics. They vary in mean, variance, burstiness, periodicity, type of required resources, and many other aspects. When studying workload periodicity and burstiness profiles, one can find workloads that have repetitive patterns with daily, weekly, monthly and/or annual cycles, e.g., the Wikipedia workload shown in Figure 1(a) has diurnal patterns with small deviations in the pattern from one day to the other [Ali-Eldin et al. 2014a]. Other workloads may have uncorrelated spikes and bursts that occur due to an unusual event, e.g., Figure 1(b) shows the load on the Wikipedia page about Michael Jackson with two significant spikes that occurred immediately after his death and during his memorial service. On the other hand, some workloads have weak or no recognizable patterns at all, e.g., the workload shown in Figure 1(c) for tasks submitted to a Google cluster [Wilkes 2011]. Most often, the characteristics of the workloads of new applications are unknown in advance.

Unfortunately, most of the state-of-the-art auto-scalers have been evaluated using less than three real workload traces – in a real deployment or in simulation – which makes it impossible to generalize the obtained results [Lim et al. 2009; Urgaonkar et al. 2008; Iqbal et al. 2011; Singh et al. 2010; Herbst et al. 2013]. In addition, the traces used for testing are typically short spanning a few minutes, hours or days [Urgaonkar et al. 2008; Roy et al. 2011; Herbst et al. 2013]. As the workload profile is well known to affect the performance of a controller and the running application, it is not possible to tell whether one auto-scaler is better than the other for a certain workload [Almeida Morais et al. 2013; Feitelson 2014].

This paper presents PEAS (Performance Evaluation framework for Auto-Scaling strategies), a framework for the evaluation of auto-scaling techniques. PEAS reformulates the performance evaluation problem into a chance constrained optimization

problem, whose solution requires an extensive yet targeted testing of the auto-scaling techniques. The solution of the chance constrained optimization is carried out through *scenario theory* [Calafiore and Campi 2005; Campi and Garatti 2011; Nemirovski and Shapiro 2006], which has been recently applied to control theory problems related to complex, stochastic, or uncertain systems [Calafiore and Campi 2006; Campi et al. 2009; Papadopoulos and Prandini 2014; Papadopoulos et al. 2016]. With the adoption of the scenario theory, PEAS can be used to extensively evaluate the performance of different auto-scaling strategies, providing probabilistic guarantees on the obtained results.

To show the feasibility of using scenario theory to compare auto-scalers, we obtained the implementations of three state-of-the-art auto-scalers from their designers [Nguyen et al. 2013; Fernandez et al. 2014; Ali-Eldin et al. 2012]. In addition, we reimplemented three other auto-scalers from the literature [Urgaonkar et al. 2008; Chieu et al. 2009; Iqbal et al. 2011]. We used a publicly available workload from the Wikimedia foundation spanning roughly six years [Mituzas 2007] to evaluate the six algorithms. The workload traces are composed of more than 3500 streams comprising all requests to all Wikimedia foundation's projects with all 287 supported languages, of which a subset of 796 streams is selected.

In summary, the contribution of this work is twofold. First, we introduce a framework that provides probabilistic guarantees on the QoS achieved by auto-scalers while permitting to compare their performance in worst-case scenarios. Secondly, we compare six state-of-the-art auto-scalers using a large real workload spanning 6 years from a production system. Thirdly, in order to allow the validation and replication of our experiments, we have open-sourced the implementation of five of the algorithms along with the simulation code, and the scripts used for downloading, and analyzing the workloads. [1]

The rest of the paper is organized as follows. Section 2 discusses the problem of evaluating different auto-scalers and comparing them along with the tradeoffs of choosing one auto-scaler over another. Section 3 discusses some of the auto-scaling techniques that have been proposed in the literature. We give special attention to the six algorithms we evaluate with the proposed framework. Section 4 introduces PEAS, focusing on its theoretical aspects. Section 5 discusses the experimental results. We conclude in Section 6.

2. THE CASE FOR PEAS

While cloud computing is a relatively new technology that was enabled recently by the advances in virtualization; elasticity control and auto-scaling of cloud resources is an incarnation of the dynamic resource provisioning problems in clusters, grids and datacenters. The dynamic resource provisioning problem – and its variants – have been discussed over the past two decades with initially some patents in the early 1990s [Liu and Silvester 1991; Dong and Treiber 1992], followed by an interest from the research community later in the late 1990s and early 2000s [Chase et al. 2001; Vahdat et al. 1998]. During this period, many algorithms have been proposed for dynamic provisioning with different flavors before and after the term cloud computing was coined [Urgaonkar et al. 2008; Chieu et al. 2009; Gandhi et al. 2012; Fernandez et al. 2014]. Since then, many new auto-scaling techniques have been proposed.

[1] We have open sourced all the algorithms except for AGILE as we do not have the permission of its authors. The codes are now available on a public repository with an obfuscated name. If the manuscript is accepted we will copy it to a publicly accessible repository with the authors name. Link to the repository: https://bitbucket.org/snippets/sigmetricsreview/

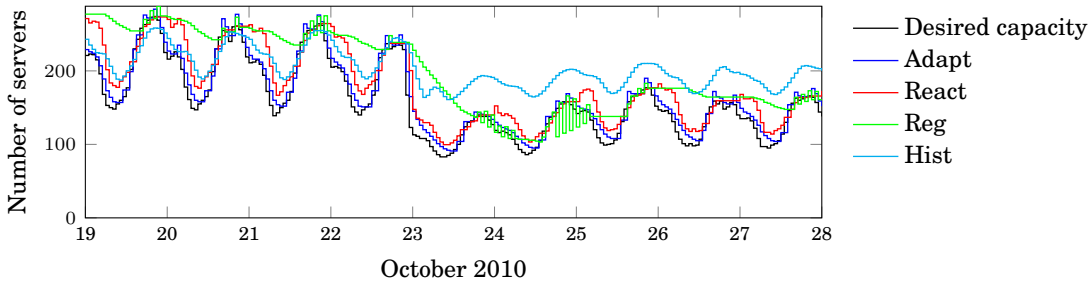

Fig. 2. There are tradeoffs when choosing which auto-scaler to use in order to control server provisioning.

As for the performance assessment of auto-scalers, many published solutions have not been compared to previously proposed ones so far, but have been rather compared with a predefined required response time for the tested application or against static provisioning [Iqbal et al. 2011; Al-Shishtawy and Vlassov 2013; Chieu et al. 2009; Lim et al. 2010; Roy et al. 2011; Herbst et al. 2013]. In addition, many of the published work have not been tested with more than one real workload [Herbst et al. 2013; Al-Shishtawy and Vlassov 2013; Fernandez et al. 2014; Islam et al. 2012; Mao et al. 2010; Meng et al. 2010; Singh et al. 2010; Lama and Zhou 2012; Mahmud et al. 2014]. Even when algorithms are tested with more than one workload, many of these workloads are typically quite old, from systems and services that are not even used today [Gandhi et al. 2012; Gong et al. 2010; Ali-Eldin et al. 2012]. While all published previous work show that "the proposed algorithm" works in certain scenarios, the results in many of these papers cannot be generalized.

Many proposed state-of-the-art auto-scalers use the desired response time or service latency as a reference signal to a controller. A proposed approach then works if it meets the experiments' required response times, mostly set by the authors based on some QoS studies. On the other hand, response time does not show under- and over-provisioning, i.e., if the allocated capacity is too low or too high with respect to the actually needed capacity to serve the incoming requests within the desired response time. Under- and over-provisioning are indeed extremely important to compare the performance of different controllers. Two controllers might be achieving the required response time while one of them uses, for example, on average only one tenth of the resources used by the other controller. Similarly, one controller can be dropping, on average, fewer requests than another one [Gandhi et al. 2012]. In addition, latencies are in general nonlinear with respect to the provisioned resources. To give an example, suppose there are two workloads running on two different machines and each of them experiences an average service latency of 50ms for a request. If the required QoS is to keep the average service latency below 100ms, can then the two workloads be handled by a single server? The answer is likely to be no, since service latencies do not depend linearly on the amount of resources provisioned. This nonlinear relationship between resources provisioned and response time has been observed in large-scale distributed systems [Dean and Barroso 2013]. In addition, latency measurements are often extremely noisy with very high variance, even for constant workloads, [Bodik et al. 2010; Bodik 2010; Wang et al. 2013]. To conclude, request response times and latencies are by themselves not suitable metrics to compare auto-scalers [Bodik 2010].

There are tradeoffs between different auto-scalers in terms of over-provisioning, under-provisioning, and the time and resources required to compute the required capacity. Figure 2 shows the output for four different controllers, [Ali-Eldin et al. 2012]

in blue, [Chieu et al. 2009] in red, [Iqbal et al. 2011] in green, and [Urgaonkar et al. 2008] in cyan, when used to predict the number of servers required to serve the load on the English Wikipedia pages between October 19^{th}, 2010 and October 28^{th}, 2010. The figure also shows the theoretical optimal number of servers required to serve such a workload (shown as the black line).[2] The first peaks of the load are part of a bursty period. The rest is when the load begins to settle down. One can see that during this transient period, the four auto-scalers show different behaviors. The behavior does not just depend on the selected auto-scaler, but also on how they are parameterized. The same auto-scaler with another set of parameters will typically behave in a different way.

2.1. Novelty of PEAS

In this work, we have been greatly inspired by the seminal work by Keogh and Kasetty [2003] which shows that many of the proposed algorithms by the time series data mining community are of very little use due to lack of testing on enough datasets or due to hidden parametrization pitfalls. Our main motivation for PEAS is to help organize the design space of auto-scalers. We believe that PEAS will help the research community to establish a theoretical framework enabling the possibility of comparing in a systematic way different auto-scalers while providing probabilistic guarantees on their performance. Such guarantees allow service providers to choose a suitable auto-scaler that fulfills a desired QoS metric for their infrastructure. We envision that such an approach to testing auto-scalers may lead to a more mathematically grounded definition of Service Level Agreements (SLAs), as defined by Sturm et al. [2000], and better design of auto-scalers.

PEAS is well grounded in the theory of stochastic systems, as the framework is based on recent advances in stochastic programming, aimed at solving chance constrained optimization problems. Chance constrained optimization problems arise when one seeks to minimize a convex objective over solutions satisfying, with a given close to one probability, a system of randomly perturbed convex constraints [Nemirovski and Shapiro 2007], as described in Section 4.1.

An exact numerical solution of CCP is in general considered not possible due to both modeling and numerical issues [Prékopa 2003]. CCP is typically NP hard but it is considered as a very useful tool for robust and stochastic control, as solving chance constraint optimization problems pursue a different probabilistic approach to robustness in control problems [Calafiore and Campi 2006]. Roughly speaking, the scenario approach approximates the solution of a CCP with the solution of a convex optimization problem with a finite number of sampled instances of the uncertain quantities (the scenarios). This approach basically reduces the problem from searching an infinite space of possibilities to a much smaller tractable problem that gives guarantees in probability.

The approach is well suited for server systems, where giving probabilistic guarantees on the QoS of a system is important to service providers. From a theoretic point of view, the novelty of PEAS is in transforming the auto-scaling performance evaluation into a chance constraint optimization problem, which is solved using the scenario approach. To the best of the authors' knowledge, PEAS is the only framework available that provides theoretical bounds, thorough analysis, and hard guarantees on the performance of auto-scalers in cloud systems.

[2]Section 4.2.1 explain in more details how the optimal and the predicted number of servers are obtained.

3. RELATED WORK

3.1. Selected auto-scaling methods

A significant number of auto-scalers have been proposed in the literature [Lorido-Botrán et al. 2014]. We choose six auto-scalers to extensively compare in this work. These six are in our opinion a representative sample of the state-of-the-art. The selected methods have been published in the following years 2008 [Urgaonkar et al. 2008] (with an earlier version published in 2005 [Urgaonkar et al. 2005]), 2009 [Chieu et al. 2009], 2011 [Iqbal et al. 2011], 2012 [Ali-Eldin et al. 2012], 2013 [Nguyen et al. 2013] and 2014 [Fernandez et al. 2014]. They thus represent the development of the field over a period of more than 7 years. The selected methods cover all major time-series analysis methodologies. An extensive survey on auto-scaling strategies can be found in [Lorido-Botrán et al. 2014], and papers quoted therein.

(1) Urgaonkar et al. [2008] propose a provisioning technique for multi-tier Internet applications. The proposed methodology adopts a queuing model to determine how many resources to allocate in each tier of the application. A predictive technique based on building **Hist**ograms of historical request arrival rates is used to determine the amount of resources to provision at an hourly time scale. Reactive provisioning is used to correct errors in the long-term predictions or to react to unanticipated flash crowds. The authors also propose a novel datacenter architecture that uses Virtual Machine (VM) monitors to reduce provisioning overheads. The technique is shown to be able to improve responsiveness of the system, also in the case of a flash crowd. We refer to this technique as **Hist**. The authors test their approach using two open source applications, RUBIS which is an implementation of the core functionality of an auctioning site, and Rubbos a bulletin-board application modeled after an online news forum. The testing is performed using 13 VMs running on Xen. Traces from the FIFA 1998 worldcup servers are scaled in time and intensity and used in the experiments. We have implemented this auto-scaler for our experiments.
(2) Chieu et al. [2009] present a dynamic scaling algorithm for automated provisioning of VM resources based on the number of concurrent users, the number of active connections, the number of requests per second, and the average response time per request. The algorithm first determines the current web application instances with active sessions above or below a given utilization. If the number of overloaded instances is greater than a predefined threshold, new web application instances are provisioned, started, and then added to the front-end load-balancer. If two instances are underutilized with at least one instance having no active session, the idle instance is removed from the load-balancer and shutdown from the system. In each case the technique **React**s to the workload change. For the rest of the paper, we refer to this technique as **React**. The authors introduce the scaling algorithm but provide no experiments to show the performance of the proposed auto-scaler. The main reason we are including this algorithm in the analysis is that this algorithm is the baseline algorithm in our opinion since it is one of the simplest possible workload predictors. We have implemented this auto-scaler for our experiments.
(3) Iqbal et al. [2011] propose a methodology for automatic detection of bottlenecks in a multi-tier web application targeting maximum response time requirements. The proposed technique uses a reactive approach for scaling up resources, and a **Reg**ression model for predicting the amount of resources needed for the actual workload, and possibly retract over-provisioned resources. The technique is tested using RUBIS running on seven physical machines using a EUCALYPTUS cloud installation. httperf is used to generate a workload that increases and decreases in predefined steps. The results are compared to static provisioning. We refer to this technique as **Reg**. We have implemented this auto-scaler for our experiments.

(4) Ali-Eldin et al. [2012] propose an autonomous elasticity controller that changes the number of VMs allocated to a service based on both monitored load changes and predictions of future load. The predictions are based on the rate of change of the request arrival rate, i.e., the slope of the workload, and aims at detecting the envelope of the workload. The designed controller **Adapt**s to sudden load changes and prevents premature release of resources, reducing oscillations in the resource provisioning. **Adapt** tries to improve the performance in terms of number of delayed requests, and the average number of queued requests, at the cost of some resource over-provisioning. The algorithm was tested using a simulated environment using a non-scaled version of the FIFA 1998 world cup server traces, traces from a Google cluster and traces from Wikipedia. We refer to this technique as **Adapt**. We obtained the original code from the authors for our experiments.
(5) Nguyen et al. [2013] propose **AGILE**, a wavelet-based algorithm to provide a medium-term resource demand prediction with a lead time of about 2 minutes, a time that allows AGILE to possibly start up new application server instances before performance falls short. The framework also uses dynamic VM cloning to reduce application startup times. Its accuracy has been proven to be better than previous schemes in terms of true positives and false negatives. AGILE was implemented on top of KVM on a cloud testbed having 10 physical machines. The algorithm was evaluated using RUBIS and the Apache Cassandra key-value store. Four workloads were used in the evaluation, namely, the FIFA World Cup 1998 webserver trace starting at 1998-05-05:00.00; NASA webserver trace beginning at 1995-07-01:00.00; EPA webserver trace starting at 1995-08-29:23.53; and ClarkNet web server trace beginning at 1995-08-28:00.00. The prediction algorithm was also tested using real system resource usage data collected on a Google cluster. The prediction results are compared to two previously published algorithms, autoregression [Buneci and Reed 2008] and PRESS [Gong et al. 2010]. The provisioning results are compared also to a reactive approach and a threshold based approach that increases and decreases the number of machines based on preset resource usage levels. We obtained the original code from the authors for our experiments.
(6) Fernandez et al. [2014] present a methodology that scales a web application in response to changes in throughput at fixed intervals of 10 minutes. The proposed algorithm is based on different components. The **P**rofiler measures the computing capacity of different hardware configurations when running an application. The predictor forecasts the future service demand using standard time-series analysis techniques, e.g., Linear Regression, Auto Regressive Moving Average (ARMA), etc. The dynamic **L**oad **B**alancer distributes the requests in the datacenter according to the capacities of the provisioned resources. Finally, the **S**caler uses the predictor and the profiler to find the scaling plan that fulfills a prescribed Service Level Objective (SLO). For evaluation, the authors test their technique on two cloud infrastructures, a private one (the DAS-4, a multi-cluster system hosted by universities in The Netherlands) and a public one (the Amazon EC2 cloud). The authors deployed MediaWiki application instances on both environments using the WikiBench benchmark to run the application. The benchmark uses a copy of Wikipedia, and replays a fraction of the actual Wikipedia access traces. The experiments used up to 12 EC2 VMs of different types and up to 8 machines of varying configurations in the DAS-4 system. In our work, we refer to this technique as **PLBS**. The code for this auto-scaler is open-sourced. We downloaded the authors' original implementation.

It is worth noticing that since regression models assume that the predicted signal is stationary, large deviations in workloads from stationarity affects the predictions [Hastie et al. 2009; Hamilton 1994]. Similarly, PLBS uses a mixture of statistics

that assume stationarity mostly in the predicted signal [Hastie et al. 2009; Hamilton 1994]. AGILE uses wavelets to predict the workload dynamics. Wavelets are suitable for non-stationary signal estimation, but not for signals with strong non-linearity [Huang et al. 1998].

3.2. Performance evaluation methodologies

Whereas a significant effort was spent by many researchers on designing "efficient" auto-scalers, not as much effort was spent on defining an evaluation methodology to compare the designed algorithms. Typically, the designed auto-scalers are evaluated in an *ad hoc* way, with a limited number of experiments, that may not be necessarily representative of the actual workload of a cloud environment, or more generically, of a service provider.

There is, however, an increasing interest in defining some common practices and methodologies for the evaluation of auto-scaling techniques [Villegas et al. 2012; Iosup 2012; Li et al. 2010]. For example, Ferraris et al. [2012] present the results of stressing the auto-scaling mechanisms implemented by Amazon and Flexiscale cloud providers. Netto et al. [2014] introduce an index for evaluating the performance achieved by an auto-scaling technique. The index is called "Auto-scaling Demand Index" (ADI), and it is used for penalizing differences between the actual and the desired resource utilization levels.

To the best of our knowledge, there is no standard methodology for the evaluation of auto-scaling strategies, that allows also for the definition of probabilistic guarantees on the performance. The evaluation is typically carried out through few experiments, but by no means to quantify the confidence that the obtained results will hold also in other situations. As a result, it is difficult, if not impossible, to use such results for the definition of performance SLAs.

4. PEAS: THE EVALUATION FRAMEWORK

PEAS is a framework that is able to provide probabilistic guarantees on the performance of auto-scaling strategies. The proposed framework is based on recent developments in the control and optimization community, related to the solution of Chance-Constrained optimization Problems (CCPs) via scenario theory [Prékopa 2003; Nemirovski and Shapiro 2006; Calafiore and Campi 2005; Campi and Garatti 2011]. Scenario theory represents a systematic theoretical foundation of the "many-experiments" approach, i.e., when one wants to evaluate the robustness of the system, a certain number of experiments is performed aimed at quantifying the confidence about the quality of the system. The scenario optimization approach has been adopted to tackle several problems in control theory, ranging from systems and control design [Campi et al. 2009], to robust control design [Calafiore and Campi 2006], from model order reduction of stochastic hybrid systems [Papadopoulos and Prandini 2016] to game theory [Bopardikar et al. 2013].

The steps of the framework can be summarized as follows.

(1) Definition of the performance measurements.
(2) Definition of a suitable performance metric.
(3) Formulation of the CCP.
(4) Application of the scenario theory.
   (a) Choice of the parameters.
   (b) Computation of the experimental results.
   (c) Computation of the randomized solution.

The formulation of the performance evaluation problem as a CCP is a nontrivial task, and is one of the contributions of this paper.

4.1. Chance-constraint optimization and scenario theory

We provide a brief overview of CCP and the main results of the scenario theory before going into the details of the framework. Consider the optimization problem

$$\min_{x \in \mathbb{R}^n} \phi(x) \quad \text{subject to: } G(x, \xi) \in C. \tag{1}$$

where $C \in \mathbb{R}^m$ is a closed convex set, and $\phi(x)$ is a real valued function. The mapping $G : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^m$ depends on uncertain parameters ξ which vary in a set $\Xi \in \mathbb{R}^d$. For a fixed ξ the constraint is either satisfied or not. If we consider ξ as a random vector with a probability distribution having support Ξ, finding the optimal value of x would require to have the constraint $G(x, \xi) \in C$ to be satisfied with probability 1. Intuitively, this would mean to have an infinite number of constraints, one for each possible realization of ξ. This is known in the stochastic programming literature as the second stage feasibility problem that has to be solvable with probability 1 [Prékopa 2003]. However, this is too conservative in most practical situations. A more realistic requirement is to ensure feasibility with probability close to 1, i.e., at least $1-\epsilon$. Hence, we can formulate problem (1) as a Chance-Constrained optimization Problem (CCP)

$$CCP : \min_{x \in \mathbb{R}^n} \phi(x) \quad \text{subject to: } \mathbb{P}\{G(x, \xi) \in C\} \geq 1 - \epsilon. \tag{2}$$

CCPs have been widely studied in the literature, and are known to be NP-hard [Prékopa 2003; Nemirovski and Shapiro 2006], thus only approximate solutions can be found.

Scenario theory or the "scenario optimization approach" is a technique for obtaining approximate solutions to robust optimization and CCPs based on randomization of the constraints [Nemirovski and Shapiro 2006; Calafiore and Campi 2005; Campi and Garatti 2011]. Scenario theory enables one to decide what is the level of confidence required, i.e., the value of the parameter ϵ in the CCP (2), and it tells how many experiments one needs to perform in order to find a feasible estimate solution to the problem. The chance constraints are then replaced with the realizations $G(x, \xi^{(i)}) \in C$, with $\xi^{(i)}$, $i = 1, \ldots, N$, and thus an approximate solution can be easily found.

4.2. Performance evaluation

This section presents the theoretical foundations of PEAS. The framework is based on the idea that one can re-formulate the problem of auto-scalers performance evaluation as a CCP, that can be solved using scenario theory. The framework assumes that the arrival rate λ (measured in request per time unit) is stochastic and the goal of the auto-scaling strategy is to provision the least amount of resources able to serve the incoming traffic with a required QoS. The proposed method involves using as input of the different auto-scaling strategies some realizations of the stochastic input. In practice, this means that either the distribution of the input is known, or some of its realizations are available as historical time series. Our goal is to introduce a method for evaluating the performance of different methods with some probability guarantees, in order to understand which method is behaving better especially in critical scenarios.

4.2.1. Definition of the performance measurements. The cloud infrastructure is modeled as a $G/G/N$ stable queue in which the number of servers N is variable [Li and Yang 2000]. This is a generalization of the model proposed by Khazaei et al. where a cloud is modeled as an $M/G/m$ queue with a constant m [Khazaei et al. 2012]. It is assumed that the servers are used to serve a non-stationary request mix with varying request sizes [Singh et al. 2010]. Figure 3 shows the system model.

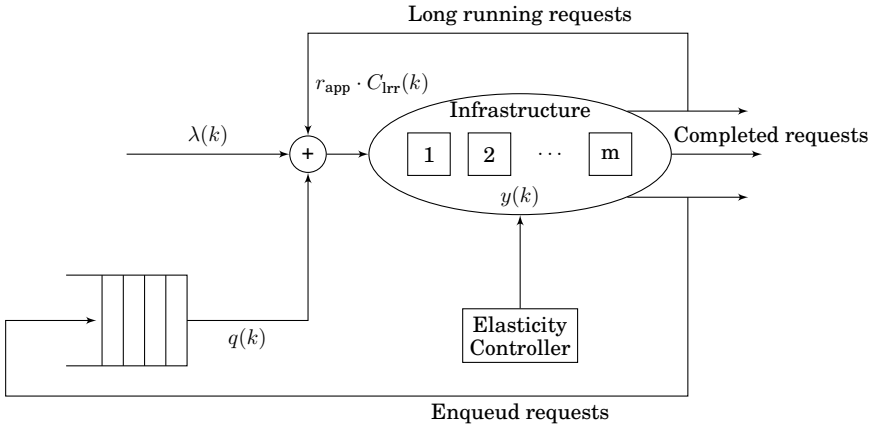

Fig. 3. Queuing model for a service deployed in the cloud.

We denote with $k \in \mathbb{N}$ the discrete (slotted) time, that counts the decisions for the auto-scaler. Let then $y(k)$ be the current capacity allocated to the service at time k, and $y^{\circ}(k)$ the required capacity needed to serve the actual incoming traffic at time k. The required capacity depends on the specific application and workload considered. In determining the necessary capacity $y^{\circ}(\cdot)$, our model accounts for queuing effects, load mix variations, and long running requests, i.e., requests requiring long processing time. The required capacity at any time k is thus the total capacity required to process the newly arriving load $\lambda(k)$, the number of requests that were not served during the past and are queued $q(k)$, and the requests that have already been assigned to a server in a previous time unit, $k-m, \forall m < k$, but are still being processed. The total required capacity at time unit k is then computed as

$$y^{\circ}(k) = \left\lceil \frac{\lambda(k) + q(k)}{r_{\text{app}}} \right\rceil + C_{\text{lrr}}(k), \tag{3}$$

where $q(k)$ is the number of requests that are enqueued in the system and that will be processed in the next time unit, r_{app} is the average number of requests per time unit that a single VM can handle for the specific application with an acceptable service time, and $C_{\text{lrr}}(k)$ is the capacity used for processing the long running requests that require more than one time unit to be served. We denote with $\lceil x \rceil$ the ceiling function that returns the smallest integer number greater than or equal to x. An acceptable service time for a request in this case is the maximum service time that the application can tolerate and that the service user requires. For a web request to a static web-page, an acceptable service time is in seconds or even milliseconds. For a Map-Reduce job, the acceptable service time can be in minutes or even hours. This approach for finding the required capacity is similar to what has been used in the literature [Gandhi et al. 2012; Urgaonkar et al. 2005; Urgaonkar et al. 2008], but it accounts also for queuing effects and long running requests.

Long running requests are any requests that require more than the time interval between two auto-scaler decisions. Since all auto-scaling algorithms calculate the required capacity on discrete intervals, e.g., AGILE and Adapt every time unit based on the sampling interval of the monitoring data, PLBS every 10 minutes, and Hist every hour with corrections on a minutes scale, a long running request is a request that needs to be processed by the system for a period that extends beyond such an interval. The addition of the long running requests effect to the model allows PEAS to take in

to account the effect of workload mix changes [Singh et al. 2010] since a request that sorts a Terabyte of data requires more time than a request that fetches a static web page.

We note that Equation 3 does not represent the state of the system when any of the auto-scaling algorithms is used but rather the requirement to maintain the system with sufficient capacity to process all requests in the minimum required time. An auto-scaling algorithm should at least provision $y^\circ(k)$ resources in order to serve all requests in an acceptable service time. If an auto-scaling algorithm provisions more capacity than $y^\circ(k)$, then it should be able to serve all requests but at a a higher cost. We later discuss how we parameterize our experiments to choose all the parameters in Equation 3 and how we handle the different assumptions in the designs of the different algorithms.

4.2.2. Definition of a suitable performance metric. Having defined the required capacity $y^\circ(t)$, it is intuitive to define a distance of the actual allocated capacity $y(t)$ with respect to the ideal behavior of the system. Therefore, an auto-scaler with minimal distance is better than all the other considered auto-scalers.

In order to appropriately evaluate the performance of an auto-scaling strategy, we then define a distance $d_{\mathcal{T}}(\cdot,\cdot)$ that maps each pair of trajectories $y(k)$, $k \in \mathcal{T}$, and $y^\circ(k)$, $k \in \mathcal{T}$, into a positive real number $d_{\mathcal{T}}(y, y^\circ)$ that represents the extent to which $y(\cdot)$ is far (as defined below) from the ideal behavior $y^\circ(\cdot)$ along the finite time interval $\mathcal{T}$. Note that $d_{\mathcal{T}}(y, y^\circ)$ is a random quantity since it depends on the realization of the stochastic input $\lambda(k)$.

We consider different distance metrics in order to obtain synthetic results both on the overall performance of the algorithms, and of specific aspects that one could be interested in. The first distance metric that we choose is

$$d_{\mathcal{T}}^{\text{norm}}(y, y^\circ) = \frac{1}{|\mathcal{T}|} \sum_{k \in \mathcal{T}} \|y^\circ(k) - y(k)\|^2, \tag{4}$$

where $|\mathcal{T}|$ is the number of samples contained in the time interval $\mathcal{T}$ of the experiment, and $\|X\|^2$ denotes the squared 2-norm of the matrix X. This distance penalizes under-provisioning (i.e., when the difference $y^\circ - y$ is positive) and over-provisioning (i.e., when the difference $y^\circ - y$ is negative) in the same way, accounting also for those situations where the allocated resources are oscillating around the ideal capacity. Notice that the normalization with respect to $|\mathcal{T}|$ is used to account for scenarios with different length. Notice also, that it is possible to modify (4) in order to weight differently under- and over-provisioning. Whereas distance (4) is able to give synthetic information about the overall performance, its value is difficult to interpret. This is the main reason for introducing other more interpretable quantities.

As argued by Abate and Prandini [2011], the directional Hausdorff distance

$$d_{\mathcal{T}}^{\text{haus}}(y, y^\circ) = \sup_{k \in \mathcal{T}} \inf_{\kappa \in \mathcal{T}} \|y^\circ(k) - y(\kappa)\|, \tag{5}$$

is a sensible choice for $d_{\mathcal{T}}(y, y^\circ)$ when performing probabilistic verification such as, e.g., estimating the probability that y will enter some set within the time horizon $\mathcal{T}$. This distance disregards the time spent under- and over-provisioning, thus it cannot be used for the evaluation of auto-scalers. Therefore, one can choose a slightly different distance

$$d_{\mathcal{T}}^{\text{sup}}(y, y^\circ) = \sup_{k \in \mathcal{T}} \|y^\circ(k) - y(k)\|, \tag{6}$$

that accounts also for the maximum discrepancy between the ideal and the actual behavior. This is the second distance that we use for the evaluation.

We here consider also two other metrics that give information about how much the auto-scaling strategy is over- and under-provisioning. These can be thus expressed as

$$d_{\mathcal{T}}^{\text{over}}(y, y^\circ) = \sup_{k \in \mathcal{T}} \| \max\{y(k) - y^\circ(k), 0\} \|, \tag{7}$$

$$d_{\mathcal{T}}^{\text{under}}(y, y^\circ) = \sup_{k \in \mathcal{T}} \| \max\{y^\circ(k) - y(k), 0\} \|. \tag{8}$$

However, it is also desirable that $y(\cdot)$ converges towards $y^\circ(\cdot)$ as soon as possible, thus producing an error converging to zero. This is extremely important from the service provider perspective, since excessive over-provisioning for long time results in a waste of resources, while excessive under-provisioning results in loss of revenue. In order to account for the time aspect, we introduce two other metrics

$$d_{\mathcal{T}}^{\text{overT}}(y, y^\circ) = \frac{1}{|\mathcal{T}|} \sum_{k \in \mathcal{T}} \| \max\{y(k) - y^\circ(k), 0\} \| \Delta k, \tag{9}$$

$$d_{\mathcal{T}}^{\text{underT}}(y, y^\circ) = \frac{1}{|\mathcal{T}|} \sum_{k \in \mathcal{T}} \| \max\{y^\circ(k) - y(k), 0\} \| \Delta k. \tag{10}$$

where Δk is the time interval between two subsequent interventions of the auto-scaling strategy. The interpretation of these two distance is thus, the average over- and under-provisioning in a time unit.

The last distance the framework uses is an adapted version of the Auto-scaling Demand Index (ADI) proposed by Netto et al. [2014]. The utilization level can be defined as

$$u(k) = \frac{y(k)}{y^\circ(k)}. \tag{11}$$

Notice that $u(k) \geq 0$, $\forall k \in \mathbb{N}$ but unlike the ADI measure proposed by Netto et al. [2014], $u(k)$ can be larger than 1, representing a situation of over-provisioning. The adapted ADI is represented by σ, and defined as

$$\sigma = \sum_{k \in \mathcal{T}} \sigma(k), \tag{12}$$

where

$$\sigma(k) = \begin{cases} L - u(k) & \text{if } u(k) \leq L, \\ u(k) - U & \text{if } u(k) \geq U, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

with $0 \leq L \leq U$ representing the required lower and upper bound of the utilization. The adapted ADI provides information analogous to (4), as it is evaluating over time the discrepancy between the actual and required capacity. In the following, we set $L = 0.9$ and $U = 1.1$, meaning that we are not penalizing strategies that oscillate in an interval $(L \cdot y^\circ(k), U \cdot y^\circ(k))$. Since ADI penalizes longer workload traces, for a fair comparison we consider its normalized version with respect to the length of the time horizon $|\mathcal{T}|$,

$$\sigma_{\mathcal{T}} = \frac{\sigma}{|\mathcal{T}|} = \frac{1}{|\mathcal{T}|} \sum_{k \in \mathcal{T}} \sigma(k). \tag{14}$$

4.2.3. Formulation of the CCP. Since both the required capacity $y^\circ(t)$ and the the actual allocated capacity $y(t)$ depend on the realization of different stochastic quantities, it is not easy to find what is the minimum distance with respect to all the possible

workload realizations for all the possible auto-scalers. The problem of evaluating the performance of the different auto-scaling strategies can thus be formulated as the CCP,

$$CCP : \min_{\rho} \rho \qquad (15)$$
$$\text{subject to: } \mathbb{P}\{d_{\mathcal{T}}(y, y^{\circ}) \leq \rho\} \geq 1 - \epsilon.$$

where the performance is evaluated in a "worst-case" fashion. This form for the CCP is sometimes referred to as *percentile optimization*, where ρ can be interpreted as the $(1-\epsilon)-$percentile.

4.2.4. Application of the scenario theory. Irrespectively of the choice of $d_{\mathcal{T}}(y, y^{\circ})$, finding the optimal solution $\rho^{\star}$ of the CCP (15) is an NP-hard problem. Indeed, it involves determining, among all sets of realizations of the stochastic input that have a probability $1-\epsilon$, the one that provides the best (lowest) value for $d_{\mathcal{T}}(y, y^{\circ})$. We then head for an approximate solution. Instead of considering all the possible realizations for the stochastic input, we consider only a finite number N of them, called *scenarios*, randomly extracted according to their probability distribution, and treat them as if they were the only admissible uncertainty instances. This leads to the formulation of Algorithm 1, where the chance-constrained solution is determined using an empirical violation parameter $\eta \in (0, \epsilon)$. In the following we denote the realization of a scenario with the superscript $^{(i)}$, with $i = 1, 2, \ldots, N$.

ALGORITHM 1: Computation of the randomized solution.

1: extract N realizations of the stochastic input $\lambda^{(i)}(k)$, $k = 1, 2, \ldots, |\mathcal{T}|$, $i = 1, 2, \ldots, N$, and let $\kappa = \lfloor \eta N \rfloor$;
2: determine the N realizations of the output signals $y^{(i)}(k)$ $k \in \mathcal{T}$, $i = 1, 2, \ldots, N$, when the policy to be evaluated is fed by the extracted input instances;
3: compute
$$\hat{\rho}^{(i)} := d_{\mathcal{T}}(y^{(i)}, y^{\circ,(i)}),\ i = 1, 2, \ldots, N;$$
4: determine the indexes $\{h_1, h_2, \ldots h_{\kappa}\} \subset \{1, 2, \ldots, N\}$ of the κ largest values of $\{\hat{\rho}^{(i)},\ i = 1, 2, \ldots, N\}$
5: **return** $\hat{\rho}^{\star} = \max_{i \in \{1,2,\ldots,N\} \setminus \{h_1, h_2, \ldots, h_{\kappa}\}} \hat{\rho}^{(i)}$.

Hence, the CCP problem (15) is translated into a sample-based optimization program

$$SP : \min_{\rho} \rho \qquad (16)$$
$$\text{subject to: } d_{\mathcal{T}}(y^{(i)}, y^{\circ,(i)}) \leq \rho, \quad i \in \{1, 2, \ldots, N\} \setminus \{h_1, h_2, \ldots, h_k\},$$

where $\{h_1, h_2, \ldots, h_{\kappa}\} \subset \{1, 2, \ldots, N\}$ of the $\kappa = \lfloor \eta N \rfloor$ largest values of $d_{\mathcal{T}}(y^{(i)}, y^{\circ,(i)})$.

Notably, if the number N of realizations is appropriately chosen, the obtained estimate of $\rho^{\star}$ is chance-constrained feasible, with *a priori* specified (high) probability.

THEOREM 4.1. *Select a confidence parameter $\beta \in (0, 1)$ and an empirical violation parameter $\eta \in (0, \epsilon)$. If N is such that*

$$\sum_{i=0}^{\lfloor \eta N \rfloor} \binom{N}{i} \epsilon^{i} (1-\epsilon)^{N-i} \leq \beta, \qquad (17)$$

then, the solution $\hat{\rho}^\star$ to Algorithm 1 satisfies

$$\mathbb{P}\{d_{\mathcal{T}}(y, y^\circ) \leq \hat{\rho}^\star\} \geq 1 - \epsilon, \tag{18}$$

with probability at least $1 - \beta$. □

Theorem 4.1 provides theoretical guarantees that the chance constraints (18) are satisfied, i.e., that the solution $\hat{\rho}^\star$ of the optimization program SP (16) obtained through Algorithm 1 is feasible for the CCP (15). Theorem 4.1 is a feasibility theorem, it was proven by Campi and Garatti [2011], and says that the solution $\hat{\rho}^\star$ obtained by inspecting $\lfloor \eta N \rfloor$ constraints only is a feasible solution for (15) with high probability $1 - \beta$, provided that N fulfills condition (17). As η tends to ϵ, $\hat{\rho}^\star$ approaches the desired optimal chance constrained solution $\rho^\star$. In turn, the computational effort grows unbounded since N scales as $1/(\epsilon - \eta)$ [Campi and Garatti 2011], therefore, the value for η depends in practice on the time available to perform experiments.

As for the confidence parameter β, one should note that $\hat{\rho}^\star$ is a random quantity that depends on the randomly extracted input realizations and initial conditions. It may happen that the extracted samples are not representative enough, in which case the size of the violation set will be larger than ϵ. Parameter β controls the probability that this happens and the final result holds with probability $1 - \beta$. N satisfying (17) depends logarithmically on $1/\beta$ [Alamo et al. 2010]. Therefore, β can be chosen as small as 10^{-10} (and, hence, $1 - \beta \simeq 1$) without N growing significantly.

It is worth mentioning that the sensitivity of this methodology mostly depends on the value of ϵ, rather than on β. Indeed, when β is chosen small enough, the number N of experiments required is not significantly increased, and the only improvement that we get is the confidence level of the feasibility. On the other hand, slightly changing ϵ affects the number of experiments significantly, but it also means that we are changing the CCP that one wants to solve. A typical use of (17) is thus deciding what is an acceptable level of risk, ϵ, choosing a small enough value of β, and computing the maximum number of experiments needed, according to the computational limits.

Notice that the guarantees provided by Theorem 4.1 are valid irrespectively of the underlying probability distribution of the input, which may not even be known explicitly, e.g., when feeding Algorithm 1 with historical time series as realizations of the stochastic input $\lambda^{(i)}$, as we are doing in this work. For the sake of completeness, we performed a correlation analysis on the input realizations used in the experiments. The results of the correlation analysis are presented in Appendix D.

5. EXPERIMENTS AND RESULTS

5.1. Solving the CCP

We assume that an acceptable risk is $\epsilon = 0.05$, meaning that the probability that the chance constraints are not satisfied is 5%. Since each experiment considers a workload over almost six years, accepting that 5% of the time we might not able to fulfill (18), is a reasonable choice for a regular web service. Among all the experiments, we can choose a fraction $\eta \in [0, \epsilon)$ that can be discarded without affecting the feasibility of the solution. We here set $\eta = 0.01$. As for the choice of the confidence level $1 - \beta$ that we have in the obtained result, as discussed in Section 4.2.3, we can take $\beta = 10^{-10}$, without affecting too much the number of experiments required for the evaluation.

Having chosen these parameters, it is possible to compute N such that it satisfies inequality (17), yielding $N = 796$. It is thus trivial to compute the number $\kappa = \lfloor \eta N \rfloor = 7$ of scenarios that can be discarded according to Algorithm 1. Publicly available workloads are scarce [Bodik et al. 2010], and finding $N = 796$ workload traces to solve the SP might not be always possible. However, one can easily adapt the above parameters to fit the available dataset, yet keeping similar probabilistic guarantees. For example,

choosing $\eta = 0$ and $\beta = 10^{-6}$, but keeping $\epsilon = 0.05$, one requires $N = 270$ workloads traces, with no discarded trace. In order to have a high confidence in the feasibility of the result, and in order to be able to discard possible traces containing sporadic events that leads to bad performance, we used the parameters leading to $N = 796$.

5.2. The workloads

Since artificially generated workloads may not be a good representation of real workloads and would thus affect the results of our study, we choose to perform the evaluations with real workloads. Given the scarcity of publicly available workloads, it is challenging to obtain 796 real commercial workload streams. On the other hand, the sixth most popular website on the World Wide Web is Wikipedia the open and collaborative online encyclopaedia, according to Alexa [2015]. Wikipedia is hosted on the servers of the Wikimedia foundation which hosts other projects such as Wikiquotes and Wikibooks. The Wikipedia foundation provides publicly available workload traces that logs the per page aggregate number of requests per hour to all services hosted by the foundation [Barrett 2008; Ali-Eldin et al. 2014a].

This workload is interesting since the traces can be separated into independent streams with each stream representing a project and a language, e.g., one stream can have the requests to the German Wikipedia pages, another can be for the Swedish Wikitionary project, and a third can be for the Zulu Wikibooks project, and so on. In total, the Wikimedia foundation hosts over 888 projects and language combinations. Many of these project streams can be divided even further into, for example, load from Mobile users, load generated by editors, load generated on "talk pages" and so on.

We downloaded the Wikimedia foundation traces for the period between the 9^{th} of December, 2007 and the 16^{th} of October, 2013. The traces were analyzed and all the different streams that are present in the load during this period of almost six years were extracted. The extraction yielded more than 3000 different workload streams. We chose $N = 796$ streams to be used to validate our framework.[3] The chosen streams are mostly of long running projects spanning the whole period of the trace or are streams which have high request arrival rates. We have performed a correlation analysis on the selected workloads, and we found that they are practically not correlated. Further details on the workloads used and the correlation analysis are given in Appendix B and Appendix D respectively.

5.3. Simulating the cloud

Since it is not feasible to run 796 experiments per auto-scaler on a real testbed – meaning 4776 experiments in total – each of them using a workload spanning around 6 years with millions of requests per hour, we decided to use a simulator which we parameterize. A summary of some of the main decisions in our experiments follows.

(1) We use a discrete event simulator that we parameterize using real server experiments on our server testbed. The experiments use one real application and one benchmark.
(2) Many of the techniques we test have more components other than auto-scaling and capacity prediction algorithms for other functionalities. For example, PLBS does not just predict the workload and the required capacity, but also tries to optimize the choice of the size of the VMs deployed, AGILE also has components for VM cloning and migration. Since our main target is to evaluate the accuracy of the prediction

[3]We have open sourced the workloads for other researchers to use. The 796 processed workloads can be found at: http://zenky.cs.umu.se/PEAS/.

of the capacity required, when needed we have only focused on evaluating this part of the system as it is the core part of auto-scaling.

(3) The traces we have are logged on hourly bases with no smaller logging granularity available. Using the data as is with the auto-scaling algorithms can undermine the fairness of the experiments since, with the exception of Hist, the algorithms are designed to operate on a seconds to few minutes interval. We have considered two main approaches to use the traces in our experiments. The first one is to interpolate the values for smaller time granularities based on either another workload that we have with a finer logging granularity or some other method such as regression. This approach is similar to the approach taken by Malkowski et al. [2011]. The second approach is to scale down the workload intensity by dividing the number of requests by a factor, e.g., scaling down the workload by a factor of 3600 gives for each hour the average arrival rate per second.
The first approach has a clear disadvantage since the interpolation can distort the actual workload and thus give wrong performance results. The second approach has the disadvantage of not capturing the transient workload effects if the workload is scaled down by a large factor, i.e., by dividing the workload by a factor, the effect of some extreme spikes that happened at a smaller time granularity can be reduced due to the down scaling of the workload. Since we did not want to introduce any artificial behavior in our evaluation, we opted for the second solution, down sizing the workload by a factor. To show that the results we obtain are still valid for bursty workloads, we choose two factors to scale down the workload, with one of the two factors small to capture the case of bursty workloads, as discussed later. Scaling-down a workload is an approach taken typically to compensate for the limitations of testbeds as done by Gandhi et al. [2012] and Urgaonkar et al. [2008].

To parameterize the simulator, we performed some experiments using C-Mart, an open-source cloud computing benchmark designed by Turner et al. [2013] that emulates modern cloud applications, and using a MediaWiki installation on which we replicated the Spanish and German Wikipedia pages [Barrett 2008]. [4]

C-Mart is chosen since it represents a modern and up-to-date web application. Turner et al. compare C-Mart to RUBIS and TPC-W. They show through experimental evaluation that their benchmark reflects a more realistic cloud application compared to many benchmarks used in the literature. We have also used another widely used web application, Mediawiki, to complement our results. There are a few motivations behind using Mediawiki with a full replica of the Spanish Wikipedia beside the usage of C-Mart to parameterize the simulator. First, the workloads we are using are workloads directed to web services using Mediawiki, e.g., Wikipedia, Wikibooks and Wiktionary. In addition, this application has recently gained popularity in the literature as a modern, real, and representative application of cloud and web services [Fernandez et al. 2014; Krioukov et al. 2011; Casalicchio and Silvestri 2013; Blagodurov et al. 2013; Difallah et al. 2013]. C-Mart contains both stateful and stateless applications. Some setups of MediaWiki can also be stateful. Since many of the algorithms tested and most of the algorithms found in the literature consider stateless applications, we choose to consider only the stateless case.

[4]The code for C-Mart is open-source and available by the authors. We have open-sourced the MediaWiki images used in our parameterization to enable other researchers to replicate our results, and use the images with not much overhead. We have also open-sourced the workload generator used. The images and the generator can be downloaded from: http://zenky.cs.umu.se/PEAS/. We note that there are two images, The first one uses MediaWiki with no optimization, while the second one uses Memcached with Mediawiki optimizing the performance considerably.

The service rate of a VM is modeled to be a Poisson process. Using a set of experiments where we have run multiple instances of both the chosen applications, we found that the average number of requests that a VM can serve per second varies considerably with the workload mix used in the experiment. For our experiments we set the number of requests that a VM can serve per second to 22 requests for a VM with one core assigned and two Gigabytes of RAM. Appendix A describes the experiments done to obtain the average number of requests a VM can serve per second.

We performed additional experiments in order to estimate the startup time and shutdown time of a VM. The average startup time resulted in 29.7s, with a very low standard deviation of 0.30s, while the average shutdown time resulted in 6.72s, with a standard deviation of 0.45s. In both cases the time required to startup or shutdown a VM is fairly contained, and always less than 1 minute, i.e., the time unit considered in the simulator. Therefore, without loss of generality, the simulator considers 1 time unit of delay for the startup of a VM, and 0 time units of delay for shutting down a VM. All the tested algorithms were tuned according to this information. A more detailed analysis of the conducted experiments can be found in Appendix E. These experiments are then used to parameterize a discrete-event python-based simulator to simulate the cloud infrastructure[5]. All the auto-scalers are then evaluated using the simulator instead of the actual application deployments.

When parameterizing the auto-scalers, we followed any guidelines set by the authors in their published papers or any tips we received through direct communication with the authors. If the authors did not provide such guidelines or tips, we performed some simple parameter sweeping and sensitivity analysis for the algorithms, and tuned the algorithms accordingly. The sheer volume of the experiments we conducted and the length of the workloads allowed us to find logical errors in the codes we obtained or implemented, such as not handling division by zero. Whenever we found such a problem, we tried to fix it, leaving the original algorithm untouched.

5.4. Three Case Studies

We consider three case studies to evaluate PEAS, in which we want to understand which is the best algorithm among the ones selected in Section 3.1, according to the metrics presented in Section 4.2.2. The three case studies are as follows:

(1) The first case study assumes that there are no queuing effects, i.e., that delayed requests are dropped, and that all requests are short and homogeneous, i.e., they require one time unit to be processed. As a result, $q(\cdot) = 0$ and $C_{\text{lrr}}(\cdot) = 0$ in (3). As discussed earlier, since the workload granularity we have is large, i.e., total number of requests per hour, the workload is divided by a factor that corresponds to the time granularity, get the average number of requests to be processed per second, and reduce the magnitude of the total workload. For this case study, we choose this factor to be 3600. This smooths the workload, and operates on the average number of requests.

(2) The second case study considers the case when delayed requests are queued in an infinite buffer. The requests are non homogeneous with some short requests and some long running requests that can take up to 60 time units to process. Long running requests and buffered requests form a significant percentage of cloud workloads [Reiss et al. 2012]. While there are no infinite buffers, assuming an infinite buffer in our simulations enables us to reason on the algorithm behavior and detect possible software bugs in the implementation of the algorithms, e.g., both PLBS and

[5]The code of the simulator is open-sourced with the auto-scalers. Relevant technical details of the simulator, can be found in Appendix A.

AGILE exhibited very bad performance due to queuing effects, leading all the considered metrics to diverge to infinity in the second case study. After careful analysis, we found that for some abrupt workload changes, AGILE was predicting negative values for the workload. We have added a check on the predictions from AGILE to make sure that when the predictor predicts a negative value, the predictions are discarded. Spotting the problem with AGILE was simplified due to the use of PEAS as the framework pointed us to where the errors occur in the predictions. The workload is divided by a factor 3600 like the first use case to get the average number of requests per second.

(3) The third case study is similar to the first one. Again, there are no queuing effects. It is assumed that all requests are short and homogeneous, and that, $q(\cdot) = 0$ and $C_{\text{lrr}}(\cdot) = 0$ in (3). The third case study differs in the considered workload which we scale down by a factor of 60 only, while assuming that the machine can serve 22 requests per second. A smaller scaling factor for the workload means that the workload is more bursty as the bursts in the scaled down workloads would now require more machines to handle the bursts. This is also equivalent to having a VM that can serve a lower number of requests per second. The scenario thus stresses the auto-scalers to a greater extent and is interesting to show the tradeoffs when all the workloads in the system are very bursty, an unlikely but interesting case to study.

5.5. Experimental results

In the experimental evaluation, recall that we are considering distances with respect to a desired behavior. Therefore, in all the presented results, the lower the value of the distance, the better the algorithm is performing. The following figures (Figures 4 to 7) have on the x-axis the considered algorithms, and on the y-axis the value $\hat{\rho}^\star$ obtained as a solution of the SP (16), after performing $N = 796$ experiments, and when considering the specified distance. The raw numbers of all the presented results are summarized in Table I.

The solution of the SP (16), when considering distance (4) for the considered auto-scaling algorithms and for the three case studies is presented in Figure 4. As discussed above, this norm distance accounts for several aspects, i.e., over- and under-provisioning, how much the assigned capacity oscillates around the ideal one, and also for how long the auto-scaler is not behaving in an ideal way. As a result, $d_{\mathcal{T}}^{\text{norm}}$ gives a general idea of the overall average performance. Adapt shows better performance than the other algorithms in all three case studies, providing significantly lower values of the considered distance. We note the difference in performance between case study 1 and case study 3. Since the scale down factor for the workloads is lower for case study 3, the workloads are much more bursty. While the performance of all algorithms for the bursty workloads is much worse, the performance of both Reg and Hist has relatively decreased more than the other algorithms while the performance of React has relatively improved, i.e., the ordering of the performance of the algorithms. Reg and Hist both depend on the regularity in the workload pattern in their predictions. For very bursty workloads with a lot of unpredictable spikes, React performs well since the predictions follow the spikes as they occur. Note the lighter blue columns in the graphs are used to indicate that the value for this algorithm is quite high compared to the other values.

However, from the service provider viewpoint, the information about the norm is difficult to interpret, and can only be used for ranking the algorithms. Therefore, other quantities need to be used for a more interesting and informative evaluation. Indeed, the service provider wants to know quantitatively how close to optimal the auto-scaling algorithm is from assigning the ideal amount of resources. This information can be ob-

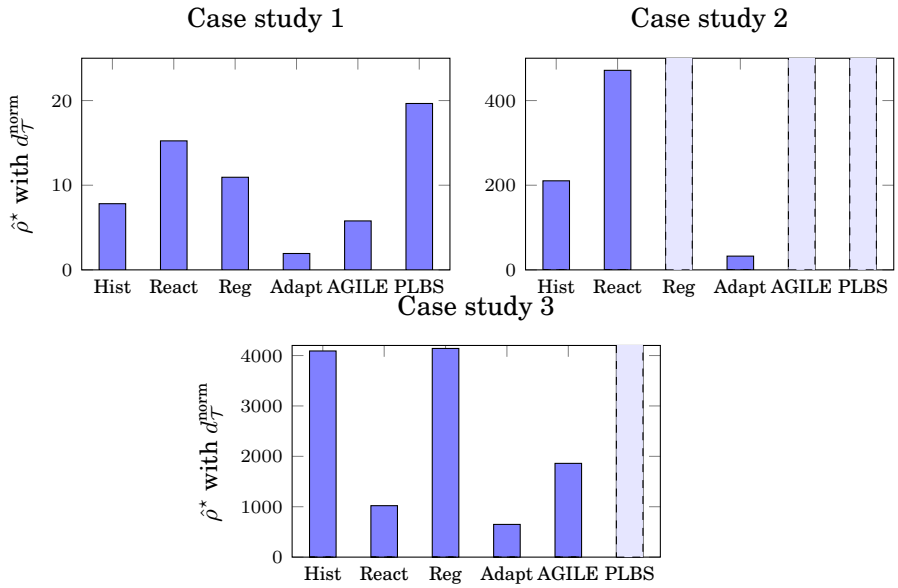

Fig. 4. Results of the scenario approach with distance (4).

tained through distances (6), (7), and (8). If these distances are adopted for the solution of the SP (16), the service provider can guarantee what will be the maximum over- and under-provisioning, with a probability of $1 - \epsilon$.

Figure 5 shows the obtained results with these three distances, and for the three case studies. For the sake of presentation, we used $-\hat{\rho}^\star$ for the under-provisioning

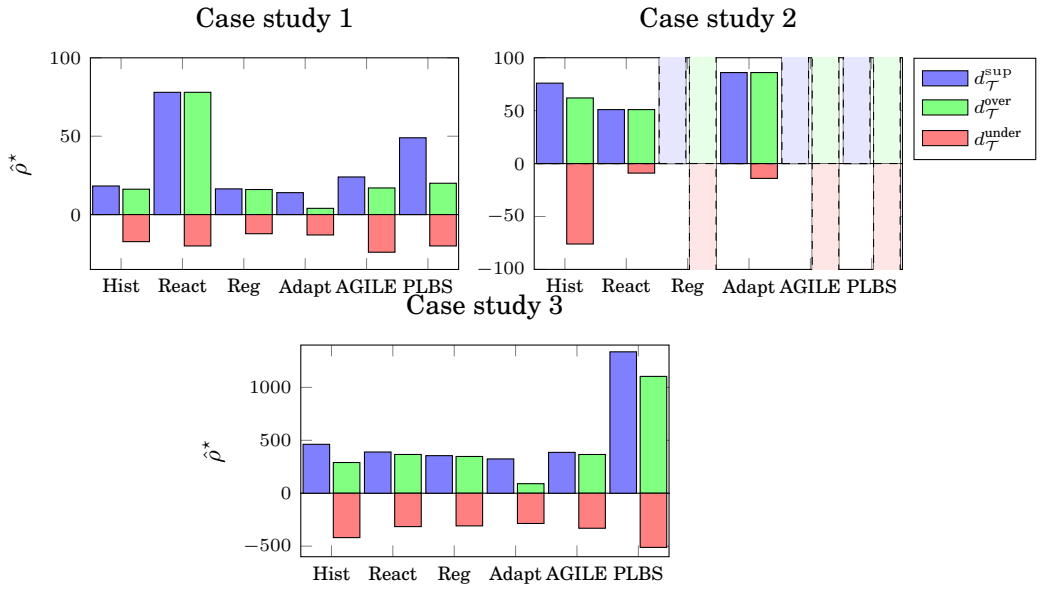

Fig. 5. Estimate of the maximum Over-Provisioning (OP) and Under-Provisioning (UP).

case, indicating that the allocated resources are below $y^{\circ}(\cdot)$. The computed values can thus be used similarly to confidence interval extremes.

Analyzing the obtained results, in the first case study Adapt is able to keep the capacity fairly close to its ideal value, with a maximum bound of 4 servers more, and of 13 server less than actually needed. On the other hand, in the second case study, React is the algorithm that exhibits better performance, with a maximum bound of 51 servers more, and of 9 server less than actually needed. Reg and AGILE have very high values of instantaneous over- and under-provisioning. The third case is similar to the first two cases except for React which again performs relatively much better than the first case. Dashed bars represent distances that are significantly higher than the other methodologies, and for improving the readability of the graph, were left out. For further details on the raw numbers see Table I. Notice that using PEAS, we can guarantee that these bounds on over- and under-provisioning hold with a probability $1-\epsilon$ with a confidence that is practically 1.

Whereas maximum over- and under-provisioning is of extreme importance, there is however one aspect that must be considered, i.e., how long an auto-scaling technique spends in an over- or under-provisioning state. This aspect is somehow orthogonal to the one discussed above. Indeed, there may be situations in which, at every time instant, the auto-scaling technique is close to the ideal behavior, but this is never reached, resulting in losing revenue (in the case of under-provisioning), or wasted energy and resources (in the case of over-provisioning). On the other hand, a technique may have some high yet short "spikes" in the capacity allocation, but generally being extremely close to the ideal allocation. This last case presents poor performance, for example, with respect to distance (6), while generally behaving better than the previous approach.

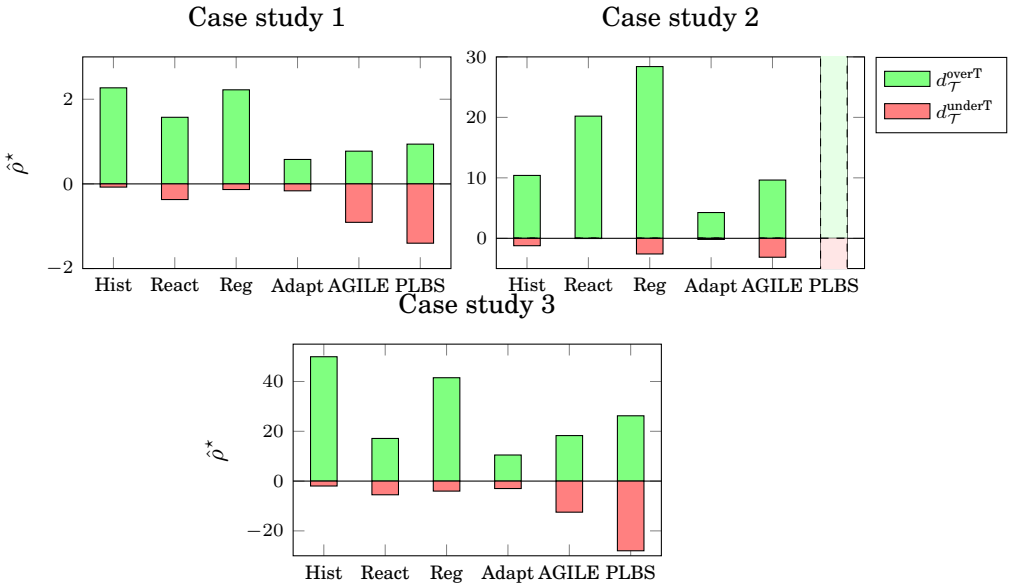

Fig. 6. Worst case average Over-Provisioning (OP) and Under-Provisioning (UP) in a time unit.

In order to take into account this aspect, we here consider distances (9), and (10), that measure the average over- and under-provisioning in a time unit. Figure 6 shows

the obtained results with the different algorithms. Also in this case we considered $-\hat{\rho}^\star$ for distance (10) analogously to Figure 5. It is possible to see from Figure 6 that Adapt is the algorithm providing better performance in both the case studies. However, if one compares Figures 5 and 6, it is possible to see how AGILE might have bad instantaneous performance but, AGILE improves its performance in all the case studies when considering this distance.

It is worth noting that the majority of the algorithms tends to over-estimate the needed capacity, rather than under-estimate it. This behavior is desirable from an auto-scaler, since under-estimation could cause large losses for the service provider, especially if the auto-scaler keeps under-estimating for long periods. In this respect, React and Adapt show the best performance in the second case study, which is also the more realistic one.

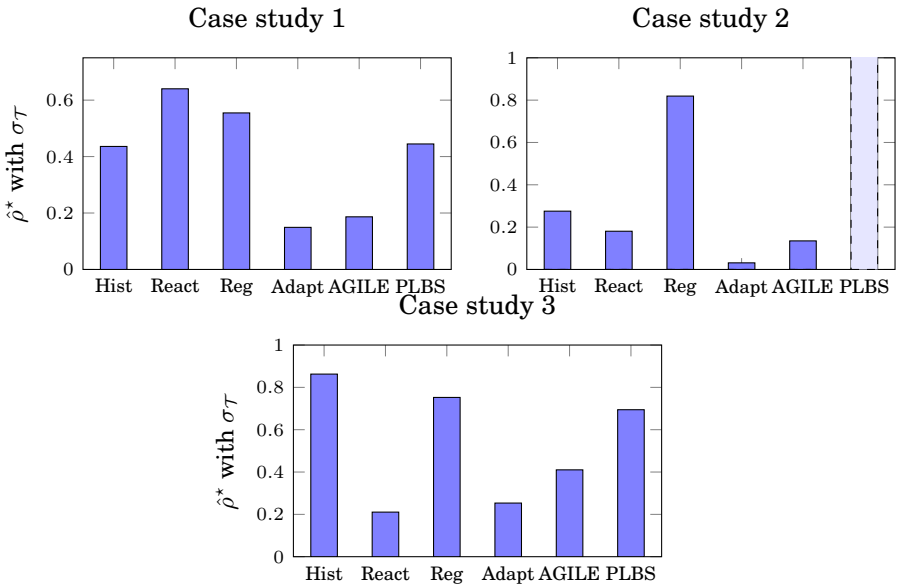

Fig. 7. Normalized adapted ADI for the considered auto-scaling strategies.

As a last metric, we consider the normalized adapted ADI (14). Figure 7 shows the obtained results. As already mentioned, this metric is similar to the norm (4), since it considers the overall performance, accounting both for over- and under-provisioning, and possible oscillations around the desired value. The only qualitative difference is that it considers an interval of values around the real target, it penalizes only when the auto-scaler is outside that interval. We here considered $L = 0.9$ and $U = 1.1$, thus not penalizing auto-scalers that are committing an error below 10%. According to this metric, Adapt is the auto-scaler with the best performance in the first two case studies, while React is the best one in the third case study.

5.6. Discussion

The experimental results suggest that newly published algorithms do not perform particularly better than the older ones. Interestingly, React, which is one of the simplest auto-scalers possible, performs better than most of the other algorithms especially in the second case study. These results hold for all the workloads similar to the ones we

use. Since we considered 796 different workloads, each over about six years, the obtained results can be generalized, and PEAS allows one to quantify the confidence in the obtained performance. In particular, Adapt is the algorithm that is almost invariantly better than the other ones both in the first and in the third case study. On the other hand, in the second case study, React is the auto-scaler showing better performance in terms of under-provisioning. Since this is a relevant aspect from the service provider viewpoint, this data would suggest that React could be a good alternative to Adapt. A summary of the obtained results is presented in Table I.

Even though the six chosen controllers are not an exhaustive set of the algorithms proposed of the state-of-the-art, they are a representative set. The framework is very flexible in terms of what performance metrics can be integrated and the ease of use to test an algorithm that has not been tested yet. The framework enables the research community to compare the performance of published auto-scalers and the contribution of newly proposed ones in the future.[6] We believe that this is of great value for testing, quantifying and comparing the performance of auto-scalers.

Table I. Summary of the obtained results. The best values are highlighted in bold.

Case study 1						
$d_{\mathcal{T}}^{(\cdot)}$	**Hist**	**React**	**Reg**	**Adapt**	**AGILE**	**PLBS**
norm	7.82	15.2	10.9	**1.94**	5.78	19.7
sup	18.2	78	16.4	**14**	24	49
under	17.3	20	**12.2**	13	24	20
over	16.2	78	16	**4**	17	20
underT	**0.0757**	0.37	0.133	0.164	0.908	1.4
overT	2.27	1.57	2.22	**0.577**	0.773	0.939
$\sigma_{\mathcal{T}}$	0.436	0.64	0.555	**0.149**	0.186	0.445
Case study 2						
$d_{\mathcal{T}}^{(\cdot)}$	**Hist**	**React**	**Reg**	**Adapt**	**AGILE**	**PLBS**
norm	210	471	1.71×10^4	**32.5**	2.25×10^3	Inf
sup	76	**51**	5.08×10^3	86	2.49×10^3	Inf
under	76	**9**	4.24×10^3	14	2.49×10^3	Inf
over	62	**51**	5.08×10^3	86	514	Inf
underT	1.22	**0.0356**	2.59	0.181	3.13	Inf
overT	10.4	20.2	28.4	**4.26**	9.64	Inf
$\sigma_{\mathcal{T}}$	0.275	0.181	0.819	**0.0312**	0.135	Inf
Case study 3						
$d_{\mathcal{T}}^{(\cdot)}$	**Hist**	**React**	**Reg**	**Adapt**	**AGILE**	**PLBS**
norm	4.09×10^3	1.02×10^3	4.14×10^3	**649**	1.86×10^3	1.38×10^4
sup	462	389	355	**324**	386	1.34×10^3
under	420	316	309	**286**	331	512
over	290	366	347	**90**	366	1.1×10^3
underT	**1.98**	5.5	4.02	2.99	12.5	28
overT	49.9	17.1	41.5	**10.5**	18.3	26.2
$\sigma_{\mathcal{T}}$	0.863	**0.211**	0.752	0.253	0.41	0.694

To provide some insights into which workloads result in the worst performance with each auto-scaler, Table II shows the workloads causing the worst performance for each auto-scaler with each of the metrics for case study 1. Analogous results can be obtained by analyzing the other case studies, but we omit it due to space limitations. One can

[6]The code for the framework and for the simulator will be open-sourced.

see that some workloads affect the performance of many of the auto-scalers negatively due to their dynamics, e.g., the Polish Wikipedia, but this effect is stronger for some of the auto-scalers as can be seen from Table I.

Table II. The workload causing the worst performance for each auto-scaler with each measure for the first case-study.

	Case study 1					
$d_T^{(\cdot)}$	**Hist**	**React**	**Reg**	**Adapt**	**AGILE**	**PLBS**
norm	Polish Wikipedia	Japanese Meta Wiki	Polish Wikipedia	Portuguese Wikipedia	Portuguese Wikipedia	Dutch Wikipedia
sup	German Wiktionary	Polish Wikipedia	Portuguese Wikipedia	French Wikipedia	Polish Wikipedia	German Wikipedia
under	French Wikipedia	Polish Wikipedia	Japanese Wikipedia	Japanese Wikipedia	Polish Wikipedia	Polish Wikipedia
over	Polish Wikipedia	Polish Wikipedia	Polish Wikipedia	Mediawiki commons	Polish Wikipedia	French Wikipedia
underT	Portuguese Wikipedia	Portuguese Wikipedia	English Wikipedia	Portuguese Wikipedia	Portuguese Wikipedia	Polish Wikipedia
overT	Portuguese Wikipedia	Portuguese Wikipedia	Portuguese Wikipedia	Dutch Wikipedia	Portuguese Wikipedia	Polish Wikipedia
σ_T	Turkish Wikipedia	Japanese Wiki	Japanese Meta Wiki	Finish Wikipedia	Japanese Meta Wiki	Dutch Wikipedia

6. CONCLUSION AND FUTURE WORK

In this paper we propose PEAS, a framework for the evaluation of auto-scaling strategies that is able to provide probabilistic guarantees on the obtainable performance. In particular, we consider three case studies, where we evaluate six different algorithms and test them against 796 distinct real workload traces from projects hosted on the Wikimedia foundations' servers. The considered case studies show the need for a deeper evaluation of the auto-scaling techniques proposed in the literature. The results obtained using PEAS highlighted the problem of generalizing the conclusions of the original published studies. As future work, we are developing a more comprehensive comparative evaluation of auto-scaling strategies using PEAS including more algorithms.

On the other hand, although PEAS was conceived for evaluating auto-scaling strategies, the approach is quite general and can be applied to different resource management problems, especially when stochastic quantities hinder the possibility of providing performance guarantees for the considered methodologies. We envision that, in the near future, the proposed framework can be successfully applied also to a larger class of algorithms, by suitably defining application specific performance measurements and metrics.

APPENDIX

A. PARAMETRIZATION OF THE SIMULATOR

In order to validate and parametrize our simulator, we used C-Mart, a recently published cloud benchmark developed by Turner et al. [2013]. The benchmark is publicly available [Payne 2014]. C-Mart has been designed with cloud performance testing in mind. It utilizes modern web technologies, such as JavaScript, CSS, AJAX, HTML5, SQLite, MongoDB and Memcache DHT, in order to build cloud web applications.

We deployed the benchmark on part of our local 640 cores cluster to emulate a modern two tiered web service running an online auction and shopping website. The backend server runs MySQL while the frontend runs Tomcat application servers. The web application hosted utilizes technologies such as HTML5, AJAX, CSS, rich multimedia, and SQLite. The utilization of these technologies to build C-Mart is the main reason for selecting C-Mart over other widely used benchmarks such as RUBiS and TPC-W.

Both the frontend and backend servers are virtualized KVM images. We used the images provided by the benchmark authors as is, but modified the workload generator to accept the number of users from a trace file while keeping the average number of requests per users bounded. The workload generator run as a standalone application on a bare-metal server with 32 CPU cores and 56GB of memory. To better investigate

the tradeoffs between the average number of requests, the average latencies and the resources allocated to a machine, the database VM also runs on a separate physical server in order to maintain no interference between the database VM and the application VM. The database VM has 8 CPU cores and 10GB of memory, in order to ensure that the database tier is not the bottleneck.

For the frontend VMs, we vary their sizes to obtain the average number of requests a VM is able to serve per time unit. The load is mixed with 24 different pages a client can access with a mixture of images, CSS, video, and text. In the beginning, the size of the front-end machines was kept small, with one CPU and 1GB of memory. This configuration corresponds to a t2.micro Amazon EC2 instance.

We ran several experiments to measure r_{app} (see Section 4.2.1), the average number of requests one VM is capable of serving in an acceptable response time, i.e., below 500ms. This number changed based on the workload mix. For some experiments it was as high as 100 to 150 requests per second per VM, while for other experiments the number was as low as 5 to 10 requests per second per VM. These numbers conform with multiple measurements from online deployed services, for example, Justin.tv backend servers can handle 10 to 20 requests per second per VM on average [Hoff 2010]. Similar numbers were also obtained for the Play framework on EC2 [Papauschek 2013].

We ran similar experiments using the MediaWiki server. In these experiments we built a workload generator which generate GET requests for different pages on the Spanish Wikipedia. Since the pages vary in size, we decided to run the experiment for the worst case scenario when all requests are directed only to the largest page on the Spanish Wikipedia, “Alsacia en 1789”. We then stress the VMs to find the maximum number of requests that a VM can handle with an end-to-end response time – between a workload generator running on a machine in Lund and the VMs located in Umeå, that are located about 1000km far as the crow flies – of 4 seconds or less for the full page to load. The average number of requests per second varied between 20 and 25 requests per seconds in different experiments for a VM with 1 core and 2 GB of RAM. We thus decided to consider the average service rate of a simulated server to follow a Poisson process with an average of $r_{\text{app}} = 22$ requests per time unit per VM. In the simulations, the number of VMs provisioned for a service is then computed using Equation (3).

B. WORKLOAD EXAMPLES

Figure 8 shows some examples of workload streams from the ones used for the evaluation. The workloads are very diverse in nature, in evolution, in burstiness, and in magnitude. Almost all of the traces in the dataset show some form of diurnal pattern where the number of requests increase during some hours of the day. Since these workloads contain very regional workloads, e.g., the requests on the Vietnamese Wikipedia, the workloads also differ on when these diurnal patterns occur. The diversity in the workloads shows the importance of evaluating the performance of an auto-scaler using many different workloads. It also strengthen our argument for an evaluation framework that can provide probabilistic guarantees on the evaluation number. Table III gives more information about the workloads shown in Figure 8.

Table III. Description of some of the workloads used for the evaluation.

Name	key	Language	Project	Name	key	Language	Project
a	ar.q	Arabic	Wikiquote	f	it	Italian	Wikipedia
b	de	German	Wikipedia	g	jp	Japanese	Wikipedia
c	en	English	Wikipedia	h	nl.q	Dutch	Wikiquote
d	es.mw	Spanish	Wikipedia Mobile	i	pl	Polish	Wikipedia
e	fr.d	French	Wiktionary				

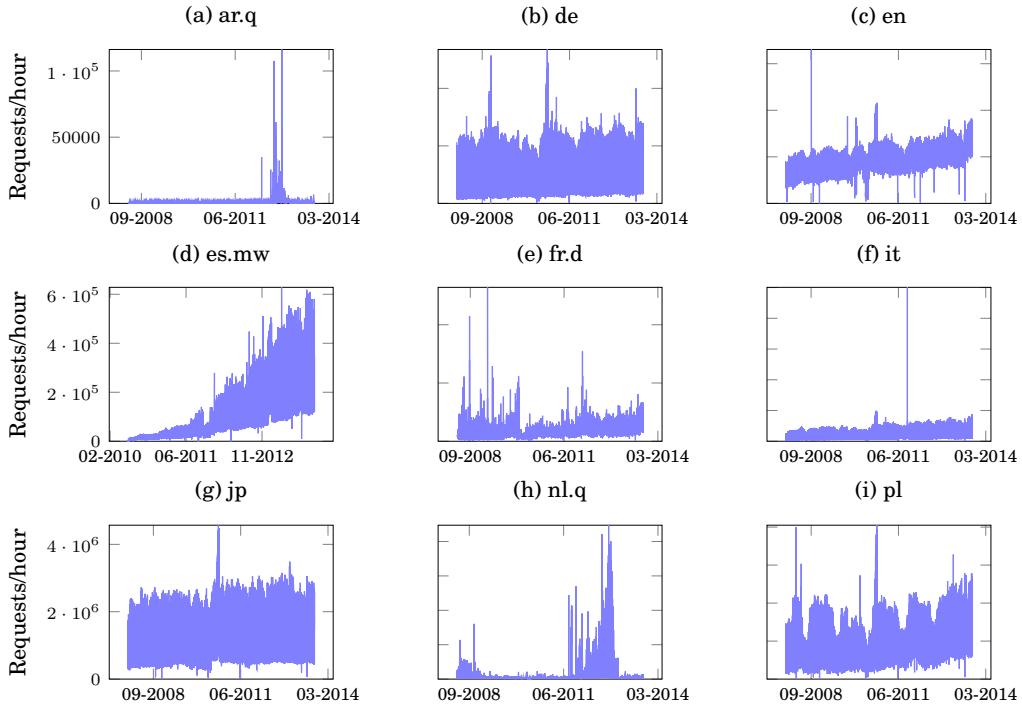

Fig. 8. Some of the workloads used for the experiments

C. WORKLOADS AT DIFFERENT GRANULARITIES

As the workloads used in the paper have a per hour granularity, a question arises if the workloads are representative enough of workloads at a lower granularity. We have acquired older Wikimedia traces that span a two months period in 2008, between the 1st of September, and the 29th of October. These traces have a granularity of milliseconds. The two months period has a non-uniform and non-Poissonian request arrival rate. In order to test how the granularity of the workload changes the workload characteristics, we have aggregated the workload from a milliseconds granularity, to a seconds granularity by summing all arrivals that occur in a second. We have also aggregated the workload to an hour granularity summing all requests that arrived in a given hour. We then divide the hour-granularity workload arrivals by 3600 to produce a case similar to the workloads used in the first and the second use cases discussed in Section 5.4. We then plotted the Auto-Correlation Functions versus time lags equal to a period of more than one day as shown in Figure 9. Note the difference in the units on the X-axis. We note that the two workloads have the same cycle with a clear daily cycle that has the same correlation at roughly the same lag, every 12 hours as shown in Figure 9.

As the ACF does not show the burstiness profile for the workloads, we have calculated the normalized entropy and the Average Sample entropy (SampEn) of the workloads at different granularities as described in [Minh et al. 2010] and [Ali-Eldin et al. 2014b]. The entropy of the workload at the large granularity is 0.979 and at the small granularity is 0.973. The SampEn for the workload at the large granularity is 0.08 and at the small granularity is 0.05. We note that SampEn is an unbounded measure, i.e., it takes values between 0 for low burstiness and infinity for very bursty. This con-

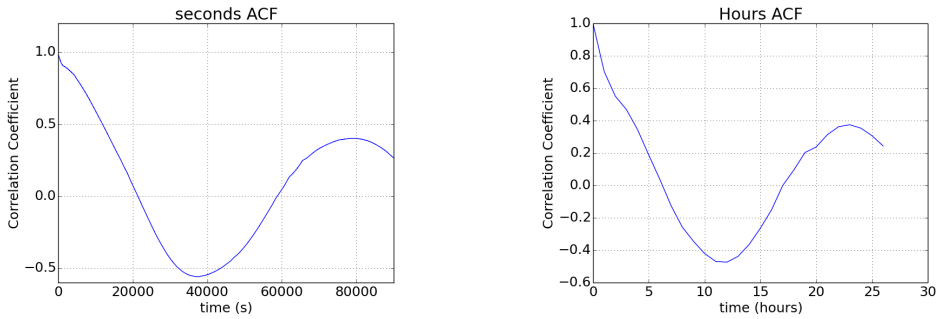

Fig. 9. The statistical characteristics of the same workload at different granularities is similar.

firms that the method we used to downsize the workload does not affect the workload characteristics considerably. [7]

D. CORRELATION ANALYSIS OF THE WORKLOADS

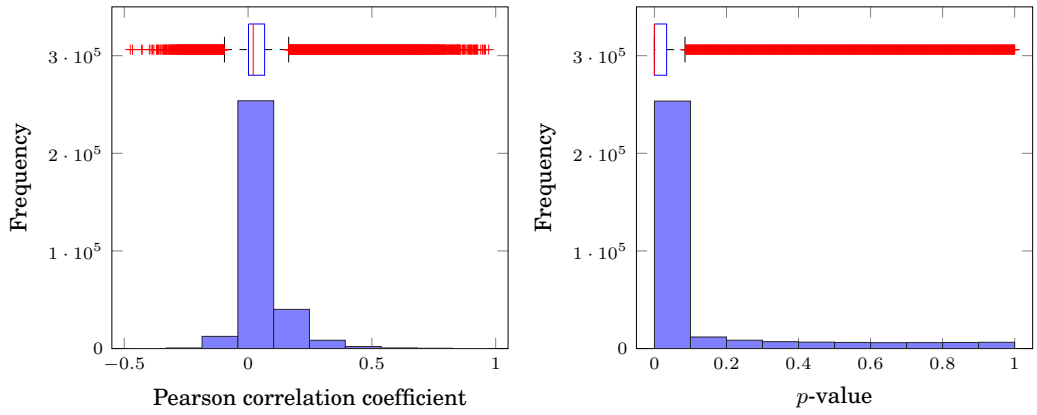

Fig. 10. Distribution of the Pearson correlation coefficients (on the left), and of the p-values (on the right) for the correlation analysis.

To provide a better understanding of how the workload streams used in this study are related, we performed a correlation analysis on the 796 workload streams used. We calculated the Pearson correlation coefficient between all possible pairs of workload streams, e.g., (jp, pl), (de, en) , (de, es) and so on. Figure 10 shows how the correlation coefficients, and the corresponding p-values are distributed. Each of the figures shows the distribution of correlations as a histogram and on top of the histogram a

[7]All scripts used for calculating the charctersitics at different granularities are available at http://zenky.cs.umu.se/PEAS/.

box-plot representing the spread of the obtained values. The figures indicate that the correlation coefficients are mostly concentrated close to 0, which is highlighted also by the box-plots. The p-values confirm that there is not statistical evidence of correlation between the different workloads, since they are mostly concentrated close to zero: The 75th percentile is 0.034. When analyzing the correlations more in detail, there are surprising results. For example, The load on the German Wikipedia is highly correlated with the load on the Spanish Wikipedia (Correlation coefficient greater than 0.7) while both are not correlated with almost any other load on any European language Wikipedia such as the load on the French Wikipedia, the load on the Italian Wikipeda, the load on the Norwegian Wikipedia, the load on the Portuguese Wikipedia, or the load on the Swedish Wikipedia.[8]

E. ANALYSIS OF STARTUP TIME AND SHUTDOWN TIME

To evaluate the statistical characteristics of the startup and shutdown time of VMs, we have made an experiment where we installed MediaWiki [Barrett 2008] on an Ubuntu Linux server VM. The VM was assigned 4 cores and 10 GB of RAM on an HP Elitedesk G1 SFF machine with a quad-core hyper-threaded Intel Core I7 processor. The VM is controlled using Vagrant [Palat 2012] running on Virtualbox. The VM is started and stopped 200 times and the VM startup and shutdown times are recorded from the time the startup command is issued till the MediaWiki application is responsive. To make the experiments realistic, the physical machine hosting the MediaWiki VM has been injected with varying background load. The frequency of the measured startup and

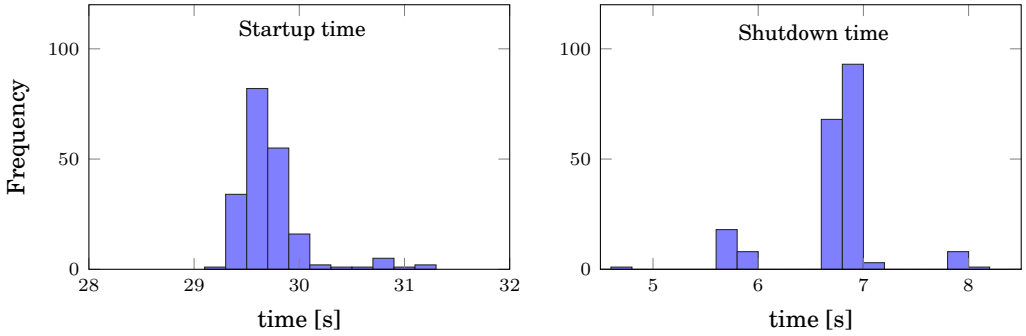

Fig. 11. Histograms for startup and shutdown time.

shutdown times are shown in Figure 11. The average and standard deviation for the measured startup times are 29.712s, and 0.310s respectively. The average and standard deviation for the measured shutdown times are 6.721s, and 0.455s respectively. By inspecting Figure 11, neither the startup nor the shutdown times appear to be normally distributed. In both cases performing a normality test rejected the hypothesis that the data is normally distributed. Figure 12 shows the empirical Cumulative Distribution Function (CDF) and the standard normal CDF for the two quantities, indicating also in the top right corner the p-value of the test. Note that the x axis indicates the normalized time, i.e., the average have been subtracted to the data, and then divided by the standard deviation. We tried to fit the data to four other distributions, namely, power law, stretched exponential, exponential, and log-normal, but none of the tested

[8]The whole dataset related to the correlation analysis of the workloads will be made publicly available, in case this manuscript is accepted.

probability distributions is able to describe these data. For the purpose of this work, the identification of the right probabilistic distribution is not relevant, since both the startup and shutdown times are below the time scale in which the auto-scaler takes its decisions. However, a probabilistic characterization of the startup and shutdown times deserves deeper investigation, which is left to future work.

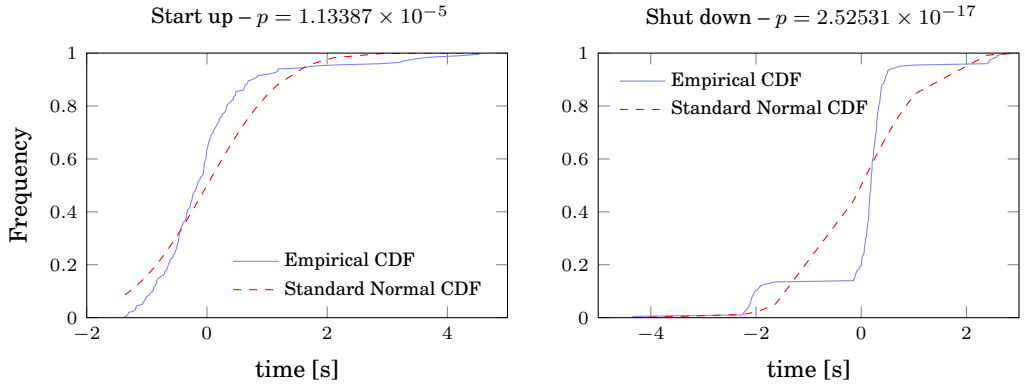

Fig. 12. Empirical cumulative distribution functions versus standard normal distribution.

References

Alessandro Abate and Maria Prandini. 2011. Approximate abstractions of stochastic systems: a randomized method. In *Proc. 50th IEEE Conf. on Decision and Control and European Control Conf. (CDC-ECC)*. IEEE, 4861–4866. DOI:http://dx.doi.org/10.1109/CDC.2011.6161148

Ahmad Al-Shishtawy and Vladimir Vlassov. 2013. ElastMan: Autonomic Elasticity Manager for Cloud-based Key-value Stores. In *Proc. 22nd Int. Symposium on High-performance Parallel and Distributed Computing (HPDC 13)*. ACM, New York, NY, USA, 115–116. DOI:http://dx.doi.org/10.1145/2462902.2462925

Teodoro Alamo, Roberto Tempo, and Amalia Luque. 2010. On the Sample Complexity of Probabilistic Analysis and Design Methods. In *Perspectives in Mathematical System Theory, Control, and Signal Processing*, JanC. Willems, Shinji Hara, Yoshito Ohta, and Hisaya Fujioka (Eds.). Lecture Notes in Control and Information Sciences, Vol. 398. Springer Berlin Heidelberg, 39–50. DOI:http://dx.doi.org/10.1007/978-3-540-93918-4_4

Alexa. 2015. The top 500 sites on the web. (2015). http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html http://www.alexa.com/topsites[Online; accessed 2015-04-09].

Ahmed Ali-Eldin, Ali Rezaie, Amardeep Mehta, Stanislav Razroev, Sara Sjöstedt-de Luna, Oleg Seleznjev, Johan Tordsson, and Erik Elmroth. 2014a. How will your workload look like in 6 years? Analyzing Wikimedia's workload. In *Proc. IEEE Int. Conf. on Cloud Engineering (IC2E 14)*. IEEE Computer Society, Washington, DC, USA, 349–354. DOI:http://dx.doi.org/10.1109/IC2E.2014.50

Ahmed Ali-Eldin, Oleg Seleznjev, Sara Sjöstedt-de Luna, Johan Tordsson, and Erik Elmroth. 2014b. Measuring cloud workload burstiness. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 566–572.

Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. 2012. An adaptive hybrid elasticity controller for cloud infrastructures. In *IEEE Network Operations and Management Symposium (NOMS 12)*. 204–212. DOI:http://dx.doi.org/10.1109/NOMS.2012.6211900

F.J. Almeida Morais, F. Vilar Brasileiro, R. Vigolvino Lopes, R. Araujo Santos, W. Satterfield, and L. Rosa. 2013. Autoflex: Service Agnostic Auto-scaling Framework for IaaS Deployment Models. In *Proc. 13th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGrid 13)*. 42–49. DOI:http://dx.doi.org/10.1109/CCGrid.2013.74

Daniel J Barrett. 2008. *MediaWiki (Wikipedia and Beyond)*. " O'Reilly Media, Inc.".

Sergey Blagodurov, Daniel Gmach, Martin Arlitt, Yuan Chen, Chris Hyser, and Alexandra Fedorova. 2013. Maximizing server utilization while meeting critical SLAs via weight-based collocation management.

In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 277–285.

Peter Bodik. 2010. *Automating Datacenter Operations Using Machine Learning*. Ph.D. Dissertation. EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-114.html

Peter Bodik, Armando Fox, Michael J. Franklin, Michael I. Jordan, and David A. Patterson. 2010. Characterizing, Modeling, and Generating Workload Spikes for Stateful Services. In *Proc. 1st ACM Symposium on Cloud Computing (SoCC 10)*. ACM, New York, NY, USA, 241–252. DOI:http://dx.doi.org/10.1145/1807128.1807166

Shaunak D. Bopardikar, Alessandro Borri, João P. Hespanha, Maria Prandini, and Maria D. Di Benedetto. 2013. Randomized sampling for large zero-sum games. *Automatica* 49, 5 (2013), 1184–1194. DOI:http://dx.doi.org/10.1016/j.automatica.2013.01.062

Emma S Buneci and Daniel A Reed. 2008. Analysis of application heartbeats: Learning structural and temporal features in time series data for identification of performance problems. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 52.

G. Calafiore and M.C. Campi. 2005. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming* 102, 1 (2005), 25–46. DOI:http://dx.doi.org/10.1007/s10107-003-0499-y

G.C. Calafiore and M.C. Campi. 2006. The scenario approach to robust control design. *IEEE Trans. on Automatic Control* 51, 5 (May 2006), 742–753. DOI:http://dx.doi.org/10.1109/TAC.2006.875041

M.C. Campi and S. Garatti. 2011. A Sampling-and-Discarding Approach to Chance-Constrained Optimization: Feasibility and Optimality. *Journal of Optimization Theory and Applications* 148, 2 (2011), 257–280. DOI:http://dx.doi.org/10.1007/s10957-010-9754-6

Marco C Campi, Simone Garatti, and Maria Prandini. 2009. The scenario approach for systems and control design. *Annual Reviews in Control* 33, 2 (2009), 149–157. DOI:http://dx.doi.org/10.1016/j.arcontrol.2009.07.001

Emiliano Casalicchio and Luca Silvestri. 2013. Mechanisms for SLA provisioning in cloud-based service providers. *Computer Networks* 57, 3 (2013), 795–810.

Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. 2001. Managing Energy and Server Resources in Hosting Centers. *SIGOPS Oper. Syst. Rev.* 35, 5 (Oct. 2001), 103–116. DOI:http://dx.doi.org/10.1145/502059.502045

T.C. Chieu, A. Mohindra, A.A. Karve, and A. Segal. 2009. Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. In *IEEE Int. Conf. on e-Business Engineering (ICEBE 09)*. 281–286. DOI:http://dx.doi.org/10.1109/ICEBE.2009.45

Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (Feb. 2013), 74–80. DOI:http://dx.doi.org/10.1145/2408776.2408794

Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment* 7, 4 (2013), 277–288.

Margaret A Dong and Richard K Treiber. 1992. Dynamic resource pool expansion and contraction in multiprocessing environments. (March 3 1992). US Patent 5,093,912.

Dror G Feitelson. 2014. *Workload modeling for computer systems performance evaluation*. Cambridge University Press. http://www.cs.huji.ac.il/~feit/wlmod/

Hector Fernandez, Guillaume Pierre, and Thilo Kielmann. 2014. Autoscaling Web Applications in Heterogeneous Cloud Infrastructures. In *IEEE Int. Conf. on Cloud Engineering (IC2E 14)*. Boston, MA, United States. https://hal.inria.fr/hal-00937944

F.L. Ferraris, D. Franceschelli, M.P. Gioiosa, D. Lucia, D. Ardagna, E. Di Nitto, and T. Sharif. 2012. Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds. In *Proc. 14th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 12)*. 423–429. DOI:http://dx.doi.org/10.1109/SYNASC.2012.58

Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. 2012. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *ACM Trans. Comput. Syst.* 30, 4, Article 14 (Nov. 2012), 26 pages. DOI:http://dx.doi.org/10.1145/2382553.2382556

Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *Proc. Int. Conf. on Network and Service Management (CNSM 10)*. 9–16. DOI:http://dx.doi.org/10.1109/CNSM.2010.5691343

James D. Hamilton. 1994. *Time Series Analysis*. Vol. 2. Princeton university press.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (second edition ed.). Springer-Verlag New York. DOI:http://dx.doi.org/10.1007/978-0-387-84858-7

Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. 2013. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *Proc. 4th ACM/SPEC Int. Conf. on Performance Engineering (ICPE 13)*. ACM, New York, NY, USA, 187–198. DOI:http://dx.doi.org/10.1145/2479871.2479899

Todd Hoff. 2010. Justin.tv's Live Video Broadcasting Architecture. (2010). http://highscalability.com/blog/2010/3/16/justintvs-live-video-broadcasting-architecture.html http://highscalability.com/blog/2010/3/16/justintvs-live-video-broadcasting-architecture.html[Online; accessed 2014-11-24].

Norden E. Huang, Zheng Shen, Steven R. Long, Manli C. Wu, Hsing H. Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung, and Henry H. Liu. 1998. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 454, 1971 (1998), 903–995. DOI:http://dx.doi.org/10.1098/rspa.1998.0193

Alexandru Iosup. 2012. IaaS Cloud Benchmarking: Approaches, Challenges, and Experience. In *Proc. 5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS 12)*. ACM, New York, NY, USA, 1–8. DOI:http://dx.doi.org/10.1145/2462307.2462309

Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. 2011. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Gener. Comput. Syst.* 27, 6 (2011), 871–879. DOI:http://dx.doi.org/10.1016/j.future.2010.10.016

Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. 2012. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Gener. Comput. Syst.* 28, 1 (2012), 155–162. DOI:http://dx.doi.org/10.1016/j.future.2011.05.027

Eamonn Keogh and Shruti Kasetty. 2003. On the Need for Time series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Min. Knowl. Discov.* 7, 4 (Oct. 2003), 349–371. DOI:http://dx.doi.org/10.1023/A:1024988512476

H. Khazaei, J. Misic, and V.B. Misic. 2012. Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queuing Systems. *Parallel and Distributed Systems, IEEE Transactions on* 23, 5 (May 2012), 936–943. DOI:http://dx.doi.org/10.1109/TPDS.2011.199

Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy Katz. 2011. Napsac: Design and implementation of a power-proportional web cluster. *ACM SIGCOMM computer communication review* 41, 1 (2011), 102–108.

P. Lama and Xiaobo Zhou. 2012. Efficient Server Provisioning with Control for End-to-End Response Time Guarantee on Multitier Clusters. *IEEE Transactions on Parallel and Distributed Systems* 23, 1 (Jan 2012), 78–86. DOI:http://dx.doi.org/10.1109/TPDS.2011.88

Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing Public Cloud Providers. In *Proc. 10th ACM SIGCOMM Conf. on Internet Measurement (IMC 10)*. ACM, New York, NY, USA, 1–14. DOI:http://dx.doi.org/10.1145/1879141.1879143

H. Li and T. Yang. 2000. Queues with a variable number of servers. *European Journal of Operational Research* 124, 3 (2000), 615–628.

Harold C Lim, Shivnath Babu, and Jeffrey S Chase. 2010. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*. ACM, 1–10.

Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. 2009. Automated Control in Cloud Computing: Challenges and Opportunities. In *Proc. 1st Workshop on Automated Control for Datacenters and Clouds (ACDC 09)*. ACM, New York, NY, USA, 13–18. DOI:http://dx.doi.org/10.1145/1555271.1555275

Howard T. Liu and John A. Silvester. 1991. Dynamic resource allocation scheme for distributed heterogeneous computer systems. (July 9 1991). US Patent 5,031,089.

Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. 2014. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing* (2014), 1–34. DOI:http://dx.doi.org/10.1007/s10723-014-9314-7

A. Hasan Mahmud, Yuxiong He, and Shaolei Ren. 2014. BATS: Budget-constrained Autoscaling for Cloud Performance Optimization. *SIGMETRICS Perform. Eval. Rev.* 42, 1 (June 2014), 563–564.

Simon J Malkowski, Markus Hedwig, Jack Li, Calton Pu, and Dirk Neumann. 2011. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 131–140.

Ming Mao, Jie Li, and M. Humphrey. 2010. Cloud auto-scaling with deadline and budget constraints. In *Proc. 11th IEEE/ACM Int. Conf. on Grid Computing (GRID 10)*. 41–48. DOI:http://dx.doi.org/10.1109/GRID.2010.5697966

Shicong Meng, Ling Liu, and Vijayaraghavan Soundararajan. 2010. Tide: achieving self-scaling in virtualized datacenter management middleware. In *Proc. 11th Int. Middleware Conf. Industrial Track (Middleware Industrial Track 10)*. ACM, New York, NY, USA, 17–22. DOI:http://dx.doi.org/10.1145/1891719.1891722

Tran Ngoc Minh, Lex Wolters, and Dick Epema. 2010. A realistic integrated model of parallel system workloads. In *The 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 464–473.

Domas Mituzas. 2007. Page view statistics for Wikimedia projects. (2007). http://dumps.wikimedia.org/other/pagecounts-raw/ http://dumps.wikimedia.org/other/pagecounts-raw/[Online; accessed 2014-11-20].

Arkadi Nemirovski and Alexander Shapiro. 2006. Scenario Approximations of Chance Constraints. In *Probabilistic and Randomized Methods for Design under Uncertainty*, Giuseppe Calafiore and Fabrizio Dabbene (Eds.). Springer London, 3–47. DOI:http://dx.doi.org/10.1007/1-84628-095-8_1

Arkadi Nemirovski and Alexander Shapiro. 2007. Convex Approximations of Chance Constrained Programs. *SIAM Journal on Optimization* 17, 4 (2007), 969–996. DOI:http://dx.doi.org/10.1137/050622328

Marco A.S. Netto, Carlos Cardonha, Renato L.F. Cunha, and Marcos D. Assunção. 2014. Evaluating Auto-scaling Strategies for Cloud Computing Environments. In *Proc. IEEE 21st Int. Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS 14)*. 1–10.

Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. 2013. AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service. In *Proc. 10th Int. Conf. on Autonomic Computing (ICAC 13)*. USENIX, San Jose, CA, 69–82. https://www.usenix.org/conference/icac13/technical-sessions/presentation/nguyen

Jay Palat. 2012. Introducing vagrant. *Linux Journal* 2012, 220 (2012), 2.

Alessandro Vittorio Papadopoulos, Cristian Klein, Martina Maggio, Jonas Dürango, Manfred Dellkrantz, Francisco Hernández-Rodriguez, Erik Elmroth, and Karl-Erik Årzén. 2016. Control-Based Load-Balancing Techniques: Analysis and Performance Evaluation via a Randomized Optimization Approach. *Control Engineering Practice* 52 (2016), 24–34. DOI:http://dx.doi.org/10.1016/j.conengprac.2016.03.020

Alessandro Vittorio Papadopoulos and Maria Prandini. 2014. Model Reduction of Switched Affine Systems: A Method Based on Balanced Truncation and Randomized Optimization. In *Proc. 17th Int. Conf. on Hybrid Systems: Computation and Control (HSCC 14)*. ACM, New York, NY, USA, 113–122. DOI:http://dx.doi.org/10.1145/2562059.2562131

Alessandro Vittorio Papadopoulos and Maria Prandini. 2016. Model reduction of switched affine systems. *Automatica* 70 (2016), 57–65. DOI:http://dx.doi.org/10.1016/j.automatica.2016.03.019

Christian Papauschek. 2013. Real-world performance of the Play framework on EC2. (2013). http://blog.papauschek.com/2013/04/real-world-performance-of-the-play-framework-on-ec2/ http://blog.papauschek.com/2013/04/real-world-performance-of-the-play-framework-on-ec2/[Online; accessed 2014-11-24].

John Payne. 2014. C-MART:Benchmarking the Cloud. (2014). http://theone.ece.cmu.edu/cmart/ http://theone.ece.cmu.edu/cmart/[Online; accessed 2014-11-20].

András Prékopa. 2003. Probabilistic programming. In *Stochastic Programming (Handbooks in Operations Research and Management Science)*, A. Ruszczyǹski and A. Shapiro (Eds.), Vol. 10. Elsevier, London, UK, 267–351. DOI:http://dx.doi.org/10.1016/S0927-0507(03)10005-9

Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*. ACM, New York, NY, USA, 7:1–7:13. DOI:http://dx.doi.org/10.1145/2391229.2391236

Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *Proc. 2011 IEEE 4th Int. Conf. on Cloud Computing (CLOUD 11)*. IEEE Computer Society, Washington, DC, USA, 500–507. DOI:http://dx.doi.org/10.1109/CLOUD.2011.42

Rahul Singh, Upendra Sharma, Emmanuel Cecchet, and Prashant Shenoy. 2010. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. 7th Int. Conf. on Autonomic Computing (ICAC 10)*. ACM, New York, NY, USA, 21–30. DOI:http://dx.doi.org/10.1145/1809049.1809053

R. Sturm, W. Morris, and M. Jander. 2000. *Foundations of Service Level Management*. SAMS.

Andrew Turner, Andrew Fox, John Payne, and Hyong S Kim. 2013. C-mart: Benchmarking the cloud. *IEEE Trans. on Parallel and Distributed Systems* 24, 6 (2013), 1256–1266.

Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, and Pawan Goyal. 2005. Dynamic provisioning of multi-tier internet applications. In *Proc. 2nd Int. Conf. on Autonomic Computing (ICAC 05)*. 217–228. DOI:http://dx.doi.org/10.1109/ICAC.2005.27

Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. 2008. Agile Dynamic Provisioning of Multi-tier Internet Applications. *ACM Trans. Auton. Adapt. Syst.* 3, 1, Article 1 (2008), 39 pages. DOI:http://dx.doi.org/10.1145/1342171.1342172

Amin Vahdat, Thomas Anderson, Michael Dahlin, Eshwar Belani, David Culler, Paul Eastham, and Chad Yoshikawa. 1998. WebOS: operating system services for wide area applications. In *Proc. 7th Int. Symposium on High Performance Distributed Computing*. 52–63. DOI:http://dx.doi.org/10.1109/HPDC.1998.709956

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 *Proc. 12th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing (CCGRID 12)*. IEEE Computer Society, Washington, DC, USA, 612–619. DOI:http://dx.doi.org/10.1109/CCGrid.2012.46

Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, and Calton Pu. 2013. Detecting transient bottlenecks in n-tier applications through fine-grained analysis. In *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 31–40.

John Wilkes. 2011. More Google Cluster Data. (2011). http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html[Online; accessed 2014-10-30].