Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey

CHENHAO QU, The University of Melbourne, Australia
RODRIGO N. CALHEIROS, Western Sydney University, Australia
RAJKUMAR BUYYA, The University of Melbourne, Australia

Web application providers have been migrating their applications to cloud data centers, attracted by the emerging cloud computing paradigm. One of the appealing features of the cloud is elasticity. It allows cloud users to acquire or release computing resources on demand, which enables web application providers to automatically scale the resources provisioned to their applications without human intervention under a dynamic workload to minimize resource cost while satisfying Quality of Service (QoS) requirements. In this article, we comprehensively analyze the challenges that remain in auto-scaling web applications in clouds and review the developments in this field. We present a taxonomy of auto-scalers according to the identified challenges and key properties. We analyze the surveyed works and map them to the taxonomy to identify the weaknesses in this field. Moreover, based on the analysis, we propose new future directions that can be explored in this area.

CCS Concepts: • **Software and its engineering** → *Cloud computing*; • **Networks** → Cloud computing; • **Computer systems organization** → Cloud computing;

Additional Key Words and Phrases: Auto-scaling, web application, cloud computing

ACM Reference format:
Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. 2018. Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey. *ACM Comput. Surv.* 51, 4, Article 73 (July 2018), 33 pages.
https://doi.org/10.1145/3148149

1 INTRODUCTION

Cloud computing is the emerging paradigm for offering computing resources and applications as subscription-oriented services on a pay-as-you-go basis. One of its features, called elasticity, allows users to dynamically acquire and release the right amount of computing resources according to their needs. Elasticity is continuously attracting web application providers to move their applications into clouds.

To efficiently utilize elasticity of clouds, it is vital to automatically and in a timely manner provision and deprovision cloud resources without human intervention, since overprovisioning leads to resource wastage and extra monetary cost, while underprovisioning causes performance degradation and violation of service-level agreement (SLA). This mechanism of dynamically acquiring or releasing resources to meet QoS requirements is called auto-scaling.

This research is supported by a Future Fellowship grant from the Australian Research Council (ARC).
Authors' addresses: C. Qu and R. Buyya, The University of Melbourne, Parkville, VIC, 3010; emails: qchenhao@gmail.com, rbuyya@unimelb.edu.au; R. N. Calheiros, Western Sydney University, Parramatta, NSW, 2150; email: R.Calheiros@westernsydney.edu.au.

https://doi.org/10.1145/3148149

Designing and implementing an efficient general-purpose auto-scaler for web applications is a challenging task due to various factors, such as dynamic workload characteristics, diverse application resource requirements, and complex cloud resources and pricing models. In this article, we aim to comprehensively analyze the challenges in the implementation of an auto-scaler in clouds and review the developments for researchers that are new to this field. We present a taxonomy regarding the various challenges and key properties of auto-scaling web applications. We compare the existing works deployed by infrastructure providers and application service providers and map them to the taxonomy to discuss their strengths and weaknesses. Based on the analysis, we also propose promising future directions that can be pursued by researchers to improve the state of the art.

Lorido-Botran et al. [70] have already written a survey about this topic. However, their major focus is on resource estimation techniques. Different from them, our work provides comprehensive discussions about all the primary challenges in the topic and also introduces new developments that occurred since their work was published.

The rest of the article is organized as follows. In Section 2, we describe our definition of the auto-scaling problem for web applications and list its major challenges that need to be addressed when trying to implement an auto-scaler. After that, we present a taxonomy regarding the existing auto-scalers. From Section 4 to Section 12, we introduce and compare how existing auto-scaling techniques tackle the listed challenges. After that, in Section 13, we discuss the gaps of current solutions and present some promising future research directions. Finally, we summarize the findings and conclude the article.

2 PROBLEM DEFINITION AND CHALLENGES

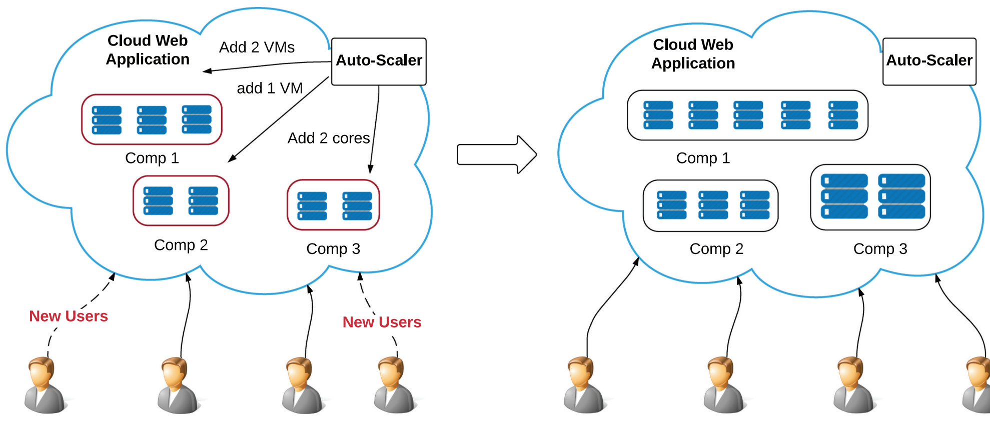
In a single cloud, the auto-scaling problem for web applications can be defined as how to autonomously and dynamically provision and deprovision a set of resources to cater to fluctuant application workloads without human intervention so that the resource cost is minimized and application SLAs or service-level objectives (SLOs) are satisfied. Figure 1 illustrates typical auto-scaling scenarios. In Figure 1(a), due to an increase in requests, the available resources are in congestion, and thus, the auto-scaler decides to provision certain resources by either scaling out (launching more VMs) or scaling up (adding resources to existing VMs) each application component. Oppositely, in Figure 1(b), the auto-scaler deprovisions some resources from each component by either scaling in (shutting down some VMs) or scaling down (removing resources from existing VMs), when the amount of requests has decreased.

This is a classic automatic control problem, which demands a controller that dynamically tunes the type of resources and the amount of resources allocated to reach certain performance goals, reflected as the SLA. Specifically, it is commonly abstracted as a MAPE (Monitoring, Analysis, Planning, and Execution) control loop [63]. The control cycle continuously repeats itself over time.

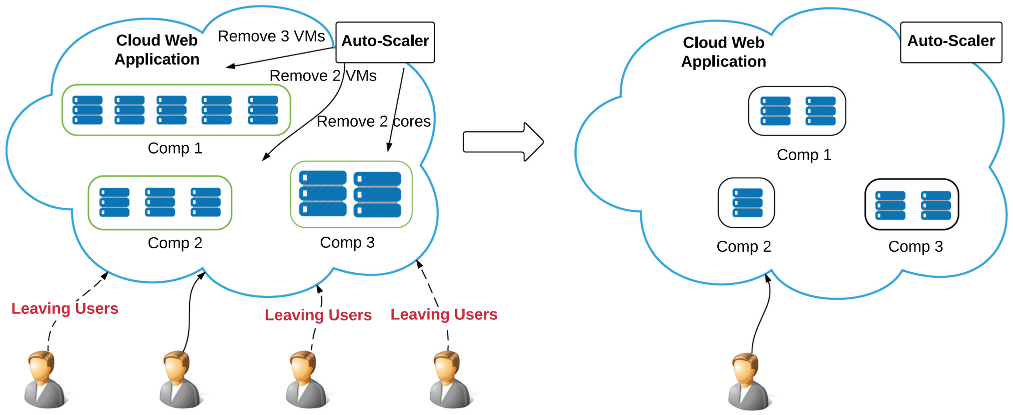
The biggest challenges of the problem lie in each phase of the loop as shown in Figure 2. We briefly explain each phase and summarize the individual challenges faced by auto-scaler designers in the following paragraphs.

Monitoring. The auto-scaler needs to monitor some performance indicators to determine whether scaling operations are necessary and how they should be performed.

- Performance indicators: selection of the right performance indicators is vital to the success of an auto-scaler. The decision is often affected by many factors, such as application characteristics, monitoring cost, SLA, and the control algorithm itself.
- Monitoring interval: the monitoring interval determines the sensitivity of an auto-scaler. However, very short monitoring intervals result in high monitoring cost both regarding



(a) Scaling Out/Up



(b) Scaling In/Down

Fig. 1. Typical auto-scaling scenarios—right sizing of resources.

computing resources and financial cost, and it is likely to cause oscillations in the auto-scaler. Therefore, it is important to tune this parameter to achieve balanced performance.

Analysis. During the analysis phase, the auto-scaler determines whether it is necessary to perform scaling actions based on the monitored information.

- Scaling timing: the auto-scaler first needs to decide when to perform the scaling actions. It either can proactively provision/deprovision resources ahead of the workload changes if they are predictable since the provision/deprovision process takes considerable time or can perform actions reactively when workload change has already happened.
- Workload prediction: if the auto-scaler chooses to scale the application proactively, how to accurately predict the future workload is a challenging task.
- Adaptivity to changes: sometimes the workload and the application may undergo substantial changes. The auto-scaler should be aware of the changes and in a timely manner adapt its model and settings to the new situation.
- Oscillation mitigation: scaling oscillation means the auto-scaler frequently performs opposite actions within a short period (i.e., acquiring resources and then releasing resources or

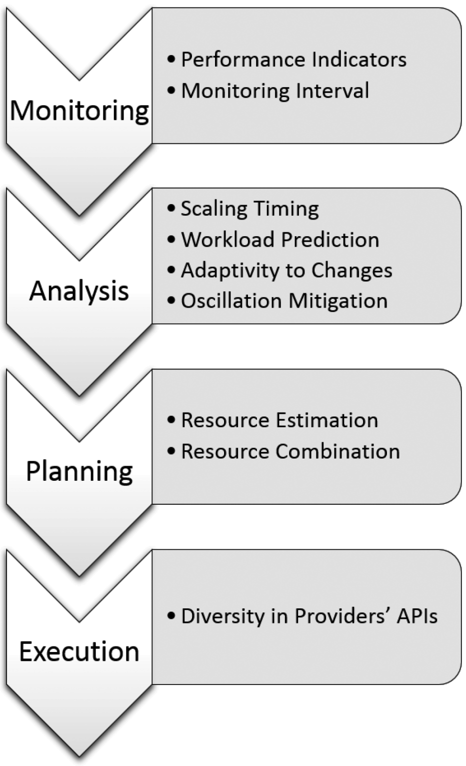


Fig. 2. The challenges of auto-scaling web applications in each phase of the MAPE loop.

vice versa). This situation should be prevented as it results in resource wastage and more SLA violations.

Planning. The planning phase estimates how many resources in total should be provisioned/deprovisioned in the next scaling action. It should also optimize the composition of resources to minimize financial cost.

- Resource estimation: the planning phase should be able to estimate how many resources are just enough to handle the current or incoming workload. This is a difficult task as the auto-scaler needs to determine required resources quickly without being able to actually execute the scaling plan to observe the real application performance, and it has to take the specific application deployment model into account in this process.
- Resource combination: to provision resources, the auto-scaler can resort to both vertical scaling and horizontal scaling. If horizontal scaling is employed, as the cloud providers offer various types of VMs, the auto-scaler should choose one of them to host the application. Another important factor is the pricing model of cloud resources. Whether to utilize on-demand, reserved, or rebated resources significantly affects the total resource cost. All these factors form a huge optimization space, which is challenging to be solved efficiently in a short time.

Execution. The execution phase is responsible for actually executing the scaling plan to provision/deprovision the resources. It is straightforward and can be implemented by calling cloud

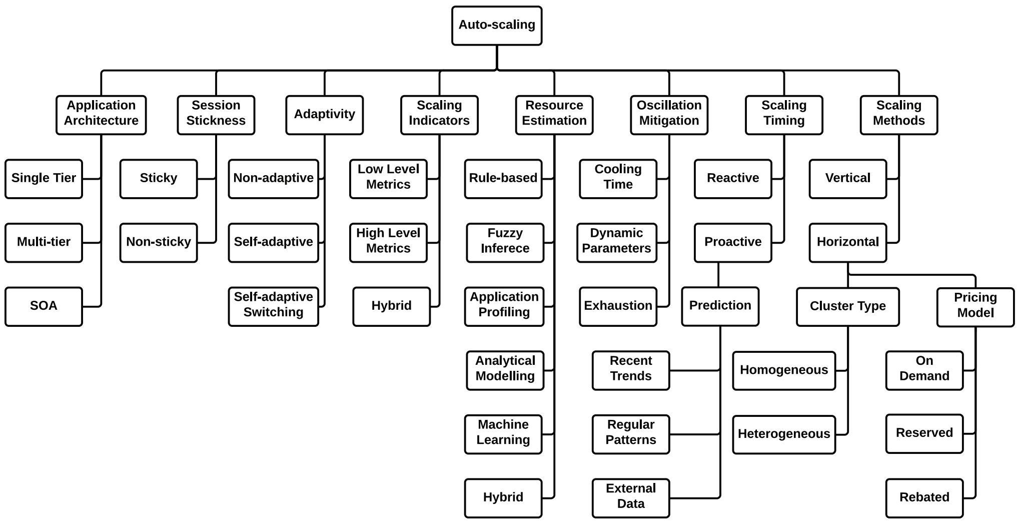


Fig. 3. The taxonomy for auto-scaling web applications in clouds.

providers' APIs. However, from an engineering point of view, being able to support APIs of different providers is a challenging task.

If the application is supposed to be deployed in multiple data centers, it is also important to identify which data center is most cost-efficient to serve the requests from certain groups of users without violating SLAs. Therefore, in addition to provisioning just enough resources during runtime, the auto-scaling problem becomes a mixed problem of data center selection, geographical load balancing, and resource provisioning in the multicloud scenario. The auto-scaler should dynamically direct users from certain areas to specific data centers and ensure that enough resources are provisioned in each of the involved data centers to handle the incoming requests. To minimize cost in this scenario, considering all available choices in these tasks, it generally requires solving a NP-hard problem to generate the provision plan.

3 TAXONOMY

Figure 3 illustrates our proposed taxonomy for auto-scaling web applications in clouds. It classifies the existing works based on the identified challenges in each of the MAPE phases in Section 2 and their targeted environment. Particularly, the taxonomy covers the following aspects of auto-scaling:

- Application Architecture: the architecture of the web application that the auto-scaler is managing
- Session Stickiness: whether the auto-scaler supports sticky sessions
- Adaptivity: whether and how the auto-scaler adapts to changes of workload and application
- Scaling Indicators: what metrics are monitored and measured to make scaling decisions
- Resource Estimation: how the auto-scaler estimates the amount of resources needed to handle the workload
- Oscillation Mitigation: how the auto-scaler reduces the chance of provisioning oscillation
- Scaling Timing: whether the auto-scaler supports proactive scaling of applications and how it predicts future workload

- Scaling Methods: how the auto-scaler decides the methods to provision resources and what combination of resources are provisioned to the application
- Environment: whether the auto-scaler works in a single- or multicloud environment

Note that the considered characteristics of auto-scaling may be correlated, especially for resource estimation and oscillation mitigation. Resource estimation is affected by many other characteristics, such as application architecture, adaptivity, and scaling indicators. Oscillation mitigation is often influenced by the choice of resource estimation model and the scaling methods used. An existing approach generally spans different subcategories and is discussed in each of them (i.e., an auto-scaler is built in a single-cloud environment for multitier applications and employs proactive scaling with machine-learning resource estimation techniques). Note that this taxonomy is based on features and thus does not reflect the relative performance of the proposed approaches. Actually, because the surveyed works target diverse workload patterns, application architectures, and pricing models, there is no single answer to the question of which approach generally performs the best [77].

In the following sections (from Section 4 to Section 12), we introduce and compare existing auto-scalers according to this taxonomy.

4 APPLICATION ARCHITECTURES

There are three types of web application architectures mentioned in the literature: namely, single-tier, multitier, and service-oriented architecture.

4.1 Single Tier/Single Service

A tier is the minimum separately deployable component in a layered software stack. In a production deployment, within a tier, a load balancer is used to balance and dispatch load among the instances of the same tier. Single-tier architecture by definition is the architecture in which an application is composed of only one tier. Relatively, the architecture with multiple connected software tiers is called multitier architecture. Instead of calling single tier as an application architecture, it is more accurate to think of it as the smallest granularity that can be possibly managed by an auto-scaler, since hardly any web application is composed of only one tier.

Nowadays, web applications are becoming more and more complicated and deviate from the traditional multitier architecture. In those cases, the fundamental scaling component is often referred to as a service/microservice. The majority of existing auto-scalers separately manage each single tier or service within an application instead of considering it as a whole. This method is both simple and general. However, it often results in globally suboptimal resource provisioning as it requires dividing the SLA requirements of the overall application into subrequirements of each tier or service, which is often a challenging and subjective task.

4.2 Multitier

Multitier applications, as introduced, are composed of sequentially connected tiers. At each tier, the request either relies on the downstream tier to complete its processing or is returned to the upstream tier and finally to the user.

A widely adopted architecture of this type usually consists of three tiers: one front-end, one application logic, and one database tier. The database tier is often considered dynamically unscalable and ignored by auto-scalers.

Many works have targeted multitier applications. Some of them employ the divide-and-conquer approach that breaks overall SLA into SLA of each tier, such as the works conducted by Urgaonkar et al. [99], Singh et al. [92], Iqbal et al. [52], Malkowski et al. [71], Upendra et al. [98], and Gergin et al. [34]. Others consider SLA of the whole application and provision resources to each tier holistically. This strategy requires more effort in modeling and estimating resource consumption using sophisticated queuing networks and machine-learning techniques as discussed in Section 8, and the resulting auto-scalers are only applicable to multitier applications. Important works of this category include approaches proposed by Zhang et al. [109], Jung et al. [59], Padala et al. [76], Lama et al. [64, 65], Sharma et al. [90], Han et al. [46], and Kaur et al. [62].

4.3 Service-Based Architectures

Service-based architectures have now become the dominant paradigm for large web applications, such as the Amazon e-commerce website and Facebook. In these architectures, applications are composed of standalone services that interact with each other through predefined APIs. More importantly, services are not necessarily connected sequentially as in multitier applications. Service-based applications are commonly abstracted as directed graphs with each node representing services and directed edges representing their interactions. The service-based architectures can be further classified into Service-Oriented Architecture (SOA) and Microservices Architecture. Richards et al. [81] provided a full discussion of their commonalities and differences, which is out of the scope of this article. From the perspective of auto-scaling, they pose the same question about how to provision individual services so that the QoS requirements of the aggregated application can be satisfied.

Due to the complexity of service-based architectures, it is difficult to manage resource provisioning of all the services holistically. Therefore, industry and most works employ the divide-and-conquer approach. Differently, Jiang et al. [55] proposed a method that can satisfy SLA of the whole SOA application. It requires each service to estimate the change of its response time if one instance is added or removed from it. After that, the system aggregates the estimations and chooses the operations that will minimize the overall response time.

5 SESSION STICKINESS

A session is a series of interactions between a client and the application. After each operation, the client waits for the reply given by the application and then proceeds. To ensure a seamless experience, it is necessary to keep the intermediate statuses of clients during their sessions. Otherwise, the operations conducted by the clients will be lost and they have to repeat the previous operations to proceed. Taking a social network application as an example, a session may involve the following operations: the client first accesses the home page and then logs into the application; after that, he or she performs several actions such as viewing his or her and friends' timeline, uploading photos, and updating his or her status, before quitting the application.

This session-based access pattern has caused issues in efficiently utilizing elastic resources in the cloud because the stateful nature of the session forces the user to be connected to the same server each time he or she submits a request within the session if the session data is stored in the server. Such sessions are considered sticky. They limit the ability of the auto-scaler to terminate underutilized instances when there are still unfinished sessions handled by them. Therefore, it is regarded as a prerequisite to transforming stateful servers into stateless servers before an auto-scaler can manage them.

There are multiple ways to achieve this, and a complete introduction to them is out of the scope of this article. The most adopted approach is to move the session data out of the web servers and store them either at the user side or in a shared Memcached cluster.

Though most auto-scalers require the scaling cluster to be stateless, some auto-scalers are designed to handle stateful instances. Chieu et al. [17] proposed an auto-scaler based on the number of active sessions in each server. Their approach requires a server to clear all its active sessions before it can be terminated. Grozev et al. [43] proposed a better approach by integrating a similar auto-scaler with a load balancing algorithm that consolidates sessions within as few instances as possible.

6 ADAPTIVITY

Auto-scalers fall in the realm of control systems. As stated in the introduction, their operation involves tuning resources provisioned to the application to reach the target performance. One major issue coupled with the design of a control system is its adaptivity to changes. As in dynamic production environments, workload characteristics, and even the application itself, can change at any moment. Therefore, adaptivity is important to auto-scalers. Based on the level of adaptivity, we classify the existing works into three categories.

6.1 Nonadaptive

In the nonadaptive approaches, the control model is predefined, and they make decisions purely based on the current input. Examples are the rule-based approaches employed by the industry, such as Amazon Auto-Scaling service [6]. They require the user to define a set of scaling-out and scaling-in conditions and actions offline. During production time, the auto-scaler makes scaling decisions only when the conditions are met. They do not allow automatic adjustment of the settings during production. When using this kind of auto-scaler, the users often need to direct considerable efforts in offline testing to find a suitable configuration.

6.2 Self-Adaptive

Self-adaptive auto-scalers are superior to their nonadaptive counterparts. Though the core control models in them are fixed as well, they are capable of autonomously tuning themselves according to real-time quality of the control actions observed. In this way, the designer only needs to determine the core control model, such as whether it is linear or quadratic, and the auto-scaler will adjust and evolve itself to meet the target performance. This feature can be implemented through extending existing self-adaptive control frameworks in control theory, such as the works done by Kamra et al. [61], Kalyvianaki et al. [60], and Grimaldi et al. [41]. Self-adaptivity can also be realized through dynamic measurement or correction of parameters in analytical models and machine-learning approaches, such as reinforcement learning and regression. Detailed explanations of them are given in Section 8.

The benefit of introducing self-adaptivity is that it significantly reduces the amount of offline preparation required to utilize an auto-scaler. Furthermore, once substantial changes are detected, self-adaptive approaches can autonomously abort the current model and retrain itself, thus reducing maintenance efforts. Their primary drawback is that it usually takes time for them to converge to a good model and the application will suffer from bad performance during the early stage of training.

6.3 Self-Adaptive Switching

Beyond the utilization of a single self-adaptive module, some auto-scalers have employed a more adaptive framework, which we call self-adaptive switching. In these auto-scalers, they concurrently connect multiple nonadaptive or self-adaptive controllers and actively switch control between controllers based on their observed performance on the application. The included self-adaptive controllers continuously tune themselves in parallel. However, at each moment, only the selected best controller can provision resources. Patikirikorala et al. [78] employed this approach, and Ali-Eldin et al. [4] proposed a self-adaptive switching approach based on the classification of the application workload characteristics, i.e., their periodicity and the burstiness. Chen et al. [16]

proposed an approach that trains multiple resource estimation models and dynamically selects the one that performs the best.

7 SCALING INDICATORS

The actions of auto-scalers are based on performance indicators of the application obtained in the monitoring phase. These indicators are produced and monitored at different levels of the system hierarchy from low-level metrics at the physical/hypervisor level to high-level metrics at the application level.

7.1 Low-Level Metrics

Low-level metrics, in the context of this survey, consist of server information monitored at the physical server/virtual machine layer by hypervisors, such as utilization of CPU, memory, network resources, memory swap, and cache miss rate. These data can be obtained through a monitoring platform of the cloud provider or from monitoring tools for operating systems. However, it is a nontrivial task to accurately infer the observed application performance merely according to the low-level metrics, and therefore, it makes it a difficult task to make sure that the SLA can be met faithfully with the available resources.

Designing an auto-scaler solely based on low-level performance indicators is possible. The simplest solution is to use the utilization of CPU and other physical resources as indicators and scale resources horizontally or vertically to maintain the overall utilization within a predefined upper and lower bound. Industry systems widely adopt this approach. If the auto-scaler of this kind only supports horizontal scaling, it can be utilized by both cloud providers and service providers. Otherwise, only cloud providers can deploy it as service providers lack the control over underlying resource allocation in the current cloud market.

7.2 High-Level Metrics

High-level metrics are performance indicators observed at the application layer. Only auto-scalers deployed by service providers can utilize these metrics as they are not visible to cloud providers. Useful metrics to auto-scaling include request rate, average response time, session creation rate, throughput, service time, and request mix.

Some metrics, such as request rate, average response time, throughput, and session creation rate, are easy to measure. They alone enable operation of an auto-scaler as they are direct indicators of the performance of the applications and can be used to trigger scaling operations. The easiest method to construct one is to replace utilization metrics in the simple auto-scaler mentioned in the previous section with any of such high-level metrics. However, this approach is not able to accurately estimate the amount of resources needed and often over- or underprovisions resources.

Some auto-scalers require obtaining information about request service time and request mix [62, 92, 109] to estimate how much resources are needed. Different from the previous metrics, they cannot directly trigger scaling actions but can be used to support making efficient scaling plans. In addition, these metrics are not straightforward to measure.

Service time is the time a server spends on processing the request, which is widely used in the queuing models to approximate the average response time or sojourn time. Except for a few works [34, 46] that assume this metric as known a priori, to accurately measure it, either offline profiling [79] or support from the application [7] is required. Therefore, instead of directly probing it, some works use other approaches to approximate it. Kaur et al. [62] mentioned the use of past server logs to infer the mean service time. Gandhi et al. [31] employed Kalman filters to estimate service time during runtime. Zhang et al. [109] used a regression method to make the approximation. Jiang et al. [56] resorted to profiling each server when it is first online using a small workload without concurrency and then estimating service time through queuing theory. In another work, Jiang et al. [55] utilized a feedback control loop to adjust the estimation of service time at runtime.

Request mix is hard to measure because an understanding of the application is essential to distinguish different types of requests. Designing a mechanism to accurately classify various types of requests from outside of the application itself is an interesting and challenging problem to be explored.

7.3 Hybrid Metrics

In some auto-scalers, both high-level and low-level metrics are monitored. A common combination is to observe request rate, response time, and utilization of resources. Some works [31, 79] monitor them because the queuing models employed for resource estimation require them as input. Some [24, 27, 58, 76, 107] use these hybrid metrics to dynamically build a model relating specific application performance to physical resource usage through online profiling, statistical, and machine-learning approaches, thus increasing the accuracy of resource estimation without constructing complex analytical models. Another important reason to monitor request rate along with low-level metrics is to conduct future workload prediction [24, 85].

Besides low-level and high-level metrics from the platform and application, other factors outside may also play a significant role. For example, Frey et al. [28], in their fuzzy-based approach, utilize other related data, such as weather and political events, to predict workload intensity.

8 RESOURCE ESTIMATION

Resource estimation lies in the core of auto-scaling as it determines the efficiency of resource provisioning. It aims to identify the minimum amount of computing resources required to process a workload to determine whether and how to perform scaling operations. Accurate resource estimation allows the auto-scaler to quickly converge to the optimal resource provisioning. On the other hand, estimation errors either result in insufficient provisioning, which leads to a prolonged provisioning process and increased SLA violations, or overprovisioning, which incurs more cost.

Various attempts have been made to develop resource estimation models from basic approaches to methods with sophisticated models. We categorize them into six groups, namely, rule-based, fuzzy inference, application profiling, analytical modeling, machine-learning, and hybrid approaches. In the following sections, we explain and compare the existing approaches in each group.

8.1 Rule-Based Approaches

Rule-based approaches are widely adopted by industry auto-scalers, such as Amazon Auto-Scaling Service [6]. Its kernel is a set of predefined rules consisting of triggering conditions and corresponding actions, such as "If CPU utilization reaches 70%, add two instances," and "If CPU utilization decreases below 40%, remove one instance." As stated in Section 7, users can use any metrics, low level or high level, to define the triggering conditions, and the control target of the auto-scaler is usually to maintain the concerned parameters within the predefined upper and lower threshold. Theoretically, simple rule-based approaches involve no accurate resource estimation, only empirical estimation, which is hardcoded in the action part of the rule as adding or removing a certain amount or percentage of instances. As the simplest version of auto-scaling, it commonly serves as a benchmark for comparison and is used as the basic scaling framework for works that focus on other aspects of auto-scaling, such as the work done be Dawoud et al. [20], which aims to compare vertical scaling and horizontal scaling, and by Rui et al. [86], which considers all possible scaling methods, or prototyping works, like the one carried out by Iqbal et al. [50].

Though a simple rule-based auto-scaler is easy to implement, it has two significant drawbacks. The first is that it requires understanding of the application characteristics and expert knowledge to determine the thresholds and proper actions. Al-Haidari et al. [1] conducted a study to show that these parameters significantly affect the auto-scaler's performance. The second is that it cannot adapt itself when dynamic changes occur to the workload or to the application.

Hardcoded numbers of instances to scale out and scale in, called step sizes, would become inappropriate when the workload changes dramatically. For example, if the application is provisioned by four instances at the start, adding one instance will boost 25% of the capability. After a while, the cluster will have increased to 10 instances due to workload surge, and adding one instance in this case only increases 10% of capacity. Improvements are made to the basic model using adaptive step sizes. Netto et al. [72] proposed an approach that decides the step size holistically at runtime based on the upper threshold, the lower threshold, and the current utilization. It first deduces the upper and lower bounds, respectively, for step sizes of scaling-out and scaling-in operations to prevent oscillation and then scales the step sizes using a fixed parameter representing the aggressiveness of the auto-scaler determined by the user. They reported that the adaptive strategy performed best for a bursty workload but led to limited improvements for other types of workloads. Cunha et al. [18] employed a similar approach. However, in their approach, the aggressiveness parameter is also dynamically tunable according to QoS requirements.

In addition to the step size, fixed thresholds also could cause inefficient resource utilization. For instance, the thresholds of 70% and 40% may be suitable for a small number of instances but are inefficient for large clusters as a single instance has a subtle impact on the overall utilization and a lot of instances actually can be removed before the overall usage reaches the 40% lower bound. A solution to mitigate this problem is also to make the thresholds dynamic. Lim et al. [67, 68] used this approach.

RightScale et al. [82] proposed another important variation of the simple rule-based approach. Its core idea is to let each instance decide whether to shrink or expand the cluster according to predefined rules and then utilize a majority voting approach to make the final decision. Calcavecchia et al. [12] also proposed a decentralized rule-based approach. In their proposal, instances are connected as a P2P network. Each instance contacts its neighbors for their statuses and decides whether to remove itself or start a new instance in a particular probability derived from their statuses.

8.2 Fuzzy Inference

Fuzzy-based auto-scalers can be considered as advanced rule-based approaches as they rely on fuzzy inference, the core of which is a set of predefined If-Else rules, to make provisioning decisions. The major advantage of fuzzy inference compared to simple rule-based reasoning is that it allows users to use linguistic terms like "high, medium, low," instead of accurate numbers to define the conditions and actions, which makes it easier for human beings to effectively represent their knowledge (human expertise) about the target. Fuzzy inference works as follows: the inputs are first fuzzified using defined membership functions; then the fuzzified inputs are used to trigger the action parts in all the rules in parallel; the results of the rules are then combined and finally defuzzified as the output for control decisions. Representative approaches of this kind include the one proposed by Frey et al. [28] and the work conducted by Lama et al. [64]. Since manually designing the rule set is cumbersome and manual rule sets cannot in a timely manner handle environmental and application changes, fuzzy-based auto-scalers are commonly coupled with machine-learning techniques to automatically and dynamically learn the rule set [54, 58, 65]. Their details are introduced in Section 8.6.

8.3 Application Profiling

We define profiling as a process to test the saturation point of resources when running the specific application using a synthetic or recorded real workload. Application profiling is the simplest way to accurately acquire the knowledge of how many resources are just enough to handle the given workload intensity concurrently. Tests need to be conducted either offline or on the fly to profile an application.

Offline profiling can produce the complete spectrum of resource consumption under different levels of workload. With the obtained model, the auto-scaler can more precisely supervise the resource provisioning process. Upendra et al. [98], Gandhi et al. [33], Fernandez et al. [27], and Qu et al. [80] employed this approach. The drawback of it is that the profiling needs to be reconducted manually every time the application is updated.

Profiling can be carried out online to overcome this issue. However, the online environment prohibits the auto-scaler from fine-grainedly profiling the application as a VM should be put into service as soon as possible to cater to the increasing workload. Vasic et al. [100] proposed an approach that first profiles the application, then classifies the application signatures into different workload classes (number of machines needed). When changes happen to the application, the profiled new application characteristics are fed into the trained decision tree to realize quick resource provisioning by finding the closest resource allocation plan stored before. Nguyen et al. [73] relied on online profiling to derive a resource estimation model for each application tier. When profiling each tier, other tiers are provisioned with ample resources. In this way, one by one, models for all the tiers are obtained. Jiang et al. [56] proposed a quick online profiling technique for multitier applications by studying the correlation of resource requirements that different tiers pose on the same type of VM and the profile of a particular tier on that type of VM. This approach allows them to roughly deduce performance of the VM on each tier without actually running each tier on it. Thus, the newly acquired VM can be put into service at relatively quicker speed.

8.4 Analytical Modeling

Analytical modeling is a process of constructing mathematical models based on theory and analysis. For resource estimation problems in auto-scaling, the predominant models are built upon queuing theory.

In the generalized form, a queue can be represented as $A/S/C$, where A is the distribution of time interval between arrivals to the queue, S is the distribution of time required to process the job, and C stands for the number of servers. Common choices for A in the existing works are M (Markov), which means that arrivals follow the Poisson process, and G (General), which means the interarrival time has a general distribution. For S, the prominent alternatives are M (Markov), which represents exponentially distributed service time; D (Deterministic), which means the service time is fixed; and G (General), which means the service time has a general distribution. Detailed introduction of different types of queues is out of the scope of this article. Interested users can refer to this book [38].

For a single application, tier, or service, if the underlying servers are homogeneous, it is more convenient to abstract the whole application/tier/service as a single queue with one server. Kamra et al. [61], Villela et al. [101], Sharma et al. [90], Gandhi et al. [31, 32], and Gergin et al. [34] employed this method. Some auto-scalers described the cluster using a queue with multiple servers, like the works done by Ali-Eldin et al. [3], Jiang et al. [57], Aniello et al. [7], and Han et al. [46]. Other works modeled each server as a separate queue, such as the systems proposed by Doyle et al. [21], Urgaonkar et al. [99], Roy et al. [85], Ghanbari et al. [36], Kaur et al. [62], Spinner et al. [93], and Jiang et al. [55]. Bi et al. [9] proposed a hybrid model, in which the first tier is modeled as an M/M/c queue, while other tiers are modeled as M/M/1 queues. Different from the traditional queuing theory, Salah et al. [87] used an embedded Markov chain method to model the queuing system.

When the application involves multiple tiers or is composed of many services, single-layer queuing models are insufficient. Instead, a network of queues is needed to describe the components and their relations. These models are known as queuing networks. As introduced in Sections 4.2 and 4.3, to decide a number of resources in each component, there are two strategies. One is to divide the SLA into separate time portions and distribute them to each component. By this method, the queuing model for each component can be easily solved. However, it usually results in suboptimal solutions globally. Another method is to holistically provision resources to all the components to satisfy the SLA. Such method is more challenging as it is difficult and computationally heavy to find the optimal resource provisioning plan regarding a complex queuing network model.

Some models and methods have been proposed to tackle the challenge. Villela et al. [101] described the model as an optimization problem and used three different approximations to simplify it. Bi et al. [9] as well employed an optimization approach. Roy et al. [85] and Zhang et al. [109] utilized MVA (Mean Value Analysis), a widely adopted technique for computing expected queue lengths, waiting time at queuing nodes, and throughput in equilibrium for a closed queuing network, to anticipate the utilization at each tier under the particular provisioning. Han et al. [46] adopted a greedy approach that continuously adds/removes one server to/from the most/least utilized tier until the estimated capacity is just enough to serve the current load.

As mentioned in Section 7.2, some parameters in the queuing models are hard to measure directly, like service time. Therefore, the proposed auto-scalers should properly handle this issue. The detailed techniques were introduced in Section 7.2.

8.5 Machine Learning

Machine-learning techniques in resource estimation are applied to dynamically construct the model of resource consumption under a specific workload (online learning). In this way, different applications can utilize the auto-scalers without customized settings and preparations. They are also more robust to changes during production as the learning algorithm can self-adaptively adjust the model on the fly regarding any notable events. Online machine-learning algorithms are often implemented as feedback controllers to realize self-adaptive evolution. Though offline learning can also be used to fulfill the task, it inevitably involves human intervention and thus loses the benefit of using machine learning. For works that use offline learning—if there exist any—we classify them into the application profiling category.

Despite their easiness of usage and flexibility, machine-learning approaches suffer from a major drawback. It takes time for them to converge to a stable model and thus causes the auto-scaler to perform poorly during the active learning period. Certainly, the application performance is affected in this process. Furthermore, the time that is taken to converge is hard to predict and varies case by case and algorithm by algorithm.

Online learning used by the existing auto-scalers can be divided into two types: reinforcement learning and regression.

8.5.1 Reinforcement Learning. Reinforcement learning aims to let the software system learn how to react adaptively in a particular environment to maximize its gain or reward. It is suitable to tackle automatic control problems like auto-scaling [8, 11, 22, 23, 25, 37, 51, 66, 95, 107, 111]. For the auto-scaling problem, the learning algorithm's target is to generate a table specifying the best provision or deprovision action under each state. The learning process is similar to a trial-and-error approach. The learning algorithm chooses an individual operation and then observes the result. If the result is positive, the auto-scaler will be more likely to take the same action next time when it faces a similar situation.

The most used reinforcement learning algorithm in the auto-scaling literature is Q-learning. A detailed description of the Q-learning algorithm and its variations in auto-scaling can be found in Section 5.2 of the survey [70].

8.5.2 Regression. Regression estimates the relationship among variables. It produces a function based on observed data and then uses it to make predictions. Under the context of resource estimation, the auto-scaler can record system utilization, application performance, and the workload for regression. As the training proceeds and more data is available, the predicted results also become more accurate. Although regression requires the user to determine the function type first, for example, whether the relationship is linear or quadratic, in the case of auto-scaling web applications, it is usually safe to assume a linear function.

Chen et al. [14] used regression to dynamically build the CPU utilization model of Live Messenger given a number of active connections and login rate. The model is then used for resource provisioning. Bi et al. [9] employed smoothing splines nonlinear regression to predict mean performance under a certain amount of resources. Then they calculated the variance based on the estimated mean. After that they used a local polynomial (LOESS) regression to map mean performance to variance. Through this method, they found out that higher workload results in both mean and variance of the response time to increase. To detect sudden changes, they rely on conducting a statistical hypothesis test of the residual distribution in two time frames with probably different sizes. If the test result is statistically significant, the model needs to be retrained. Padala et al. [76] utilized auto-regressive moving-average (ARMA) to dynamically learn the relationship between resource allocation and application performance considering all resource types in all tiers. Gambi et al. [30] proposed an auto-scaler using a Kriging model. Kriging models are spatial data interpolators akin to radial basis functions. These models extend traditional regression with stochastic Gaussian processes. The major advantage of them is that they can converge quickly using fewer data samples. Grimaldi et al. [41] proposed a Proportional-Integral-Derivative (PID) controller that automatically tunes parameters to minimize integral squared error (ISE) based on a sequential quadratic programming model.

Yanggratoke et al. [105] proposed a hybrid approach using both offline learning and online learning. They first used a random forest model and traces from a testbed to train the baseline. Then they applied regression-based online learning to train the model for real-time resource estimation.

8.6 Hybrid Approaches

All the previously listed approaches have their pros and cons. Therefore, some works have integrated multiple methods together to perform resource estimation. We classify them as hybrid approaches and individually introduce them and the rationale behind such integration.

Rule-based approaches are inflexible when significant changes occur to applications and often require expert knowledge to design and test. However, if the rules can be constructed dynamically and adaptively by some learning techniques, such concern vanishes. Jing et al. [58] and Jamshidi et al. [54] proposed approaches that combine machine learning and fuzzy rule-based inference. They utilized machine learning to dynamically construct and adjust the rules in their fuzzy inference engine. Lama et al. [64] first proposed a fixed fuzzy-based auto-scaler with a self-adaptive component that dynamically tunes the output scaling factor. After that, they built another fuzzy inference system as a four-layer neural network [65] in which membership functions and rules can self-evolve as the time passes.

Some analytical queuing models require the observation of volatile metrics that are hard to measure directly. In these cases, a widely adopted solution is to use machine-learning approaches to estimate the concealed metrics dynamically. Gandhi et al. [31] adopted the Kalman filter to assess the average service time, background utilization, and end-to-end network latency. Zhang et al. [109] employed application profiling and regression to learn the relationship of average CPU utilization and average service time at each tier under a given request mix to solve their queuing network model using Mean Value Analysis.

To mitigate the drawback of machine-learning approaches, which are slow to converge and may cause plenty of SLA violations, another model can be used to substitute the learning model temporarily and then shift it back after the learning process has converged. Tesauro et al. [96] and Gambi et al. [29] proposed this kind of approach. Both of them utilized an analytical queuing model for temporary resource estimation during the training period. Tesauro et al. [96] employed reinforcement learning, while Gambi et al. [29] adopted a Kriging-based controller for training.

To make the model more general purpose, Chen et al. [16] first applied feature selection and then utilized the selected features to train different models using various machine-learning algorithms. The most accurate model is selected at runtime as the model for resource provisioning.

9 OSCILLATION MITIGATION

Oscillation is the situation in which an auto-scaler continuously performs opposite scaling operations, such as provisioning two VMs and then in short time deprovisioning two VMs. It happens when monitoring and scaling operations are too frequent, or the auto-scaler is poorly configured. Such concerns are magnified when dealing with rule-based auto-scalers whose resource estimations are relatively empirical and coarse grained. If the scaling thresholds are poorly configured, oscillation is likely to happen. For example, suppose the scale-out threshold is set to 70%, the scale-in threshold is set to 50%, and the current utilization is 71% with only one instance running; the auto-scaler will add one more instance to the cluster to reduce the utilization. It then quickly drops to 35%, which is below the scale-down threshold, thus causing oscillation.

9.1 Cooling Time

One common solution adopted in practice [6] to mitigate oscillation is to conservatively wait a fixed minimum amount of time between each scaling operation. The time is set by users and is known as the cooling time. It should be set to at least the time taken to acquire, boot up, and configure the VM. Such method is simple but effective to avoid frequent scaling operations. However, setting a long cooling time will also result in more SLA violations as the application cannot be scaled out as quickly as before. Besides, it cannot handle the situation in which the auto-scaler is poorly configured.

Another way of setting the cooling time is to refine the scaling condition. Suppose the monitoring interval of the auto-scaler is 1 minute; we can achieve a prolonged scaling interval by setting the scaling trigger to how many times the monitored value exceeds the defined threshold consecutively.

9.2 Dynamic Parameters

Besides static cooling time, approaches that dynamically adjust parameters to reduce the possibility of causing oscillation have been proposed.

Lim et al. [67, 68] described an approach through dynamically tuning triggering thresholds for scale-down operations. The core idea is to increase the scale-down threshold when more resources are allocated to decrease the target utilization range and vice versa when resources are deallocated, which can effectively mitigate oscillation if the application resource requirement varies significantly during peak time and nonpeak time. Usually, during nonpeak time, a large target range is desirable to avoid the situation described in the poorly configured example, while during peak hours, a small target range is preferred to keep the utilization as close to the scale-out threshold as possible.

Bodik et al. [10] introduced a mechanism that they call "hysteresis parameters," to reduce oscillation. These parameters control how quickly the controller provisions and deprovisions resources. They are determined by simulations using Pegasus, an algorithm that compares different control settings to search the suitable one. Padala et al. [76] used a stability factor to adjust the aggressiveness of the approach. As the factor increases, the control objective will be more affected by the previous allocation. As a result, the auto-scaler responds more slowly to the resulting errors caused by the previous actions in the following resource scaling windows and thus reduces oscillations. Lama et al. [64] employed a similar approach on their fuzzy-based auto-scaler. Their approach is more advanced and flexible as the factor is self-tunable during runtime according to the resulting errors.

Ali-Eldin et al. [2, 3] utilized a parameter to dynamically tune the prediction horizon of the workload prediction algorithm according to its real-time error rate. In this way, the auto-scaler can reduce the number of scaling oscillations caused by inaccurate workload predictions.

9.3 Exhaustion

The above methods are only capable of mitigating the possibility of oscillations. If in theory we can identify the settings that might cause oscillations and thus pose restrictions on such settings, the risk of oscillation will be eliminated. Cunha et al. [18] and Netto et al. [72] adopted this approach and presented models that identify the potential oscillation conditions in their rule-based auto-scalers.

10 SCALING TIMING

When to scale the application is a critical question that needs to be answered by auto-scalers. However, there is no perfect solution for this issue as different applications have diverse workload characteristics and preferences of cost and QoS. Auto-scalers can be classified into two groups based on this criterion: approaches that reactively scale the application only when necessary according to the current status of the application and the workload, and approaches that support proactive provisioning or deprovisioning of resources considering the future needs of the application.

For applications with gradual and smooth workload changes, reactive approaches are usually preferred because they can save more resources without causing a significant amount of SLA violations. In contrast, applications with drastic workload changes or strict SLA requirements often require proactive scaling before the workload increases to avoid incurring a significant amount of SLA violations during the provisioning time. Such strategy relies on prediction techniques to in a timely manner foresee incoming workload changes. Prediction is the process of learning relevant knowledge from the history and then applying it to forecast the future behaviors of some object. The assumption that behaviors are predictable lies in that they are not completely random and follow some rules. Therefore, workload prediction is only viable for the workloads with patterns and thus cannot handle the random bursts of requests, which are common in some applications, like news feeds and social networks. For these bursty workload scenarios, currently there is no effective solution, and we can only deal with them reactively in the best effort. Hence, regardless of the existence of support for proactive scaling, a qualified auto-scaler should always be able to scale reactively.

10.1 Proactive Scaling

As the accuracy of the prediction algorithm determines the capability of the auto-scaler to scale applications proactively, in this section, we survey prediction algorithms that have been employed by state-of-the-art works.

10.1.1 Workload Prediction Data Source. It is necessary to study the past workload history to understand the characteristics of the workload, including its intensity and mix during each time frame, to predict it.

Besides workload history, individual applications can rely on available information from other aspects to predict request bursts that are impossible to be derived from past workload data alone, such as weather information for an outdoor application, and political events for a news feed application. However, the relevant parameters are application specific and thus this feature is hard to integrate into a general-purpose auto-scaler. Besides, it is also challenging to devise a prediction algorithm with real-time accuracy for resource provisioning, because there are too many parameters in the model and errors can quickly accumulate. The work by Frey et al. [28] considers multiple outside parameters in an auto-scaler. Their approach integrates all the prediction information into a fuzzy controller.

Though it is challenging to predict the exact workload intensity using outside information, it is viable to in a timely manner detect events that may affect incoming workload intensity through social media and other channels [108]. Since this is a broad topic itself, we focus on prediction algorithms only based on workload history.

10.1.2 Prediction Horizon and Control. Typically, a prediction algorithm loops in a specified interval to predict the average or maximum workloads arriving at the application during each of the next few intervals, which form the prediction horizon. It determines how far in the future the auto-scaler aims to predict.

There are two approaches in which auto-scalers can apply prediction results in resource provisioning. The first way, which is adopted by the majority of works, takes the prediction horizon as the control interval and scales the application only based on the predicted workload of the next horizon. The other strategy is called Model Predictive Control (MPC). It sets the control interval the same as the prediction interval. When making decisions, it considers all the intervals within the horizon and determines the scaling operations at each interval using optimization. However, when executing the scaling operations, it only performs the action for the next interval and discards operations for the other intervals in the horizon. This method mitigates the problem of provisioning for short-term benefits, but it requires solving complex optimization models and, thus, consumes much more computing power. Ghanbari et al. [35, 36] and Zhang et al. [110] employed this approach.

To tune the length of the horizon, users can adjust the duration of either each interval or number of intervals in the horizon. The size of the interval is critical to prediction precision. A large interval can significantly degrade the prediction accuracy and is useless for real-time control if the interval is greater than the control interval of the auto-scaler. The number of intervals in the horizon is also a crucial parameter, especially for the MPC approach. A balanced number should be chosen for the auto-scaler to reach good performance. If it is too small, MPC cannot fully realize its potential to make decisions for the long-term benefit. A large number, on the other hand, may mislead the auto-scaler as predictions for the intervals far in the future become increasingly inaccurate.

10.1.3 Workload Prediction Algorithms. Regarding workload prediction algorithms, they can be coarsely classified into two types: prediction according to recent trends and prediction based on regular patterns.

Prediction according to recent trends aims to use the workload data monitored in the near past to determine whether the workload is increasing or decreasing and how much it will change. In this case, only a few data are stored for prediction purposes. Well-established time-series analysis algorithms are commonly applied to this type of prediction task, such as linear regression [10, 37], various autoregressive models (ARs) [14, 26, 85, 104, 107], and neural-network-based approaches [7, 74, 79]. Besides using time-series analysis, Nguyen et al. [73] proposed another method, which considers each time interval as a wavelet-based signal and then applies signal prediction techniques.

Prediction algorithms based on regular patterns assume the workload is periodic, which is valid for many applications as they tend to be accessed more during the daytime, weekdays, or particular days in a year (tax report period, Christmas holidays). By finding these patterns, predictions can be easily made. Different from prediction algorithms based on recent trends, this type of algorithm requires a large workload archive across an extended period. Various approaches have been explored to identify workload patterns when building auto-scalers. Fang et al. [26] employed signal processing techniques to discover the lowest dominating frequency (which corresponds to the longest repeating pattern). Dias et al. [19] utilized the Holt-Winter model, which aims to identify seasonality in the workload for prediction. Jiang et al. [57] devised an approach by first identifying the top K most relevant monitored data using an auto-correlation function and then employing linear regression on the selected data for prediction. Urgaonkar et al. [99] adopted an algorithm based on the histogram for the workload with daily patterns.

Herbst et al. [48] integrated many predictors into one auto-scaler. They presented an approach to dynamically select appropriate prediction methods according to the extracted workload intensity behavior (WIB, simply the workload characteristics) and user's objectives. The mappings of prediction methods to WIBs are stored in a decision tree and are updated during runtime based on the recent accuracy of each algorithm.

10.1.4 Resource Usage Prediction. Instead of predicting workload, it is also possible to directly predict resulting resource usage according to the historical usage data. This strategy is commonly used by auto-scalers that only support vertical scaling, as for a single machine, resource usage can substitute workload intensity. Some proposals [5, 13, 53] that target horizontal scaling also follow this strategy to accomplish both workload prediction and resource estimation together.

Gong et al. [39] used signal processing to discover the longest repeating pattern of resource usage and then relied on the dynamic time warping (DTW) algorithm to make the prediction. For applications without repeating patterns, they referred to a discrete-time Markov chain with finite states to derive a near prediction of future values. Islam et al. [53] explored using linear regression and neural network to predict CPU usage. Caron et al. [13] adopted a pattern matching approach that abstracts it as a string matching problem and solved it using the Knuth-Morris-Pratt (KMP) algorithm. Yazdanov et al. [106] utilized an AR method to predict short-term CPU usage. Morais et al. [5] employed multiple time-series algorithms to predict CPU usage, and based on their runtime accuracy, the best is selected. Loff et al. [69] also used various prediction algorithms. However, instead of selecting the best one, their approach combines the results of different predictors using the weighted k-nearest neighbors algorithm. The weight of each predictor is dynamically adjusted according to their recent accuracy.

11 SCALING METHODS

Depending on the particular cloud environment, elastic scaling can be performed vertically, horizontally, or both. Each of them has their advantages and limitations. In this section, we discuss the key factors that need to be considered when making the provisioning plan.

11.1 Vertical Scaling—VM Resizing

Vertical scaling means removing or adding resources, including CPU, memory, I/O, and network, from or to existing VMs. To dynamically perform these operations, modern hypervisors utilize mechanisms such as CPU sharing and memory ballooning to support CPU and memory hot-plug. However, major cloud providers, such as Amazon, Google, and Microsoft, do not support adjusting resources during runtime. In these platforms, it is essential to shut down the instance first to add resources. Some providers like Centurylink[1] allow users to scale CPU cores without downtime vertically. Profitbricks[2] permits adding both CPU and memory to the VMs dynamically.

Vertical scaling is considered not suitable for highly scalable applications due to its limitations. Ideally, the maximum capacity a VM can scale to is the size of the physical host. However, multiple VMs are usually residing on the same physical machine competing for resources, which further confines the potential scaling capability. Though limited, dynamic vertical scaling outperforms horizontal scaling in provisioning time as it can be in effect instantaneously. Besides, some services or components that are difficult to replicate during runtime, such as a database server and stateful application servers, can benefit from vertical scaling. Dawoud et al. [20] conducted an experimental study of vertical scaling using the RUBBOS benchmark on both its application server and database, which highlights the advantages of vertical scaling mentioned above.

Many auto-scalers have been developed using solely vertical scaling to manage VMs on the same physical host. Some of them only considered scaling CPU resources [60, 91, 93, 106]. Some targeted both CPU and memory [15, 20, 39, 102, 107, 111]. Jing et al. [58] focused on CPU in the prototype and claimed their method could be extended to other resources. Bu et al. [11] proposed an approach that adjusts not only CPU and memory allocation but also application parameters. Padala et al. [76] scaled both CPU and disk. These auto-scalers are experimental prototypes designed to be deployed in either private or public clouds. However, they have not been included in any commercial offerings yet.

11.2 Horizontal Scaling—Launching New VMs

Horizontal scaling is the core of the elasticity feature of the cloud. Most cloud providers offer standardized VMs of various sizes for customers to choose from. Others allow users to customize their VMs with a specific amount of cores, memory, and network bandwidth. Besides, multiple pricing models coexist in the current cloud market, which further increases the complexity of the provisioning problem.

11.2.1 Heterogeneity. Regarding a single tier/service within a web application, if the billing is constant, the use of homogeneous VMs is well acceptable as it is easy to manage. The auto-scaling services offered by cloud providers only allow the use of homogeneous VMs. Selecting which type of VM to use is considered the responsibility of users in commercial auto-scalers. The optimal solution depends on the resource profile of the tier/service, e.g., whether it is CPU or memory intensive, and the workload characteristic. Under large workloads, using a small or large instance results in little difference as the remainder of resources that are not used account for a very small percentage of the total resources. Under a small and fluctuant workload smaller instances are preferred, as scaling can be conducted in finer granularity and thus saves cost.

Cost efficiency of VMs is highly corelated to the application and workload. If changes happen to them, the choice of VM type should also be reconfigured. Grozev et al. [44] proposed a method that

[1]https://www.ctl.io/autoscale/.
[2]https://www.profitbricks.com/help/Live_Vertical_Scaling.

detects changes online using the Hierarchical Temporal Memory (HTM) model and a dynamically trained artificial neural network (ANN) and then reselects the most cost-efficient VM type.

The use of heterogeneous VMs to scale web applications has been explored in the literature. Under conventional billing, where price grows linearly with VM's capability, heterogeneity can bring some extra cost savings, but these are not significant. Furthermore, the search of the optimal provisioning plan with a combination of heterogeneous VMs is often computing intensive. Srirama et al. [94] employed linear programming to solve the provisioning problem, yet only achieved limited cost savings against AWS auto-scaling. Fernandez et al. [27] abstracted the provisioning combinations as a tree and searched the proper provisioning by traversing the tree according to different SLAs. Sharma et al. [90] used a greedy algorithm to obtain the provisioning plan. In a different scenario in which the capability of VMs increases exponentially to their prices, heterogeneity has the potential to save significant cost, which is shown in Sedaghat et al. [88] and Upendra et al. [98]. They both consider the transition cost (the time and money spent to convert from the current provisioning to the target provisioning) and the cost of resource combination in the optimization problem.

We proposed an auto-scaler [80] that uses heterogeneous spot instances to provision web applications. The intention of using heterogeneous VMs in this case is to boost the reliability of clusters based on spot instances to save cost, which is explained in the following section.

11.2.2 Pricing Models. The current cloud pricing models can be classified into three types by pricing model: on-demand, reserved, and rebated. In on-demand mode, the provider sets a fixed unit price for each type of VM or unit of a certain resource and charges the user by units of consumption. Users are guaranteed to obtain the required resources and agreed performance, which most auto-scalers assume the target application is adopting. The reserved mode requires the user to pay an upfront fee for cheaper use of a certain amount of resources. If highly utilized, users can save a considerable sum of money than acquiring resources in on-demand mode. Providers create the rebate mode aiming to sell their spare capacity. They are usually significantly cheaper than on-demand resources. There are several ways to offer rebated resources. Amazon employed an auction-like mechanism to sell instances, called spot instances. In this mode, the user is required to submit a bid on the resources. Suppose the bid exceeds the current market price, the bid is fulfilled, and the user is only charged for the current market price. The acquired spot instances are guaranteed to have the same performance of their on-demand counterparts. However, they are reclaimed whenever the market price goes beyond a user's bidding price. Google offers its spare capacity as preemptible VMs. Different from Amazon, it sets a fixed price to the VM, which is 30% of the regular price, and the VM is available at most for 24 hours. Rebated instances are considered not suitable to host web applications that are availability critical as they can be reclaimed by providers at any time.

ClusterK[3] and our previous work [80] demonstrated that it is feasible to build an auto-scaler utilizing spot instances by exploiting various market behaviors of different spot markets to achieve both high availability and considerable cost savings. Sharma et al. [89] and He et al. [47] endeavored to build a reliable general-purpose cloud platform upon spot instances using nested virtualization. As these systems lie in the underlying virtualization layer, all types of auto-scaling techniques can be applied to them.

Pricing models can also be classified according to their billing period, which is the minimum unit consumption. Providers have set their billing period to every minute, hour, day, week, month,

[3]http://www.geekwire.com/2015/amazon-buys-clusterk-a-startup-that-lets-developers-run-aws-workloads-more-cheaply/ acquired by AWS in 2015.

or year. The length of the billing period has a significant impact on the cost efficiency for elasticity. Obviously, the shorter the billing period, the more flexible and cost efficient it is for auto-scaling. If the billing period exceeds the order of hour, there is no use in applying auto-scaling to save resource costs, since provisioning for the peak load of the day incurs the same cost.

11.3 Hybrid

As mentioned previously, horizontal scaling is slow in provisioning and vertical scaling is restricted by resources available in the host. It is natural to employ vertical scaling and horizontal scaling together to mitigate these issues. The idea is to utilize vertical scaling when possible to quickly adapt to changes and only conduct horizontal scaling when vertical scaling reaches its limit. Urgaonkar et al. [99], Huber et al. [49], Rui et al. [86], and Yang et al. [104] followed this strategy.

Mixing vertical scaling and horizontal scaling can also bring cost benefits. Dutta et al. [24] and Gandhi et al. [32] explored optimization techniques to search for the scaling plan that incurs the least cost with a hybrid of vertical and horizontal scaling.

Vertical scaling and horizontal scaling can be separately applied to different components of the application as well since some parts such as database servers are difficult to horizontally scale. Nisar et al. [75] demonstrated this approach in a case study.

12 ENVIRONMENT

In the previous sections, we discussed the characteristics of auto-scalers operating in a single-cloud data center. In this section, we first give a summary of single-cloud auto-scalers. After that, we introduce their counterparts that can coordinately work across multiple-cloud data centers.

12.1 Single Cloud

The auto-scaling challenges and developments in single-cloud environments have been thoroughly covered in the previous sections. The auto-scaling process is abstracted as an MAPE loop, and within each phase of the loop, we have identified corresponding design challenges. Based on the taxonomy and explanation of the concepts, we map each work to the specific categories for each discussed feature in Table 1. Readers can refer to it to quickly grasp the general design of each surveyed auto-scaler.

12.2 Multiple Clouds

Modern applications are often deployed in multiple-cloud data centers for various purposes [42]: (1) multicloud deployment helps reducing response latency if users can be served by the nearest data center, (2) it improves availability and reliability of the application against data center outages by replicating the application stack in multiple regions, (3) it enables the service provider to exploit cost differences among different vendors, and (4) it prevents vendor lock-in. Auto-scalers should be able to support this type of deployment as well.

When expanded to multiple clouds, auto-scaling remains the same problem if applications in different cloud data centers are managed completely standalone, which is the common practice of the industry. In this case, usually the service provider first selects a set of cloud data centers to host the application. Each data center is intended only to serve requests coming from nearby users and is separately managed by a dedicated local auto-scaler without global coordination of request routing and resource provisioning.

Though easy to manage, such strategy is not optimal in an environment where both workload and resource price are highly dynamic. As time passes, it is better to move resources to cheaper data centers to save cost, or to data centers that are closer to certain groups of users to improve

Table 1. A Review of Auto-Scaling Properties of Key Works for Single Cloud

Work	Application Architecture	Sticky Session	Adaptivity	Scaling Indicators	Resource Estimation	Oscillation Mitigation	Proactive	Scaling Methods
Doyle et al. [21]	single-tier	✓	nonadaptive	hybrid	analytical model	–	×	vertical
Kamra et al. [61]	3-tier	✓	self-adaptive	high-level	analytical model	–	×	vertical
Tesauro et al. [95]	single-tier	×	self-adaptive	high-level	reinforcement learning	–	×	hom. horizontal
Tesauro et al. [96]	single-tier	×	self-adaptive	high-level	hybrid	–	×	hom. horizontal
Jing et al. [58]	single-tier	✓	self-adaptive	high-level	hybrid	–	×	vertical
Villela et al. [101]	single-tier	×	nonadaptive	hybrid	analytical model	–	×	hom. horizontal
Zhang et al. [109]	multitier	–	–	hybrid	hybrid	–	–	–
Chen et al. [14]	single-tier	✓	self-adaptive	hybrid	regression	–	✓	hom. horizontal
Urgaonkar et al. [99]	multitier	×	nonadaptive	high-level	analytical model	–	✓	hybrid
Iqbal et al. [50]	single-tier	×	nonadaptive	high-level	rule-based	–	×	hetr. horizontal
Lim et al. [68]	single-tier	×	self-adaptive	low-level	rule-based	dynamic para.	×	hom. horizontal
Bodik et al. [10]	single-tier	×	self-adaptive	high-level	regression	dynamic para.	✓	hom. horizontal
Padala et al. [76]	multitier	✓	self-adaptive	hybrid	regression	dynamic para.	×	vertical
Kalyvianaki et al. [60]	single-tier	✓	self-adaptive	low-level	rule-based	–	×	vertical
Lama et al. [64]	multitier	×	self-adaptive	high level	fuzzy inference	dynamic para.	×	hom. horizontal
Lim et al. [67]	storage-tier	×	self-adaptive	low-level	rule-based	dynamic para.	×	hom. horizontal
Dutreilh et al. [23]	single-tier	×	self-adaptive	high-level	hybrid	cooling time	×	hom. horizontal
Gong et al. [39]	single-tier	✓	self-adaptive	low-level	hybrid	–	✓	vertical
Islam et al. [53]	single-tier	×	self-adaptive	low-level	neural net./regression	–	✓	–
Lama et al. [65]	multitier	×	self-adaptive	high-level	hybrid	–	×	hom. horizontal
Bi et al. [9]	multitier	×	nonadaptive	high-level	analytical model	–	×	hom. horizontal
Singh et al. [92]	multitier	×	nonadaptive	high-level	analytical model	–	×	hom. horizontal
Jiang et al. [55]	SOA	×	self-adaptive	high-level	analytical model	–	×	hom. horizontal
Chieu et al. [17]	single-tier	✓	nonadaptive	high-level	rule-based	–	×	hom. horizontal
Dutreilh et al. [22]	single-tier	×	self-adaptive	high-level	reinforcement learning	–	×	hom. horizontal
Li et al. [66]	single-tier	×	self-adaptive	low-level	reinforcement learning	–	×	hom. horizontal
Caron et al. [13]	single-tier	×	self-adaptive	low-level	string matching	–	✓	–
Huber et al. [49]	single-tier	×	nonadaptive	hybrid	rule-based	–	×	hybrid

(Continued)

Table 1. Continued

Work	Application Architecture	Sticky Session	Adaptivity	Scaling Indicators	Resource Estimation	Oscillation Mitigation	Proactive	Scaling Methods
Iqbal et al. [52]	multitier	×	self-adaptive	hybrid	hybrid	—	×	hom. horizontal
Jiang et al. [56]	multitier	×	nonadaptive	hybrid	online profiling	—	×	hetr. horizontal
Malkowski et al. [71]	multitier	×	self-adaptive	hybrid	hybrid	—	×	hom. horizontal
Roy et al. [85]	multitier	×	nonadaptive	hybrid	analytical model	—	✓	hom. horizontal
Upendra et al. [98]	multitier	×	nonadaptive	high-level	profiling	—	✓	hetr. horizontal
Vasic et al. [100]	single-tier	×	self-adaptive	low-level	online profiling	—	×	hom. horizontal
Ali-Eldin et al. [3]	single-tier	×	self-adaptive	high-level	analytical model	dynamic para.	✓	hom. horizontal
Ali-Eldin et al. [2]	single-tier	×	self-adaptive	high-level	analytical model	dynamic para.	✓	hom. horizontal
Dawoud et al. [20]	single-tier	×	nonadaptive	low-level	rule-based	—	×	compare ver. hor.
Fang et al. [26]	single-tier	×	—	—	—	—	✓	—
Yazdanov et al. [106]	single-tier	✓	self-adaptive	low-level	regression	—	✓	vertical
Ghanbari et al. [36]	single-tier	×	nonadaptive	high-level	analytical model	—	×	hetr. horizontal
Zhu et al. [111]	single-tier	✓	self-adaptive	low-level	reinforcement learning	—	×	vertical
Dutta et al. [24]	multitier	×	nonadaptive	hybrid	application profiling	—	×	hybrid
Gandhi et al. [33]	multitier	×	nonadaptive	hybrid	profiling	—	×	hom. horizontal
Rui et al. [86]	multitier	×	nonadaptive	high-level	rule-based	—	×	hybrid
Sharma et al. [90]	multitier	×	nonadaptive	high-level	analytical model	—	×	hetr. horizontal
Jiang et al. [57]	single-tier	×	nonadaptive	high-level	analytical model	—	✓	hom. horizontal
Al-Haidari et al. [1]	single-tier	×	nonadaptive	high-level	rule-based	—	×	hom. horizontal
Bu et al. [11]	single-tier	✓	self-adaptive	high-level	reinforcement learning	—	×	vertical
Gambi et al. [30]	single-tier	×	self-adaptive	low-level	Kriging regression	—	×	hom. horizontal
Barrett et al. [8]	single-tier	×	self-adaptive	high-level	reinforcement learning	—	×	hetr. horizontal
Sedaghat et al. [88]	single-tier	×	nonadaptive	high-level	—	—	×	hetr. horizontal
Yazdanov et al. [107]	single-tier	×	self-adaptive	hybrid	reinforcement learning	—	✓	vertical
Ali-Eldin et al. [4]	single-tier	×	switch	—	—	—	✓	hom. horizontal
Morais et al. [5]	single-tier	×	self-adaptive	low-level	various regressions	—	✓	hom. horizontal
Nguyen et al. [73]	multitier	×	nonadaptive	hybrid	online profiling	—	✓	hom. horizontal
Herbst et al. [48]	—	×	self-adaptive	—	—	—	✓	—
Grozev et al. [43]	single-tier	✓	nonadaptive	low-level	rule-based	—	×	hom. horizontal

(Continued)

Table 1. Continued

Work	Application Architecture	Sticky Session	Adaptivity	Scaling Indicators	Resource Estimation	Oscillation Mitigation	Proactive	Scaling Methods
Dias et al. [19]	single-tier	×	nonadaptive	hybrid	rule-based	–	✓	hom. horizontal
Loff et al. [69]	single-tier	×	nonadaptive	low-level	rule-based	–	✓	hom. horizontal
Cunha et al. [18]	single-tier	×	self-adaptive	low-level	rule-based	theory	×	hom. horizontal
Netto et al. [72]	single-tier	×	self-adaptive	low-level	rule-based	theory	×	hom. horizontal
Aniello et al. [7]	single-tier	×	nonadaptive	high-level	analytical model	–	✓	hom. horizontal
Frey et al. [28]	single-tier	×	nonadaptive	hybrid	fuzzy inference	–	✓	hom. horizontal
Yang et al. [104]	single-tier	×	nonadaptive	hybrid	rule-based	–	✓	hetr. horizontal
Fernandez et al. [27]	single-tier	×	nonadaptive	high-level	profiling	–	×	hetr. horizontal
Srirama et al. [94]	single-tier	×	nonadaptive	–	–	–	×	hetr. horizontal
Gandhi et al. [32]	single-tier	×	self-adaptive	high-level	analytical model	–	×	hybrid
Spinner et al. [93]	single-tier	✓	self-adaptive	hybrid	analytical model	–	×	vertical
Gergin et al. [34]	multitier	×	nonadaptive	high-level	analytical model	–	×	hom. horizontal
Han et al. [46]	multitier	×	nonadaptive	high-level	analytical model	–	×	hom. horizontal
Kaur et al. [62]	multitier	×	nonadaptive	high-level	analytical model	–	✓	hom. horizontal
Gandhi et al. [31]	multitier	×	self-adaptive	hybrid	analytical model	–	×	hom. horizontal
Nikravesh et al. [74]	–	–	–	–	–	–	✓	–
Yanggratoke et al. [105]	single-tier	×	self-adaptive	high-level	batch & online learning	–	×	hom. horizontal
Grimaldi et al. [41]	single-tier	×	self-adaptive	low-level	rule-based	–	×	hom. horizontal
Gambi et al. [29]	single-tier	×	self-adaptive	high-level	hybrid	–	×	hom. horizontal
Salah et al. [87]	single-tier	×	nonadaptive	high-level	analytical model	–	×	hom. horizontal
Iqbal et al. [51]	multitier	×	self-adaptive	high-level	reinforcement learning	–	×	hom. horizontal
Chen et al. [15]	single-tier	✓	self-adaptive	hybrid	analytical model	–	×	vertical
Amazon [6]	single-tier	×	nonadaptive	high/low	rule-based	cooling time	×	hom. horizontal
RightScale [82]	single-tier	×	nonadaptive	high/low	rule-based	cooling time	×	hom. horizontal
Qu et al. [80]	single-tier	×	nonadaptive	low-level	profiling	–	×	hetr. horizontal
Jamshidi et al. [54]	single-tier	×	self-adaptive	high-level	hybrid	–	×	hom. horizontal
Grozev et al. [44]	single-tier	×	self-adaptive	hybrid	rule-based	–	×	hetr. horizontal
Chen et al. [16]	single-tier	×	self-adaptive	hybrid	hybrid	–	–	–
Wang et al. [102]	single-tier	✓	self-adaptive	hybrid	analytical modeling	–	✓	vertical
Ghobaei-Arani et al. [37]	single-tier	×	self-adaptive	hybrid	reinforcement learning	–	✓	hom. horizontal

their QoS. Auto-scaling becomes more complicated in these scenarios as it needs to make decisions on not only resource provisioning but also location selection and request routing.

Some works explored holistic solutions for resource management of web applications in multiple clouds. They can be further divided into two types. The first type always deploys the whole application stack in the chosen data centers. The other type allows separate deployment of application components in different data centers. Zhang et al. [110] and Rodolakis et al. [84] targeted the first type of problems. Zhang et al. [110] assumed that each potential data center is capped, which is in contrast to the common illusion that cloud data centers have an "infinite" amount of resources, and applications are deployed in one VM. Their objective is to minimize the total cost of resources used by applications through dynamically acquiring and releasing servers from geographically dispersed data centers under the constraint of demand, capacity, and SLA. They employed the MPC framework and a quadratic optimization model to adjust resource allocation in each data center and request routing from each location. Differently, Rodolakis et al. [84] considered a scenario without data center capacity constraints and dynamic pricing. They dissected the problem into three parts and proposed approximation algorithms for each of them to form an integrated solution. Calcavecchia et al. [12] devised a decentralized auto-scaler for multiple clouds. It can autonomously start VMs at regions suffering from insufficient capacity through a voting mechanism. Guo et al. [45] proposed a geo-aware auto-scaler that predicts not only the amount of workload in the future but also the distribution of workload. By doing this, it is able to start and shut down VMs in data centers that are close to the places of workload changes.

Regarding the second problem type, Tortonesi et al. [97] proposed a genetic-based algorithm to search the deployment of a two-tier application across multiple clouds with minimum resource and SLA violation cost. Rochman et al. [83] modeled the problem as a min-cost flow problem and solved it with the Bipartite Graph Algorithm. Grabarnik et al. [40] added more complexity to the problem by also optimizing the chosen VM types for each tier in a multitier application. They devised a two-phase metaheuristic algorithm with the outer phase responsible for assigning components to data centers also using a genetic-based algorithm, and the inner phase using a random search algorithm to map the components to specific types of VMs. None of these solutions bears reliability in mind, which is necessary for this kind of deployment. If poorly planned, instead of improving the reliability and availability of the application, dispersing replicas into multiple data centers can create multiple points of failure and substantially reduce uptime. It is important for auto-scalers to ensure that every component is properly replicated in multiple data centers all the time.

The cost of data replication is another reason that it is beneficial to provide a holistic solution for auto-scaling in multiple clouds for some applications, such as video streaming applications. For these applications, QoS cannot be met without enough bandwidth between the video storage site and the end customer. The simplest solution to the bandwidth provisioning solution is to replicate all the videos in every data center and serve each customer from the one with sufficient bandwidth available. However, it is unrealistic and extremely wasteful. The service provider needs to decide for each video how many replicas it should keep and where they should be placed to save cost. Along with the data, serving applications should be colocated as well, and user requests need to be properly diverted to particular serving replicas because of the bandwidth limit of serving VMs. To realize the above targets, Wu et al. [103] proposed and implemented a prototype using Model Predictive Control and subgradient algorithm.

The mentioned holistic approaches require solving complex optimization problems, which takes considerable time, making them only applicable to perform auto-scaling in coarse-grained time intervals and limiting their ability to react to drastic workload changes. Therefore, for applications with highly variable workloads, the choice of using holistic approaches is doubtful. In these cases, the local auto-scalers can be deployed in each data center to handle the fine-grained scaling needs.

13 DISCUSSION AND FUTURE DIRECTIONS

According to the taxonomy and analysis, it is clear that there are gaps between the current solutions and an ideal auto-scaler in various aspects. In the following section, we discuss them and point out potential methods and directions to improve current solutions.

13.1 Service-Based Architectures

The research on scaling complex applications following service-based architectures is still at the early stage, and limited literature can be found in this area. Moreover, due to lack of accurate resource estimation models, only a simple approach that tentatively and recursively provisions resources to a selected service is proposed, which takes a long time to reach the overall target performance. If an accurate resource estimation model is available for service-based applications, the auto-scaler can provision resources in one shot to every service with minimum provision time. Models using queuing networks can be explored to fulfill the gap. It also calls for efficient online optimization algorithms to decide how each service should be provisioned in real time to minimize cost.

13.2 Monitoring Tools for Hidden Parameters

It is important to implement low-cost monitoring tools that can provide real-time measurement of unknown parameters, such as average service time and request mix, for general-purpose applications to facilitate accurate resource estimation and provisioning. Because of the intrusive nature of these parameters, such tools can be integrated into application service containers.

13.3 Resource Estimation Models

Although plenty of resource estimation models have been proposed for various types of application architectures, they still need to be improved in accuracy, generality, computing requirements, and ease of use. We believe hybrid estimation models that encompass the strengths of both analytical modeling and machine-learning approaches are the most promising ones. Other directions, such as general-purpose queuing network models for service-based applications, and efficient and accurate online profiling techniques, are important and need to be further investigated.

13.4 Provisioning Using Rebated Pricing Models

Besides Amazon's spot cloud, providers like Google and Microsoft have introduced their rebated pricing models. However, studies have only concentrated on exploring how to utilize Amazon's spot market and have been oblivious to other providers' offerings. New works can aim to use cost models from other providers to provision resources. It is also interesting to research the use of rebated resources in a multiple-cloud environment with resources from multiple data centers of the same provider or from multiple providers to minimize cost under QoS constraints. Besides, the proposed approaches only combine on-demand resources with rebated resources. Auto-scalers that can employ on-demand, reserved, and rebated resources would be useful in industry, which can be another potential future research direction.

13.5 Better Vertical Scaling Support

Only a few providers enable users to vertically scale up their VMs without downtime, and none of them allow live scaling down, mainly due to the induced complication in the resource management of the data centers as it will result in more VM live migrations, and limitations of operating systems. More research needs to be conducted to help providers enable the vertical scaling option in their infrastructure, which involves proposing vertical-scaling-aware VM allocation and live migration algorithms, devising and implementing generic vertical-scaling APIs, and enhancing support for vertical scaling in hypervisors and operating systems.

13.6 Event-Based Workload Prediction

As mentioned before, existing auto-scalers mostly rely on past workload history to predict future workload. With the growing popularity of social media and other real-time information channels, it is interesting to investigate the use of these sources of information to predict workload burst accurately. Although it is difficult to design a general-purpose predictor of this kind for various applications, there is potential to build auto-scalers that cater to the characteristics of a certain type of application that can benefit from this approach, such as news applications whose workloads are boosted by events in the physical world, and outdoor applications whose workloads are subject to weather conditions.

13.7 Reliability-Aware Multicloud Auto-Scaling

Holistic auto-scaling solutions in multicloud environments ignore the impact on application availability caused by data center outages. It is necessary to address this issue before holistic approaches can be applied in a production scenario, which requires new models that quantitatively measure the level of reliability for specific deployments and include a reliability requirement as a constraint in the optimization problem.

13.8 Energy and Carbon-Aware Auto-Scaling

Existing works only focus on financial cost and QoS aspects. As another primary concern of the ICT sector, the energy and carbon footprint should also be considered in the auto-scalers. Nowadays, many data centers are equipped with on-site generators utilizing renewable energy. However, these sources of energy, such as wind and solar, are unstable. At the cloud provider level, the auto-scalers can gather the real-time energy usage information and preferentially provision resources in data centers that have renewable energy available to maximize use of on-site renewable energy. Within a single data center, auto-scalers can utilize vertical scaling as much as possible to avoid starting new physical machines to save energy.

13.9 Infrastructure-Level Auto-Scaling with User Preferences

From a provider's perspective, enabling auto-scaling helps them to cut cost and meet environmental obligations by reducing electricity consumption and carbon emission. However, they often do not have the freedom to allocate resources asked for by users to whichever data center they own, which limits their ability to maximize their savings. For example, the cloud provider prefers to start VMs in its US data center because it generates extra solar energy, while the application provider hopes the resources can be allocated in Europe, which is the target market that the application is serving. It is an interesting problem to study, how cloud providers should automatically allocate resources for their customers to minimize their own bills while respecting each user's individual preferences regarding the time and space locality of their applications, government regulations, and resource types.

13.10 Container-Based Auto-Scalers

The emergence of containers, especially container-supported microservices and service pods, has raised a new revolution in web application resource management. However, dedicated auto-scaling solutions that cater to the specific characteristics of the container era are still left to be explored. Though this survey focuses on auto-scalers based on VMs, we believe some of the notions and techniques mentioned in this article can inspire research of container-based auto-scalers as the core requirements of them are similar. For example, current workload prediction and oscillation mitigation techniques can be directly applied to container-based systems as they are not dependent on the underlying platforms. However, in some aspects they are different; for example, containers are more flexible in size and quicker to provision, which results in a larger optimization space for making scaling plans. Besides, containers are more susceptible to noisy neighbors as they offer a weaker level of isolation than VMs. To make things worse, modern container management systems, such as Mesos, Kubernetes, and Openshift, schedule different applications on a shared cluster. These issues can cause resource contentions and high variability in processing power, which should be considered in the resource estimation process. Last but not least, the container-based auto-scaling problem is also mixed with the resource allocation problem as containers need to be efficiently scheduled and consolidated on physical hosts or VMs to save cost.

14 SUMMARY AND CONCLUSIONS

Auto-scaling is a technique that automatically adjusts resources provisioned to applications according to real-time workloads without human intervention. It helps application providers minimize their resource bills of using cloud resources while meeting the QoS expectations of their customers. However, designing and implementing an auto-scaler faces many challenges. Many research works have targeted this problem and many auto-scalers with diverse characteristics have been proposed.

In this article, we surveyed the developments of auto-scaling techniques for web applications in clouds. Auto-scaling can be abstracted as an MAPE (Monitoring, Analysis, Planning, and Execution) loop. We identified key challenges that need to be addressed in each phase of the loop and presented a taxonomy of auto-scalers regarding their key properties. Our taxonomy comprehensively covers the listed challenges and categorizes the works based on their solutions to each problem. According to the taxonomy, we analyzed existing techniques in detail to discuss their strengths and weaknesses. Based on the analysis, we proposed promising directions that the research community can pursue in the future.

ACKNOWLEDGMENTS

We thank Dr. Adel Nadjaran Toosi, Dr. Amir Vahid Dastjerdi, Dr. Yaser Mansouri, Xunyun Liu, Minxian Xu, and Bowen Zhou for their valuable comments and suggestions in improving the quality of the article.

REFERENCES

[1] F. Al-Haidari, M. Sqalli, and K. Salah. 2013. Impact of CPU utilization thresholds and scaling size on autoscaling cloud resources. In *Proceedings of 2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 2. 256–261.

[2] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth. 2012. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date (ScienceCloud'12)*. ACM, New York, 31–40.

[3] A. Ali-Eldin, J. Tordsson, and E. Elmroth. 2012. An adaptive hybrid elasticity controller for cloud infrastructures. In *Proceedings of 2012 IEEE Network Operations and Management Symposium (NOMS'12)*. IEEE, 204–212.

[4] A. Ali-Eldin, J. Tordsson, E. Elmroth, and M. Kihl. 2013. *Workload Classification for Efficient Auto-Scaling of Cloud Resources*. Technical Report. Retrieved from http://www.cs.umu.se/research/uminf/reports/2013/013/part1.pdf.

[5] 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 *Proceedings of 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13)*. 42–49. DOI: http://dx.doi.org/10.1109/CCGrid.2013.74

[6] Amazon. 2016. Amazon Auto Scaling Service. Retrieved from http://aws.amazon.com/autoscaling/.

[7] L. Aniello, S. Bonomi, F. Lombardi, A. Zelli, and R. Baldoni. 2014. *An Architecture for Automatic Scaling of Replicated Services*. Springer International Publishing, 122–137.

[8] E. Barrett, E. Howley, and J. Duggan. 2013. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience* 25, 12 (2013), 1656–1674.

[9] J. Bi, Z. Zhu, R. Tian, and Q. Wang. 2010. Dynamic provisioning modeling for virtualized multitier applications in cloud data center. In *Proceedings of 2010 IEEE 3rd International Conference on Cloud Computing*. 370–377. DOI: http://dx.doi.org/10.1109/CLOUD.2010.53

[10] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. 2009. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*. 12–16.

[11] X. Bu, J. Rao, and C. Z. Xu. 2013. Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. *IEEE Transactions on Parallel and Distributed Systems* 24, 4 (April 2013), 681–690. DOI: http://dx.doi.org/10.1109/TPDS.2012.174

[12] N. M. Calcavecchia, B. A. Caprarescu, E. Di Nitto, D. J. Dubois, and D. Petcu. 2012. DEPAS: A decentralized probabilistic algorithm for auto-scaling. *Computing* 94, 8 (2012), 701–730.

[13] E. Caron, F. Desprez, and A. Muresan. 2011. Pattern matching based forecast of non-periodic repetitive behavior for cloud clients. *Journal of Grid Computing* 9, 1 (2011), 49–64.

[14] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. 2008. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proceedings of 5th USENIX Symposium on Networked Systems Design and Implementation*, Vol. 8. 337–350.

[15] T. Chen and R. Bahsoon. 2015. Self-adaptive trade-off decision making for autoscaling cloud-based services. *IEEE Transactions on Services Computing* 10, 4 (2015), 618–632. DOI: http://dx.doi.org/10.1109/TSC.2015.2499770

[16] T. Chen and R. Bahsoon. 2016. Self-adaptive and online QoS modeling for cloud-based software services. *IEEE Transactions on Software Engineering* 43, 5 (2016), 453–475. DOI: http://dx.doi.org/10.1109/TSE.2016.2608826

[17] T. C. Chieu, A. Mohindra, and A. A. Karve. 2011. Scalability and performance of web applications in a compute cloud. In *Proceedings of 2011 IEEE 8th International Conference on e-Business Engineering (ICEBE'11)*. 317–323. DOI: http://dx.doi.org/10.1109/ICEBE.2011.63

[18] R. L. F. Cunha, M. D. Assuncao, C. Cardonha, and M. A. S. Netto. 2014. Exploiting user patience for scaling resource capacity in cloud services. In *Proceedings of 2014 IEEE 7th International Conference on Cloud Computing*. 448–455.

[19] A. da Silva Dias, L. H. V. Nakamura, J. C. Estrella, R. H. C. Santana, and M. J. Santana. 2014. Providing IaaS resources automatically through prediction and monitoring approaches. In *Proceedings of 2014 IEEE Symposium on Computers and Communication (ISCC'14)*. 1–7. DOI: http://dx.doi.org/10.1109/ISCC.2014.6912590

[20] W. Dawoud, I. Takouna, and C. Meinel. 2012. *Elastic Virtual Machine for Fine-Grained Cloud Resource Provisioning*. Communications in Computer and Information Science, Vol. 269. Springer, Berlin, 11–25.

[21] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. Vahdat. 2003. Model-based resource provisioning in a web service utility. In *Proceedings of the 2003 USENIX Symposium on Internet Technologies and Systems*, Vol. 4. 5–5.

[22] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck. 2011. Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow. In *Proceedings of 2011 International Conference on Autonomic and Autonomous Systems*. 67–74.

[23] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck. 2010. From data center resource allocation to control theory and back. In *Proceedings of 2010 IEEE 3rd International Conference on Cloud Computing*. 410–417.

[24] S. Dutta, S. Gera, V. Akshat, and B. Viswanathan. 2012. SmartScale: Automatic application scaling in enterprise clouds. In *Proceedings of 2012 IEEE 5th International Conference on Cloud Computing (CLOUD'12)*. 221–228.

[25] M. Fallah, M. G. Arani, and M. Maeen. 2015. NASLA: Novel auto scaling approach based on learning automata for web application in cloud computing environment. *International Journal of Computer Applications* 113, 2 (2015).

[26] W. Fang, Z. Lu, J. Wu, and Z. Cao. 2012. RPPS: A novel resource prediction and provisioning scheme in cloud data center. In *Proceedings of 2012 IEEE Ninth International Conference on Services Computing (SCC'12)*. 609–616. DOI: http://dx.doi.org/10.1109/SCC.2012.47

[27] H. Fernandez, G. Pierre, and T. Kielmann. 2014. Autoscaling web applications in heterogeneous cloud infrastructures. In *Proceedings of 2014 IEEE International Conference on Cloud Engineering (IC2E'14)*. 195–204.

[28] S. Frey, C. Luthje, C. Reich, and N. Clarke. 2014. Cloud QoS scaling by fuzzy logic. In *Proceedings of 2014 IEEE International Conference on Cloud Engineering (IC2E'14)*. 343–348.

[29] A. Gambi, M. Pezze, and G. Toffetti. 2016. Kriging-based self-adaptive cloud controllers. *IEEE Transactions on Services Computing* 9, 3 (2016), 368–381. DOI: http://dx.doi.org/10.1109/TSC.2015.2389236

[30] A. Gambi, G. Toffetti, C. Pautasso, and M. Pezzé. 2013. Kriging controllers for cloud applications. *IEEE Internet Computing* 17, 4 (July 2013), 40–47. DOI: http://dx.doi.org/10.1109/MIC.2012.142

[31] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang. 2014. Adaptive, model-driven autoscaling for cloud applications. In *Proceedings of the 11th International Conference on Autonomic Computing*.

[32] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang. 2014. Modeling the impact of workload on cloud resource scaling. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'14)*. 310–317.

[33] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. 2012. AutoScale: Dynamic, robust capacity management for multitier data centers. *ACM Transactions on Computer Systems* 30, 4 (Nov. 2012), 14:1–14:26.

[34] I. Gergin, B. Simmons, and M. Litoiu. 2014. A decentralized autonomic architecture for performance control in the cloud. In *Proceedings of 2014 IEEE International Conference on Cloud Engineering (IC2E'14)*. 574–579.

[35] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna. 2014. Replica placement in cloud through simple stochastic model predictive control. In *2014 IEEE 7th International Conference on Cloud Computing*. 80–87. DOI: http://dx.doi.org/10.1109/CLOUD.2014.21

[36] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai. 2012. Optimal autoscaling in a IaaS cloud. In *Proceedings of the 9th International Conference on Autonomic Computing (ICAC'12)*. ACM, New York, 173–178. DOI: http://dx.doi.org/10.1145/2371536.2371567

[37] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina. 2018. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems* 78, Part 1 (2018), 191–210. DOI: http://dx.doi.org/10.1016/j.future.2017.02.022

[38] B. V. Gnedenko and I. N. Kovalenko. 1989. *Introduction to Queueing Theory*. Birkhauser Boston.

[39] Z. Gong, X. Gu, and J. Wilkes. 2010. PRESS: PRedictive elastic resource scaling for cloud systems. In *Proceedings of 2010 International Conference on Network and Service Management*. 9–16.

[40] G. Y. Grabarnik, L. Shwartz, and M. Tortonesi. 2014. Business-driven optimization of component placement for complex services in federated clouds. In *Proceedings of 2014 IEEE Network Operations and Management Symposium (NOMS'14)*. 1–9.

[41] D. Grimaldi, V. Persico, A. Pescape, A. Salvi, and S. Santini. 2015. A feedback-control approach for resource management in public clouds. In *2015 IEEE Global Communications Conference (GLOBECOM'15)*. 1–7. DOI: http://dx.doi.org/10.1109/GLOCOM.2015.7417016

[42] N. Grozev and R. Buyya. 2014. Inter-cloud architectures and application brokering: Taxonomy and survey. *Software: Practice and Experience* 44, 3 (2014), 369–390. DOI: http://dx.doi.org/10.1002/spe.2168

[43] N. Grozev and R. Buyya. 2014. Multi-cloud provisioning and load distribution for three-tier applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 9, 3 (2014), 13.

[44] N. Grozev and R. Buyya. 2016. Dynamic selection of virtual machines for application servers in cloud environments. *CoRR* abs/1602.02339 (2016). Retrieved from http://arxiv.org/abs/1602.02339.

[45] T. Guo, P. Shenoy, and H. H. Hacigumus. 2016. GeoScale: Providing geo-elasticity in distributed clouds. In *Proceedings of 2016 IEEE International Conference on Cloud Engineering (IC2E'16)*. 123–126. DOI: http://dx.doi.org/10.1109/IC2E.2016.40

[46] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond. 2014. Enabling cost-aware and adaptive elasticity of multitier cloud applications. *Future Generation Computer Systems* 32 (2014), 82–98.

[47] X. He, P. Shenoy, R. Sitaraman, and D. Irwin. 2015. Cutting the cost of hosting online services using cloud spot markets. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 207–218.

[48] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn. 2014. Self-adaptive workload classification and forecasting for proactive resource provisioning. *Concurrency and Computation: Practice and Experience* 26, 12 (2014), 2053–2078. DOI: http://dx.doi.org/10.1002/cpe.3224

[49] N. Huber, F. Brosig, and S. Kounev. 2011. Model-based self-adaptive resource allocation in virtualized environments. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'11)*. ACM, New York, 90–99. DOI: http://dx.doi.org/10.1145/1988008.1988021

[50] W. Iqbal, M. Dailey, and D. Carrera. 2009. *SLA-Driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud*. Springer, 243–253.

[51] W. Iqbal, M. N. Dailey, and D. Carrera. 2016. Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multitier web applications. *IEEE Systems Journal* 10, 4 (2016), 1453–1446. DOI: http://dx.doi.org/10.1109/JSYST.2015.2424998

[52] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek. 2011. Adaptive resource provisioning for read intensive multitier applications in the cloud. *Future Generation Computer Systems* 27, 6 (2011), 871–879.

[53] S. Islam, J. Keung, K. Lee, and A. Liu. 2010. An empirical study into adaptive resource provisioning in the cloud. In *Proceedings of IEEE International Conference on Utility and Cloud Computing (UCC'10)*. 8.

[54] P. Jamshidi, C. Pahl, and N. C. Mendonça. 2016. Managing uncertainty in autonomic cloud elasticity controllers. *IEEE Cloud Computing* 3, 3 (May 2016), 50–60.

[55] D. Jiang, G. Pierre, and C.-H. Chi. 2010. Autonomous resource provisioning for multi-service web applications. In *Proceedings of the 19th International Conference on World Wide Web*. ACM, 471–480.

[56] D. Jiang, G. Pierre, and C.-H. Chi. 2011. Resource provisioning of web applications in heterogeneous clouds. In *Proceedings of the 2nd USENIX Conference on Web Application Development*. USENIX Association, 49–60.

[57] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. 2013. Optimal cloud resource auto-scaling for web applications. In *Proceedings of 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'13)*. IEEE, 58–65.

[58] X. Jing, Z. Ming, J. Fortes, R. Carpenter, and M. Yousif. 2007. On the use of fuzzy modeling in virtualized data center management. In *Proceedings of 4th International Conference on Autonomic Computing (ICAC'07)*. 25–25.

[59] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. 2008. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proceedings of 2008 International Conference on Autonomic Computing (ICAC'08)*. 23–32.

[60] E. Kalyvianaki, T. Charalambous, and S. Hand. 2009. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the 6th International Conference on Autonomic Computing (ICAC'09)*. ACM, New York, 117–126. DOI: http://dx.doi.org/10.1145/1555228.1555261

[61] A. Kamra, V. Misra, and E. M. Nahum. 2004. Yaksha: A self-tuning controller for managing the performance of 3-tiered Web sites. In *12th IEEE International Workshop on Quality of Service, 2004 (IWQOS'04)*. 47–56. DOI: http://dx.doi.org/10.1109/IWQOS.2004.1309356

[62] P. D. Kaur and I. Chana. 2014. A resource elasticity framework for QoS-aware execution of cloud applications. *Future Generation Computer Systems* 37 (2014), 14–25.

[63] J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (Jan. 2003), 41–50.

[64] P. Lama and X. Zhou. 2009. Efficient server provisioning with end-to-end delay guarantee on multitier clusters. In *Proceedings of 17th International Workshop on Quality of Service (IWQoS'09)*. 1–9.

[65] P. Lama and X. Zhou. 2010. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 151–160.

[66] H. Li and S. Venugopal. 2011. Using reinforcement learning for controlling an elastic web application hosting platform. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*. ACM, New York, 205–208. DOI: http://dx.doi.org/10.1145/1998582.1998630

[67] H. C. Lim, S. Babu, and J. S. Chase. 2010. Automated control for elastic storage. In *Proceedings of the 7th International Conference on Autonomic Computing (ICAC'10)*. ACM, New York, 1–10.

[68] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh. 2009. Automated control in cloud computing: Challenges and opportunities. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC'09)*. ACM, New York, 13–18.

[69] J. Loff and J. Garcia. 2014. Vadara: Predictive elasticity for cloud applications. In *Proceedings of 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom'14)*. 541–546. DOI: http://dx.doi.org/10.1109/CloudCom.2014.161

[70] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing* 12, 4 (2014), 559–592.

[71] S. J. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann. 2011. Automated control for elastic N-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*. ACM, New York, NY, USA, 131–140.

[72] M. A. S. Netto, C. Cardonha, R. L. F. Cunha, and M. D. Assuncao. 2014. Evaluating auto-scaling strategies for cloud computing environments. In *Proceedings of 2014 IEEE 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems*. 187–196.

[73] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. 2013. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the USENIX International Conference on Automated Computing (ICAC'13)*.

[74] A. Y. Nikravesh, S. A. Ajila, and C. H. Lung. 2015. Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In *Proceedings of the 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 35–45. DOI: http://dx.doi.org/10.1109/SEAMS.2015.22

[75] A. Nisar, W. Iqbal, F. S. Bokhari, and F. Bukhari. [n.d.]. Hybrid auto-scaling of multi-tier web applications: A case of using Amazon public cloud.

[76] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. 2009. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys'09)*. ACM, New York, 13–26.

[77] A. V. Papadopoulos, Ahmed Ali-Eldin, Karl-Erik Årzén, Johan Tordsson, and Erik Elmroth. 2016. PEAS: A performance evaluation framework for auto-scaling strategies in cloud applications. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 1, 4, Article 15 (Aug. 2016), 31 pages.

[78] T. Patikirikorala, A. Colman, J. Han, and L. Wang. 2011. A multi-model framework to implement self-managing control systems for QoS management. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 218–227.

[79] R. Prodan and V. Nae. 2009. Prediction-based real-time resource provisioning for massively multiplayer online games. *Future Generation Computer Systems* 25, 7 (2009), 785–793.

[80] C. Qu, R. N. Calheiros, and R. Buyya. 2016. A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications* 65 (2016), 167–180.

[81] M. Richards. 2015. Microservices vs. service-oriented architecture. O'Reilly Media.

[82] RightScale. 2016. Understanding the voting process. Retrieved from https://support.rightscale.com/12-Guides/RightScale_101/System_Architecture/RightScale_Alert_System/Alerts_based_on_Voting_Tags/Understanding_the_Voting_Process/.

[83] Y. Rochman, H. Levy, and E. Brosh. 2014. Efficient resource placement in cloud computing and network applications. *SIGMETRICS Performance Evaluation Review* 42, 2 (Sept. 2014), 49–51. DOI:http://dx.doi.org/10.1145/2667522.2667538

[84] G. Rodolakis, S. Siachalou, and L. Georgiadis. 2006. Replicated server placement with QoS constraints. *IEEE Transactions on Parallel and Distributed Systems* 17, 10 (Oct. 2006), 1151–1162. DOI:http://dx.doi.org/10.1109/TPDS.2006.145

[85] N. Roy, A. Dubey, and A. Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of 2011 IEEE International Conference on Cloud Computing (CLOUD'11)*. IEEE, 500–507.

[86] H. Rui, G. Li, M. M. Ghanem, and G. Yike. 2012. Lightweight resource scaling for cloud applications. In *Proceedings of 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'12)*. 644–651.

[87] K. Salah, K. Elbadawi, and R. Boutaba. 2016. An analytical model for estimating cloud resources of elastic services. *Journal of Network and Systems Management* 24, 2 (April 2016), 285–308. DOI:http://dx.doi.org/10.1007/s10922-015-9352-x

[88] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth. 2013. A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC'13)*. ACM, New York, Article 6, 10 pages. DOI:http://dx.doi.org/10.1145/2494621.2494628

[89] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy. 2015. SpotCheck: Designing a derivative IaaS cloud on the spot market. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys'15)*. ACM, New York, Article 16, 15 pages.

[90] U. Sharma, P. Shenoy, and D. F. Towsley. 2012. Provisioning multitier cloud applications using statistical bounds on sojourn time. In *Proceedings of the 9th International Conference on Autonomic Computing (ICAC'12)*. ACM, New York, 43–52.

[91] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. 2011. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 5.

[92] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. 2010. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceedings of the 7th International Conference on Autonomic Computing (ICAC'10)*. ACM, New York, 21–30.

[93] S. Spinner, S. Kounev, X. Zhu, L. Lu, M. Uysal, A. Holler, and R. Griffith. 2014. Runtime vertical scaling of virtualized applications via online model estimation. In *2014 IEEE 8th International Conference on Self-Adaptive and Self-Organizing Systems*. 157–166. DOI:http://dx.doi.org/10.1109/SASO.2014.29

[94] S. N. Srirama and A. Ostovar. 2014. Optimal resource provisioning for scaling enterprise applications on the cloud. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom'14)*. 262–271. DOI:http://dx.doi.org/10.1109/CloudCom.2014.24

[95] G. Tesauro. 2005. Online resource allocation using decompositional reinforcement learning. In *Proceedings of AAAI Conference on Artificial Intelligence*. Vol. 5. 886–891.

[96] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. 2007. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing* 10, 3 (2007), 287–299.

[97] M. Tortonesi and L. Foschini. 2016. Business-driven service placement for highly dynamic and distributed cloud systems. *IEEE Transactions on Cloud Computing* PP, 99 (2016), 1–1. DOI:http://dx.doi.org/10.1109/TCC.2016.2541141

[98] S. Upendra, P. Shenoy, S. Sahu, and A. Shaikh. 2011. A cost-aware elasticity provisioning system for the cloud. In *Proceedings of 2011 31st International Conference on Distributed Computing Systems (ICDCS'11)*. 559–570.

[99] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. 2008. Agile dynamic provisioning of multitier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 3, 1 (2008), 1.

[100] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. 2012. DejaVu: Accelerating resource allocation in virtualized environments. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVII'12)*. ACM, New York, 423–436.

[101] D. Villela, P. Pradhan, and D. Rubenstein. 2007. Provisioning servers in the application tier for e-commerce systems. *ACM Transactions on Internet Technology* 7, 1 (Feb. 2007), Article No. 7.

[102] C. Wang, A. Gupta, and B. Urgaonkar. 2016. Fine-grained resource scaling in a public cloud: A tenant's perspective. In *Proceeding of 2016 IEEE 9th International Conference on Cloud Computing (CLOUD'16)*. 124–131.

[103] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. M. Lau. 2012. Scaling social media applications into geo-distributed clouds. In *Proceedings of 2012 IEEE INFOCOM*. 684–692.

[104] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen. 2014. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers* 16, 1 (2014), 7–18. DOI: http://dx.doi.org/10.1007/s10796-013-9459-0

[105] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler. 2015. Predicting service metrics for cluster-based services using real-time analytics. In *Proceedings of 2015 11th International Conference on Network and Service Management (CNSM'15)*. 135–143. DOI: http://dx.doi.org/10.1109/CNSM.2015.7367349

[106] L. Yazdanov and C. Fetzer. 2012. Vertical scaling for prioritized vms provisioning. In *Proceedings of 2012 2nd International Conference on Cloud and Green Computing (CGC'12)*. IEEE, 118–125.

[107] L. Yazdanov and C. Fetzer. 2013. VScaler: Autonomic virtual machine scaling. In *Proceedings of 2013 IEEE 6th International Conference on Cloud Computing (CLOUD'13)*. 212–219.

[108] Y. You, G. Huang, J. Cao, E. Chen, J. He, Y. Zhang, and L. Hu. 2013. GEAM: A general and event-related aspects model for Twitter event detection. In *Proceedings of the 14th International Conference on Web Information Systems Engineering, Part II (WISE'13)*. Springer, Berlin, 319–332. DOI: http://dx.doi.org/10.1007/978-3-642-41154-0_24

[109] Q. Zhang, L. Cherkasova, and E. Smirni. 2007. A regression-based analytic model for dynamic resource provisioning of multitier applications. In *Proceedings of the 4th International Conference on Autonomic Computing (ICAC'07)*. 27–27.

[110] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. 2013. Dynamic service placement in geographically distributed clouds. *IEEE Journal on Selected Areas in Communications* 31, 12 (Dec. 2013), 762–772. DOI: http://dx.doi.org/10.1109/JSAC.2013.SUP2.1213008

[111] Q. Zhu and G. Agrawal. 2012. Resource provisioning with budget constraints for adaptive applications in cloud environments. *IEEE Transactions on Services Computing* 5, 4 (2012), 497–511. DOI: http://dx.doi.org/10.1109/TSC.2011.61

Received September 2016; revised September 2017; accepted September 2017