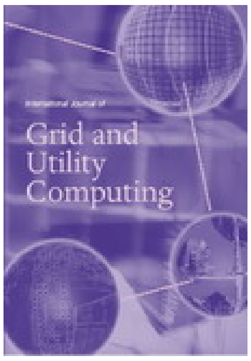


International Journal of Grid and Utility Computing

ISSN online: 1741-8488 - ISSN print: 1741-847X
https://www.inderscience.com/ijguc

A survey on auto-scaling: how to exploit cloud elasticity

Marta Catillo, Umberto Villano, Massimiliano Rak

DOI: 10.1504/IJGUC.2022.10049101

Article History:

Received:	05 November 2020
Accepted:	06 May 2021
Published online:	21 March 2023

A survey on auto-scaling: how to exploit cloud elasticity

Marta Catillo* and Umberto Villano

DING,
Università Del Sannio,
Benevento, Italy
Email: marta.catillo@unisannio.it
Email: villano@unisannio.it
*Corresponding author

Massimiliano Rak

Università Della Campania Luigi Vanvitelli,
Caserta, Aversa, Italy
Email: massimiliano.rak@unicampania.it

Abstract: Elasticity plays an essential role as far as the wide diffusion of cloud computing is concerned. It enables a cloud application deployment to 'scale' automatically, adapting to workload changes, guaranteeing the performance requirements with minimum infrastructure leasing costs. However, auto-scaling poses challenging problems. This paper gives a detailed overview of the current state of the art on auto-scaling. Firstly, the key design points for auto-scaling tools are presented and discussed. Then, literature proposals and on-going research are dealt with. Finally, existing auto-scaling implementations, including those used by commercial cloud providers, are reviewed.

Keywords: cloud computing; auto-scaling; resource provisioning; elasticity; QoS; service level agreement; scalable applications.

Reference to this paper should be made as follows: Catillo, M., Villano, U. and Rak, M. (2023) 'A survey on auto-scaling: how to exploit cloud elasticity', *Int. J. Grid and Utility Computing*, Vol. 14, No. 1, pp.37–50.

Biographical notes: Marta Catillo is a PhD student in Information Technologies for Engineering at University of Sannio. She received the MSc degree in Computer Engineering from the University of Sannio in 2019. Her research interests include parallel architectures and computer security.

Umberto Villano is Full Professor and Past Dean at the University of Sannio, Benevento, Italy. He received the Laurea degree in Electronic Engineering *cum laude* from the University of Naples in 1983. His research interests include parallel architectures, tools for parallel and distributed programming and performance evaluation, grid and cloud computing, security of cloud architectures.

Massimiliano Rak is currently an Associate Professor at the Information Engineering Department of the University of Campania Luigi Vanvitelli, Italy. He received the Laurea degree in Computer Science Engineering at the University of Naples Federico II in 1999. He got the PhD degree in Computer Engineering from the Second University of Naples (now University of Campania Luigi Vanvitelli) in 2002. His research interests include performance evaluation of computing systems, parallel and distributed software engineering and security of information systems.

This paper is a revised and expanded version of a paper entitled 'Auto-scaling in the Cloud: Current Status and Perspectives' presented at the '14th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2019)', 7–9 November 2019, University of Antwerp, Antwerp, Belgium.

1 Introduction

Cloud computing has well established in the last few years. Even if historically the initial interest for this computing paradigm was aroused by the search for low-cost and never-obsolescent computing resources, currently elasticity is playing an essential role. Elasticity is indeed an intrinsic feature of cloud environments (Mell and Grance, 2011), enabling a cloud application deployment to adapt to workload changes, ensuring Quality of Service (QoS) requirements with minimum infrastructure leasing costs.

Maybe the most elusive among the traditional design goals of distributed systems is scalability. A scalable system can be adapted to different load levels by adding or, not less interestingly, removing resources. As is well known, scalability can be exploited horizontally, increasing/decreasing the number of compute nodes/servers, or vertically, upgrading/ downgrading the nodes. As will be discussed later, depending on the virtualisation level, the compute nodes can be *Virtual Machines* (VMs) or *containers*. However, in traditional computing environments the scaling requires non-trivial and costly system reconfiguration. In clouds, scaling a given deployment can be performed rapidly and possibly in a completely automated manner. Cloud 'elasticity' means to look at the leased deployment configuration as to a rubber band, to be stretched when the load is high, and released when it decreases. Stated another way, elasticity paves the way to the design of software systems that are able to guarantee high QoS levels under highly variable, possibly unexpected, workload bursts, at the same time minimising hardware leasing costs.

However, even if cloud elasticity is a powerful enabling factor, obtaining optimal scaling remains challenging. Simply stated, the problem is to devise automatic or semi-automatic tools to find out *when* and *how to scale*. These tools, which allow to trigger scaling actions and to change the amount of leased computing resources according to some kind of user-generated or self-managed, autonomic input, are called *auto-scalers*. Auto-scalers can be extraordinarily powerful instruments, but their adoption poses challenging issues that need to be addressed. In particular:

- Workload forecasting is surely an important matter. To avoid QoS losses, it is desirable to acquire additional resources before an actual workload increase. This can be achieved only by workload prediction;
- Accurate estimation of the resources actually needed is currently an open issue. Resource under-provisioning will inevitably worsen performance; resource over-provisioning reduces computing efficiency. In both cases, the undesired effect is to incur in unnecessary costs.

Therefore, there are a number of complex objectives that an auto-scaler should achieve by balancing QoS and costs. Currently the issues linked to auto-scaling are not widely understood, and a reference standard or implementation is still lacking. Each commercial cloud provider has its own interpretation of the problem and its own way of addressing it. Most of the times, the available solutions are far less than optimal, and suspiciously neglect leasing costs.

In the light of the above, in order to design an efficient auto-scaler, it is necessary to take targeted decisions. However, there are no general 'best practices' that allow to design a perfect auto-scaler. Overall, there are some key points that every designer should consider when building an auto-scaler. These are:

- *When to scale and how to estimate resources?* In other words, when the scaling operation is triggered and how the resources needed are estimated to provide the desired QoS level;
- *How to scale?* An auto-scaler can scale a deployment both horizontally and vertically;
- *Which metrics should be monitored?* This is strictly related to the context, since in each different scenario some metrics may be more meaningful than others. A careful selection of the metric to be monitored is indeed necessary to avoid unnecessary overheads;
- *How to dynamically adapt to changes?* In general, bursty workloads are unpredictable; therefore, a good auto-scaler should be capable of adapting to load in a timely manner. However, due to its complexity, adaptability is not a common feature in existing scalers;
- *What is the virtualisation level?* The layer of virtualisation of hardware resources might be different. Current advancements in OS-level virtualisation have made viable the use of containers along with traditional VM-based solutions.

In light of the above, finding an optimal solution is a complex matter. In general, as stated above, there is no universal parameter set to consider for a complete auto-scaler design. The goal of this paper is to give an overview of the current state of the art on auto-scaling, pointing out the key design points and discussing the on-going research on this topic, so as to stimulate and support the production of better auto-scaler tools.

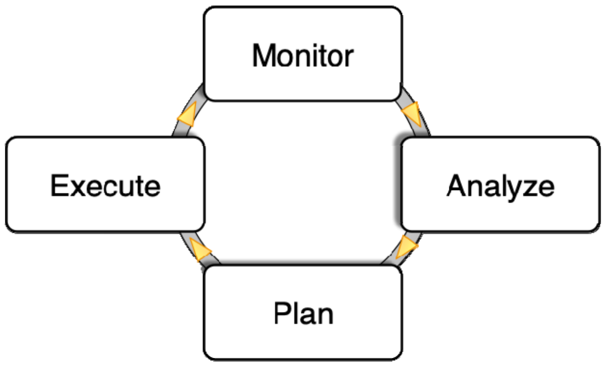
Over time, a few auto-scaling surveys and design taxonomies have been proposed, such as Lorido-Botran et al. (2014); Qu et al. (2018) and Singh et al. (2019). Our approach here is slightly different. More than a mere systematic description of all the scaling factors to be considered, our goal is to identify the crucial auto-scaling key points and then understand how these are mapped on recent literature and implementations.

In this paper, after a formalisation of the auto-scaling problem in Section 2, we focus on scaling timing in Section 3, which allows to understand '*when to scale*' and on techniques for resource estimation in Section 4. We describe the metrics that can be used to trigger the scaling actions in Section 5 and the scaling methods that allow to understand '*how to scale*' in Section 6. We also consider the adaptivity in Section 7 and the virtualisation level that can be adopted by an auto-scaler in Section 8. In the second part of the paper, we present literature solutions organised into four macro-categories in Section 9. Each of these identifies the researchers vision to improve one or more factors such as cost, resource optimisation, Service Level Agreement (SLA) violation, etc. Later, we deal with existing auto-scaling implementations, including those used by several commercial providers in Section 10. Finally, the main open research issues and directions are traced and discussed in Section 11 and our conclusions are drawn in Section 12.

2 The auto-scaling problem and the MAPE loop

In the literature, the auto-scaling problem is typically addressed as an autonomous control problem by using the Monitor-Analyse-Plan-Execute (MAPE) loop as a reference model (Maurer et al., 2011) (see Figure 1). During the *monitoring* phase, an information gathering activity is planned to assess the status of resource utilisation. This task requires the measurement of purposely-selected metrics about system and application current state. During the *analysis* phase, the information collected is analysed to be used later on. Scaling decisions are taken by avoiding oscillations that occur when scaling actions are carried out too quickly. The analysis process could be even more complex, if machine learning techniques for knowledge discovery from the information gathered by monitoring are adopted (Amiri and Mohammad-Khanli, 2017). The results obtained during the analysis phase are subsequently matched to rules defined by the application provider.

Figure 1 The MAPE loop



During the *planning* phase, the scaling decisions are taken. As will be discussed in Section 6, it is possible to choose between *horizontal* and *vertical scaling*. In cloud systems, horizontal scaling is the most widely used approach; it adjusts the number of node instances, e.g., VMs and/or containers, by acquiring further nodes (scaling out) or by releasing idle nodes (scaling in). Vertical scaling, instead, involves adding (scaling up) or subtracting (scaling down) to/from existing node resources as compute cores or RAM.

The last phase of the MAPE loop is the *execution* one, which involves the execution of the scaling decision previously taken. Conceptually, *execution* is an easy phase, but there are hidden complexities due to choices to be made in the presence of multiple providers with data centres in different geographical regions.

As a matter of fact, many auto-scaling techniques are MAPE-loop-based (Qu et al., 2018). Considering all the aspects mentioned above, it is clear that the choice of an optimal auto-scaling strategy is challenging. This complexity is also reinforced by the presence of multiple cloud providers with an extraordinary variety of cost plans.

3 Scaling timing

The first factor to consider regarding the '*when to scale*' key point is the timing of the scaling, i.e., when the scaling operation is triggered and how long does it take. This is a crucial issue especially when scaling up/out. A delay in the availability of additional resources required to cover load peaks can lead to unacceptable low performance and QoS violation. On the other hand, delays encountered when scaling down/in lead to unnecessary costs, but this is often only a secondary concern. For this reason, in this section we will consider mainly the timing of up/out scaling.

The 'enrolment' of an additional VM may require few minutes to setup an image, boot the OS, and launching the software. Understanding *when* to trigger scaling is a critical decision, due to the delay between the instant in time when the scaling decision is taken and the instant when it becomes effective. This delay could have an impact on both Quality of Service (QoS) and costs. The approaches used to decide *when* to scale take two forms: *reactive* and *proactive*. Sometimes *hybrid* approaches are adopted, which combine both these scaling policies.

Reactive scalers observe workload changes and act accordingly. The changes are detected using the past values of a set of monitored performance indicators. Consequently, the provisioning of resources is adjusted. This approach is cost-saving, as the exact amount of resources required are provisioned. On the other hand, resource provisioning takes time. Therefore, there is a risk of outages if provisioning is not fast enough. In order to 'cover' rapid spikes of load, an obvious solution is over-provisioning. Unfortunately, this rises the costs. A trade-off solution might involve the use of a threshold (more on this later). Examples of reactive scalers are proposed in Kumar and Gondhi (2017) and Han et al. (2014).

In *proactive scalers* the system provisions additional resources *before* the actual increase in load in order to take early scaling decisions. There are two common approaches to proactive scaling: scaling based on *scheduling* and *predictive* scaling. In the first case, the user provides a scheduling of the required resources. For example, it is possible to double the capacity of an on-line shop one hour before a big promotion; in this case, the goal is to plan scaling decisions. The second approach, instead, involves the prediction of the future loads by using historical data and forecasting techniques. In this context, the accuracy of the prediction algorithm is an issue. In the literature there are many solutions based on workload prediction, as Alipour and Liu (2017). In particular, this work shows two solutions to predict CPU demand by a machine-learning approach. Overall, the drawbacks of proactive scalers are high costs, because they tend to add more capacity than actually needed.

In *hybrid scalers* the system exploits a combination of reactive and proactive approaches. In many situations it is possible to obtain forecasts as workload patterns can be found, but it is not possible to deal with unplanned bursts. In these cases, mixed scaling approaches would be beneficial. Two solutions that propose hybrid scalers are shown in Moore et al. (2013b, 2013a).

4 Techniques for resource estimation

In light of the discussion in the Section 3 above, it can be said that auto-scalers (whether reactive, proactive or hybrid) react to stimuli linked to the system current and/or expected load, trying to match at best an estimation of the resources needed to provide the desired QoS level. The techniques used for resource estimation can be classified as follows:

- Threshold-based rules;
- Time series analysis;
- Control theory;
- Queuing theory;
- Reinforcement learning.

These approaches will be orderly dealt with below.

With *Threshold-based rules*, the scaling decisions are triggered by means of predefined thresholds. Threshold-based rules follow a simple *reactive* approach. For example, assuming the application is CPU bound, provisioning can be started at 50% CPU utilisation. In this case, assuming as a threshold value 50% CPU utilisation, it is possible to have always more capacity than needed. This can be considered a healthy safety factor. Threshold-based approaches have become quite popular due to their (apparent) simplicity. Nevertheless, there is no systematic method that allows the thresholds to be defined rigorously. Hence, the effectiveness of these settings under highly variable workloads is questionable. Some conventional reactive threshold-based approaches are presented in Biswas et al. (2015); Casalicchio and Silvestri (2013) and Beltrán (2015). Moreover, the use of thresholds is the only approach widely used in the commercial auto-scaling systems such as AWS (https://aws.amazon.com/autoscaling), Google (https://cloud.google.com/appengine/docs), Azure (https://azure.microsoft.com/en-in/features/autoscale), and Kubernetes (https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler). Many open-source cloud-computing platforms, such as Eucalyptus (https://docs.eucalyptus.cloud/eucalyptus/4.4.5/user-guide-4.4.5.pdf), OpenNebula (https://docs.opennebula.io/5.8/advanced_components/application_flow_and_auto-scaling/appflow_elasticity.html) and Nimbus (http://www.nimbusproject.org/doc/phantom/latest/protocol.html) also adopt a threshold-based reactive approach.

The overall process is based on the selection of the right working rules to set a threshold. These rules are tuned on the basis of one or more performance metrics. The most popular metrics are average CPU load of the VMs, response time and input request rate. It is clear that the performance of rule-based approaches strongly depends on these parameters, and, as a matter of fact, their use is not trivial. To avoid unwanted oscillations in the number of nodes and/or of the resources assigned to each of them, *duration* parameters are used to decrease the number of scaling actions. Duration defines how long the threshold conditions must be exceeded to trigger a scaling action. This issue is highlighted in Dutreilh et al. (2010). In Hasan et al. (2012), six parameters (four thresholds and two durations) are used in order to perform the right scaling action. It is also worth pointing out that the approaches based on thresholds require frequent tuning in order to perform the right scaling action. Therefore, they are suitable for fairly regular load patterns, but turn out to be less effective for bursty workloads.

Time series are very useful to represent the change of a measured parameter over time. A time series is a sequence of samples, produced at uniform time intervals. An example might be the number of requests for a server, sampled at one minute intervals. In general, *Time series analysis* is the dominant approach in the cloud workload prediction area. In this context, the effectiveness of prediction largely depends on parameters such as the monitoring-interval length and the size of the history window. Clearly, the accuracy of the model depends on the window input size. Sometimes a subset of these parameters, such as the size of the history window, is the input for a neural network (Islam et al., 2012), which, thanks to an off-line training, predicts the values of performance indicators adopted for scaling decisions. Hu et al. (2016) proposed a predictive scaling approach for VMs provisioning.

Time series analysis techniques are indeed appealing for implementing auto-scalers, as they allow to provision resources proactively. However, despite their potential, they often fail to provide satisfactory prediction accuracy, which strongly depends on system parameters (i.e., the target application, the input workload patterns, the history window, etc.).

The *Control theory* approach involves the creation of a model of the application. A controller (reactive or proactive) is defined to adjust automatically the required resources to the application demands. For example, Ghanbari et al., 2011) described a feedback controller used to guarantee the satisfaction of application constraints. In the literature many well-known controllers, including Proportional-Integral-Derivative (PID), Proportional-Integral (PI) and Integral (I), have been extensively used. For example, Lim et al. (2010) exploited an I controller in order to adjust the number of VMs based on average CPU usage. The Kubernetes auto-scaling mechanism (https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler) is also based on the control theory approach. Reactive controllers could be used in order to react to the observed workload change, but also proactive controllers as Model Predictive Control (MPC) are suitable in order to design effective scalers. For example, in Roy et al. (2011) an Autoregressive-Moving-Average (ARMA) model for workload forecasting is combined with the look-ahead controller to optimise the resource provisioning. The suitability of controllers for auto-scaling highly depends on the type of controller and the dynamics of the target system. The idea of having a controller that automates the provisioning process is sound, but devising a reliable one-fits-all model is a complex task.

Queuing theory is a traditional approach to computer system modelling. Network of queues have been widely applied in order to find the relationship between the jobs arriving and leaving a system. In particular, Han et al. (2014) followed this approach in order to estimate resource need. They exploit a greedy approach to modulate the number of nodes on the basis of the current load. In Ali-Eldin et al. (2012), instead, a cloud-hosted application is modelled as a G/G/N queue, where the number of nodes (servers, according to queuing theory terminology) N is variable. The model can be exploited in order to compute, for example, the necessary resources required to process a given input workload λ, or the mean response time for requests, given a particular configuration of nodes. Queues can also be exploited to model elastic applications, representing each server as a separate queue. An example that follows this approach is reported in Urgaonkar et al. (2008). In particular, it uses a network of G/G/1 queues (one per each node).

The information required for a queuing model, such as the input workload or service time, can be obtained by on-line monitoring or estimated using different methods. Zhang et al. (2007) used a regression-based approximation to estimate the CPU demand, based on the number and type (browsing, ordering or shopping) of client requests.

Queuing models have been extensively used to model applications and systems. Unfortunately, due to their fixed architecture, they are not very flexible to changes. As a matter of fact, any changes in structure or parameters is likely to be costly. This might be a problem for elastic applications that have to deal with a changing input workload and a varying pool of resources.

Reinforcement learning is an alternative to control theory adoption. Its main advantage is the requirement of no *a priori* knowledge or model of the application. When the deployment is scaled, a reward or a punishment is computed to identify how good the resource estimation had been (Sutton and Barto, 2018).

In general, proactive timing is assumed for all reinforcement learning approaches. Auto-scaling in the cloud based on reinforcement learning is surveyed in Garí et al. (2020). Horovitz and Arian (2018) propose simulation and real-application tests results by comparing *Q*-Learning, a model-free reinforcement learning algorithm, to the static threshold based method in a real (Kubernetes-based) environment. The Authors show that their improved *Q*-Learning algorithm successfully prevents SLA violations. John et al. (2019), instead, present a reinforcement learning-based algorithm that addresses two crucial problems in classical approaches such as *Q*-learning: slow convergence and lack of scalability. They exploit the technique of adaptive tile coding and workload forecasting in order to guarantee efficient utilisation of resources. The validity of the proposed method as compared to static, threshold-based and other reinforcement learning-based allocation schemes is shown by experiments on the Cloudsim platform (Calheiros et al., 2011).

Reinforcement learning techniques are a promising solution to the auto-scaling problem, since they do not require any application modelling. However, some reinforcement learning algorithms, such as *Q*-Learning, may lead to unacceptably poor performance, as they require a long training period before a good-enough solution is found. Even assuming that an optimal policy is found, any change of the environmental conditions (e.g., the workload pattern) requires re-adaption and further long training periods.

5 Scaling metrics

The techniques for resource estimation dealt with in Section 4 rely on the measurement of metrics that can be collected at different levels. There is no general rule for selecting scaling metrics. We classified the metrics according to the level at which the measurement is performed, as follows:

- *CSP level metrics*: metrics adopted to measure resources offered by Cloud Service Providers (CSPs), as CPU performance and network bandwidth;
- *Application level metrics*: metrics adopted to measure the resources leased by a specific cloud application (e.g., CPU and memory utilisation of acquired nodes);
- *Business logic level metrics*: metrics that are specific for the application under analysis, adopted to measure business logic performance.

The *CSP level metrics* are the metrics adopted for the measurement made by Cloud Service Providers. Measurements of such metrics are useful to characterise the global behaviour of a CSP. In fact, they are commonly adopted to compare different CSPs. Examples of these metrics are the bandwidth of the external network and/or the CPU performance in MIPS/FLOPS of a single core VM. Such metrics are typically measured by third party services (e.g., CloudHarmony (https://cloudharmony.com)) or by the CSPs themselves (Amazon's CloudWatch (https://aws.amazon.com/cloudwatch) offers many of such metrics). At the best of the authors' knowledge, these metrics are never adopted by auto-scalers, as they provide data too coarse-grained, even if they can be useful for the performance tuning of applications with well-known behaviour.

The *application level metrics* are associated to the resources acquired by Cloud Service Customers (CSCs) and are mostly indicators collected at operating system level or by the hypervisor. These types of metrics can be measured by the CSCs, using *ad hoc* tools and/or using services offered by the CSP, as CloudWatch. Many commercial scaling solutions exploit these types of metrics. For example, container orchestration systems as Kubernetes (https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler) or providers as Amazon (https://aws.amazon.com/autoscaling) consider infrastructure related metrics, such as CPU utilisation, as reference metrics. In particular, the CPU utilisation seems to be the most common indicator used in the design of auto-scalers. This aspect is also highlighted in some literature works such as Ye et al. (2017) and Liao et al. (2015). However, sometimes also other low-level indicators such as disk access, memory usage, or page faults are adopted. Lu et al. (2017) model an optimisation problem in order to minimise the VMs' configuration cost. In particular, they evaluate the migration delay occurring when a VM must be shut down and its data is transferred to a new VM by considering the size of the image, the bandwidth and the booting time of a VM, etc.

The *business logic level metrics* are collected on top of the application level and rely on the business logic. These metrics are more complex to analyse than the previous ones, as they require a deep knowledge of the application. In some cases, off-line application profiling may also be required. Chieu et al. (2009) proposed a dynamic scaling algorithm for web applications based on the number of active sessions, the request rate and mean response time. In Ilyushkin et al. (2017), instead, in order to infer the number of resources to be provisioned, the Authors consider as scaling metrics the response time, the request rate and the mean number of requests served per VM and per unit time. Finally, Assuncao et al. (2016) described a scheduling algorithm and auto-scaling triggering strategies based on *user patience*, a metric that estimates the perception end-users have of the QoS delivered by a service provider. This metric takes into account the ratio between expected and actual response times for each request. This approach reduces the resource costs while maintaining the perceived QoS at adequate levels.

It is worth noting that each indicator has strengths and weaknesses. For example, the CSP perspective approach cannot ensure that the application SLA constraints are met, since the low-level indicators used are not directly linked to the application performance. On the other hand, the business logic level approach does not consider low-level information that might be relevant to resource allocation. This is a strong motivation for the adoption by auto-scalers of hybrid approaches, that can consider metrics at different levels. Fernandez et al. (2014), in order to trigger the best scaling action, consider both high-level (such as response time of a specific service) and low-level metrics (such as % CPU utilisation). Similarly, Taherizadeh et al. (2019) proposed an approach based on both CSP and application level metrics to scale in a container-based environment.

6 Scaling methods

As anticipated in Section 2, there are two different approaches to application scaling: *horizontal scaling* and *vertical scaling*. The first one allows to lease additional compute nodes (*scaling out*), or to release idle/lightly loaded nodes (*scaling in*). Vertical scaling, instead, means to add (*scaling up*) or subtract (*scaling down*) compute 'power' (compute cores, RAM, etc.) to/from existing nodes. Each approach has strengths and weaknesses. In general, the availability and the implementation of the VM vertical and horizontal supports are subject to the particular CSP. For example, the Amazon Web Services (https://aws.amazon.com/autoscaling) provides only horizontal scaling out-of-the-box. Microsoft Azure (https://azure.microsoft.com/en-in/features/autoscale), instead, offers also support for vertical scaling. The Google Compute Engine Auto-scaler allows both vertical and horizontal auto-scaling of pods, the basic deployable objects in Kubernetes.

Although horizontal and vertical scaling are widely exploited by many auto-scalers in the literature, such as Li et al. (2020) and Incerto et al. (2018), there are also *hybrid* approaches that combine vertical and horizontal scaling. In light of the above, the scaling methods can be classified as follows:

Horizontal scaling: horizontal scaling is the most widely used approach because, although affected by the overhead for adding/removing nodes, allows a growth of the leased computing resources not bounded by the physical characteristics of the underlying hardware. Furthermore, it is of simple implementation. For example, the number of web servers can be easily modified by reconfiguring a load balancer. On the downside, scaling out is not suitable for all software systems and adds communication overhead. For example, scaling out a database by adding replicas in not an easy task.

Scaling out/in is a not-trivial optimisation problem if the cloud offers heterogeneous resources, e.g., VM with several different configurations (number of CPU cores, amount of memory, etc.). Some works dealing with this problem and oriented towards cost efficiency are Lu et al. (2017) and Sedaghat et al. (2013).

Vertical scaling: an advantage of vertical scaling concerns its simplicity in managing and installing/removing hardware resources, without the need to manage multiple instances of software components. On the other hand, it tends to be less dynamic then horizontal scaling as sometimes requires reboots. Moreover, scaling up may be more expensive than scaling out, as in most CSP offerings high-end server configurations tend to be very expensive.

Hybrid scaling: combining horizontal and vertical scaling can help to optimise both resource costs and reconfiguration overhead. In fact, vertical scaling is inevitably limited by the physical node characteristics, but entails low reconfiguration overheads. Horizontal scaling, instead, can scale the application to a higher throughput but the reconfiguration costs are higher. Dutta et al. (2012) suggested dividing the scaling problem in two phases, firstly optimising the size of the virtual machine and then finding accordingly the minimum number of instances. A novel combined approach is also proposed in Incerto et al. (2018).

7 Adaptivity

Bursty workloads are unexpected and unpredictable, and so the ability of a scaler to adapt to changes is a desirable feature. However, this is a difficult task to implement; in fact, most existing scalers are *non-adaptive*. The two possible categories are described below.

Non-adaptive: following this approach, a predefined schema is used to make auto-scaling actions. Examples of non-adaptive approaches are rule-based scaling methods. In this context, decisions are taken on the basis of the current input. For example, Amazon AWS (https://aws.amazon.com/autoscaling) follows a non-adaptive rule-based approach. A predefined static threshold triggers a scaling action; the threshold is static, and does not change unless the user updates it. Automatic adjustments of settings during production are not allowed.

Self-adaptive: the auto-scaler is capable of autonomously tuning its settings and to update its decisions according to new incoming information. For example, a self-adaptive scaler could adjust the thresholds for CPU utilisation and handle the creation of VMs according to the workload demand. An advantage of self-adaptivity is that it reduces the amount of off-line preparation required to tune an auto-scaler. However, the time to convergence of the adaptations could be long, causing poor performance during the early stages of training.

8 Virtualisation level

One of the core characteristics of the vast majority of clouds is the lease of virtualised computing resources, as bare-metal offerings are relatively rare. Virtualisation consists in abstracting hardware resources in order to enhance resource management. A *Virtual Machine* (VM) is an image file managed by the *hypervisor* that exhibits the behaviour of a separate unit by running a full abstraction of the operating system. One of the strengths of the use of VMs is that it guarantees a high degree of isolation. As a recent alternative to VMs, *container-based* or operating system-level virtualisation provides lightweight consolidation, isolation and quick provisioning of resources. The two approaches are described in the following.

Virtual machines: in general, VMs behave like a dedicated machine leased to host applications. Customarily an application hosted on VMs can be scaled horizontally and vertically. Every VM requires a guest operating system for running on a physical host. In a typical VM-based deployment, an auto-scaler may need to launch new VMs dynamically. In general, this process might require to boot the operating system of choice, affecting the application performance. Tighe and Bauer (2017) proposed a method for dynamic virtual machine provision capable of satisfying both CSP and CSC requirements. Another example is presented in Chieu et al. (2009), where a dynamic scaling algorithm for automated provisioning of VMs based on the number of user logins in a web application is proposed.

Containers: recent advancements in OS-level virtualisation have made the use of containers an attractive and viable solution. Containers represent a lightweight virtualisation that guarantees reasonable process isolation without the need for a guest operating system. The container provides similar resource allocation benefits as the VMs. However, containers are more portable as compared to the VMs, and the open-source Docker implementation (Merkel, 2014) is currently widely diffused. Unlike VMs, containers are able to boot quickly. This is mainly due to the fact that they do not require a guest operating systems.

Despite the rising popularity of containers, VM provisioning is still the solution most often adopted in literature solutions. In Ye et al. (2017), a scaling solution that features container-based virtualisation is presented. In particular, it is an auto-scaling framework for containerised elastic applications. Another auto-scaling case of study container-based is described in Klinaku et al. (2018). Both solutions include scaling actions based on increasing or reducing the number of containers.

Hybrid: as mentioned above, most of literature studies address scalability of VMs, neglecting containers. However, containers are often used on the top of virtual machines. In fact, studies that analyse the estimation of resources using a combination of VMs and containers are lacking. An example of *hybrid* approach is described in Al-Dhuraibi et al. (2018), where the problem of the joint scaling of VM and containers is considered.

9 Auto-scaling approaches in the literature

Following up the discussion on the main key points to be considered for devising an auto-scaling implementation, from here onward we will deal with existing solutions, starting from the ones that are the object of papers in the state-of-the-art literature. In this section we identify four macro categories of approaches to auto-scaling: *load prediction*, *resource-aware*, *SLA-aware* and *cost-aware*. Each of these identifies the researchers' vision to improve one or more factors, such as cost, resource optimisation, SLA violation avoidance, etc. This grouping is not due to a forced choice or is based on specific standards, since there is no commonly accepted reference. However, we think that it might be a good organisation in order to get a clear idea about the works that tend to improve the specific objectives. To summarise the findings, based on the taxonomy and explanation of the above sections, we list the features of the works surveyed in the following in Table 1.

9.1 Load prediction approaches

Estimating the use of resources in a context where there is a strong variability in the workload is quite complex. For many websites, e.g., it is not easy to plan load peaks, as it is necessary to consider the interleaving of a number of factors (time of day, day of week and other seasonal factors) that concur, along with unplanned ones, to the predictability of the load. It is indeed possible to take into account average load or maximum peaks, but each of the two solutions has disadvantages. In the first case, provisioning problems could arise in the presence of unexpected peaks; in the second, there could be a waste of resources if the workload remains under the peaks.

Table 1 A review of the auto-scaling properties of surveyed works

Work	*Macro-category*	*Scaling timing*	*Resource estimation*	*Scaling metrics*	*Scaling methods*	*Adaptivity*	*Virtualisation level*
Islam et al. (2012)	*load prediction*	proactive	time-series analysis	application level	horizontal	self-adaptive	VM
Herbst et al. (2013)	*load prediction*	proactive	control theory	business logic level	horizontal	self-adaptive	VM
Li et al. (2016)	*load prediction*	proactive	time-series analysis	application level	hybrid	self-adaptive	VM
Roy et al. (2011)	*load prediction*	proactive	control-theory	business logic level	horizontal	self-adaptive	VM
Bunch et al. (2012)	*load prediction*	proactive	control-theory	application/ business logic level	horizontal	self-adaptive	VM
Shariffdeen et al. (2016)	*load prediction*	proactive	time-series analysis	application/business logic level	horizontal	self-adaptive	VM
Huang et al. (2012)	*resource-aware*	proactive	time-series analysis	application level	N/A	self-adaptive	VM
Moltó et al. (2016)	*resource-aware*	reactive	threshold-based rules	application level	hybrid	self-adaptive	hybrid
Novak et al. (2019)	*resource-aware*	reactive	queuing theory	application level	hybrid	self-adaptive	VM
Liu et al. (2017)	*resource-aware*	reactive	threshold-based rules	business logic level	horizontal	non-adaptive	VM
Hasan et al. (2012)	*resource-aware*	reactive	threshold-based rules	application level	hybrid	non-adaptive	VM
Qavami et al. (2014)	*resource-aware*	proactive	queuing theory	application level	horizontal	self-adaptive	VM
García et al. (2014)	*SLA-aware*	reactive	threshold-based rules	business logic level	horizontal	non-adaptive	VM
Alzahrani et al. (2016)	*SLA-aware*	proactive	time-series analysis	application/business logic level	hybrid	self-adaptive	hybrid
Gujarati et al. (2017)	*SLA-aware*	proactive	queuing theory	business logic level	horizontal	self-adaptive	hybrid
Tran et al. (2017)	*SLA-aware*	proactive	time-series analysis	application/business logic level	horizontal	self-adaptive	VM
Souza and Netto (2015)	*SLA-aware*	reactive	threshold-based rules	business logic level	vertical	non-adaptive	N/A
Yaqub et al. (2014)	*SLA-aware*	reactive	threshold-based rules	application/business logic level	horizontal	non-adaptive	container
Moldovan et al. (2016)	*COST-aware*	reactive	threshold-based rules	business logic level	horizontal	non-adaptive	VM
Catillo et al. (2021)	*COST-aware*	hybrid	N/A	business logic level	horizontal	non-adaptive	VM
Lesch et al. (2018)	*COST-aware*	proactive	control theory	business logic level	vertical	self-adaptive	VM
Kriushanth and Arockiam (2014)	*COST-aware*	reactive	threshold-based rules	business logic level	horizontal	non-adaptive	VM
Mao et al. (2010)	*COST-aware*	reactive	queuing theory	business logic level	vertical	non-adaptive	VM
Horn and Skrzypek (2018)	*COST-aware*	proactive	N/A	business logic level	horizontal	self-adaptive	VM

In the literature there are many works that deal with the problem of the prediction of the incoming load, in order to obtain careful management of resources. Islam et al. (2012) proposed prediction-based resource measurement and provisioning strategies by using Neural Networks and Linear Regression. The goal is to satisfy the incoming demand by making the best use of resources. The solution, besides providing accurate load forecasts, makes also it possible to predict the resource demand ahead of the VM instance setup time. Herbst et al. (2013) described a novel self-adaptive approach that selects load forecasting methods depending on the context. The approach is based on decision trees and direct feedback cycles. The results obtained from the experimentation show that the context selection of a forecast method reduces the error of the load previsions, compared to the results obtained by a statically-selected forecasting method. Li et al. (2016) proposed a load prediction algorithm for automatic scaling. The algorithm is combined with Linear Regression and improved Knuth-Morris-Pratt string matching. The authors show that their approach can solve successfully the problem of resource allocation. Roy et al. (2011), instead, use predictive techniques to automatically scale resources by exploiting a look-ahead resource allocation algorithm. In particular, they propose a model based on control theory in order to predict future workloads. Empirical results show the validity of the proposal for both cloud users and providers. Finally, Bunch et al. (2012) and Shariffdeen et al. (2016) described two different workload prediction methods for efficient auto-scaling in PaaS systems.

9.2 *Resource-aware approaches*

The progressive diffusion of cloud computing, with data centers spread over different geographical areas, calls for the optimisation of resource provisioning policies. One of the main challenges in the field of resource provisioning concerns finding the distribution of resources to applications that reduces power consumption and costs. Many recent studies have addressed this problem.

Huang et al. (2012) proposed a resource prediction model based on double exponential smoothing. Besides considering the current state of resources, the system also takes into account historical data. The experiments performed on the CloudSim simulator show a good model accuracy. Moltó et al. (2016) instead propose a system providing automatic vertical elasticity, that adapts the memory size of the VMs to their memory consumption. The system uses live migration to prevent overload scenarios, thus preventing downtime for VMs. Novak et al. (2019) described an architecture for auto-scaling VMs quickly by using cloud functions and a reactive scaling algorithm. Cloud functions are available from cloud providers as temporary resources to manage the delay in starting VMs; the main feature of the reactive scaling algorithm is that it does not require the setting of a threshold. The Authors show the validity of the proposal by implementing the system on both AWS and Azure. In order to ensure the stability of service performance, Liu et al. (2017) proposed a framework for cloud resource management by exploiting two novel turnaround time-driven auto-scaling mechanism. As an extension of a previous work (Liu et al., 2016), the system provides both dynamic and schedule-based auto-scaling and shows good performance. Hasan et al. (2012) presented Integrated and Autonomic Cloud Resource Scaler (ICARS), a cloud resource auto-scaling system. The power of the proposal is that it integrates performance metrics from multiple domains for effective scaling of resources. This allows optimised scaling by avoiding outages. Finally, Qavami et al. (2014) focused on resource allocation by proposing a learning-based system called Smart Virtual Machine Provisioner (SVMP). They propose a dynamic resource provisioning following a heuristic Markovian approach. The experimental results show the effectiveness of the solution.

9.3 *SLA-aware approaches*

Autonomic provisioning should allow to meet the requirements specified by the Service Level Agreement signed by application providers and cloud service providers. In an SLA, the quality of service is specified as non-functional requirements of services. Inadequate values of QoS may lead to SLA penalties.

Many SLA-aware resource provisioning techniques have been proposed. García et al. (2014) proposed *Cloudcompass*, an SLA-aware cloud platform that manages the complete resource life cycle. In particular, it allows cloud service providers adopting a generic SLA model to manage higher-level metrics, closer to the end user perception. Cloudcompass also allows to correct QoS violations by exploiting the elasticity features of cloud systems. Alzahrani et al. (2016) describe Energy Based Auto-Scaling (EBAS), as a novel resource auto-scaling approach that takes into account Service Level Agreements. The proposed system, besides estimating the amount of resources that are needed for computation, enables the CPU to be aware of the SLA constraints. The experimental results show that the system is well designed to meet SLA conditions. Gujarati et al. (2017) presented Swayam, a fully distributed auto-scaling framework that cares about SLA compliance and resource efficiency. The analysis is based on Microsoft Azure Machine Learning, a commercial MLaaS platform. Extensive experimentation on 15 popular services shows the validity of the proposal. Tran et al. (2017) described a proactive cloud scaling model based on fuzzy time series and SLA awareness. In particular, in addition to using a fuzzy approach, they exploit a genetic algorithm and a neural network to process historical monitoring time series data; the scaling action is triggered by the occurrence of SLA violations. Souza and Netto (2015) presented a study on the effectiveness of auto-scaling driven by data generated by the supported application. In particular, they use SLA in order to estimate the necessary amount of resources. The proposed algorithm can reduce the number of SLA violations up to 95%. Finally, Yaqub et al. (2014) presented a capacity planning for SLA-aware resource management in PaaS clouds. The novelty of the contribution is the application of metaheuristic local search. The Authors claim that is one of the first works on the subject based on multiple metaheuristic algorithms and holistic evaluations.

9.4 *COST-aware approaches*

The use of auto-scaling mechanisms that try to meet cost requirements is an active research topic. As a matter of fact, optimising the cost of a scalable application is not easy. The use of a VM is often billed by the hour, or based on each GB of generated I/O traffic. For these reasons, it is crucial to develop scaling strategies taking into account the billing cycles.

Moldovan et al. (2016) introduced a model for capturing the pricing schemes of cloud services. The solution is useful for the developers of scalable applications for public clouds, as it allows to monitor costs and to develop cost-aware scalability controllers. Catillo et al. (2021) proposed an analysis method based on off-line benchmarking that allows to define scaling policies to be used by auto-scalers. The approach consists in benchmarking the web application to discover the load processing capacities of each component, making it easier to apply scaling policies in the presence of load variations. The analysis enables to identify the trade-offs between costs and quality of service of the application even in multiple deployment configurations. Lesch et al. (2018) described a cost-aware approach for autonomic resource management called FOX. It operates as a mediator between the application and the cloud in order to reduce the charged costs. The experimental results show that FOX is able to reduce the charged costs by increasing the accounted instance time for the Amazon EC2 charging model. Kriushanth and Arockiam (2014) introduced a dynamic rule-based auto-scaling mechanism in order to reduce the cost of the VM instances. The results show that the proposed approach reduces the cost of the service and SLA violations related to cost. A mechanism to dynamically scale cloud computing instances, based on deadline and budget information, is presented in Mao et al. (2020). The scaling actions are triggered considering performance and budget of a cloud application. The system is deployed on the Windows Azure platform and evaluated by using both simulation and a real scientific application. Finally, Horn, G. and Skrzypek (2018) presented MELODIC, a framework that supports cost-aware auto-scaling. It finds a good initial deployment in the cloud and continuously optimises it according to the variable execution context, possibly taking into account the cost in its utility function.

10 Commercial scalers

In this section, we analyse some existing auto-scaling solutions, including those used by several commercial cloud providers. In particular, we focused on the implementations provided by the main CSPs: *Amazon AWS*, *Microsoft Azure*, *Rackspace* and *Google*.

10.1 Amazon AWS

Amazon Web Service (AWS) offers an auto-scaling service in the IaaS Elastic Compute Cloud (EC2) public cloud (https://aws.amazon.com/autoscaling). An EC2 instance is a virtual server for running applications on the AWS infrastructure. As far as elasticity is concerned, a key component is the *Auto Scaling Group (ASG)*. This is characterised by the configuration of the virtual machines that will be part of the group. The ASG maintains EC2 instances in the group by performing a periodic health check. If any instance becomes unhealthy, it is stopped and replaced with another instance within the group. Another fundamental component is the *Launch Configuration*. It is a template used by Auto Scaling Group to launch EC2 instances. It allows to specify the Amazon Machine Image (AMI) (instances type, key pair, security groups, etc.) during the launch configuration step. Finally, the *Scaling Plans* specify when and how to scale. There are several ways to scale within the ASG:

- *Maintaining current instance level at all times*: maintenance of a fixed number of running instances in the ASG. If an instance becomes unhealthy, it is immediately replaced.
- *Manual scaling*: once the group capacity is specified, auto-scaling maintains the instances with updated capacity.
- *Scale based on schedule*: this approach can be used when traffic peaks are expected. In this case, scaling actions are performed at specific times.
- *Scale based on demand*: following this approach, resources scale by a scaling policy. It is possible to scale in or out considering specific parameters (CPU utilisation, Memory, Network In and Out, etc.).
- *Use predictive scaling*: this approach combines predictive scaling and dynamic scaling (proactive and reactive approaches, respectively) in order to scale faster.

Amazon AWS supports only horizontal auto-scaling out-of-the-box. However, vertical auto-scaling is possible by harnessing *AWS Ops Automator*. It is a solution that features vertical scaling for Amazon EC2 instances by exploiting the AWS services (e.g., *CloudWatch* and *Lambda*).

Moreover, Amazon EC2 Container Service (ECS), a high performance container management service that supports Docker containers, allows to run easily applications on a managed cluster of Amazon EC2 instances. ECS provides auto-scaling for containers. It allows containerised services to handle variable load over time and react in real-time to dynamic workloads.

10.2 Microsoft Azure

Azure's auto-scaling allows to set up rules to automatically scale applications (both horizontally and vertically) without manual intervention (https://azure.microsoft.com/en-in/features/autoscale). Rules can be based on time (scaling is carried out at specified times), or on two types of metrics:

1) *Resource metrics*: related to usage within Azure (memory, CPU and disk usage, thread count, queue length). It is possible to set Azure auto-scaling to scale up or down based on these usage parameters.
2) *Custom metrics*: these are metrics produced by the application itself. If they are sent to *Application Insights* (a performance monitoring service by Microsoft), they can be used to make decisions on whether to scale or not.

Microsoft Azure also provides auto-scaling for *Azure Kubernetes Service* (*AKS*), a service for running containers in the cloud. This lets developers elastically provision quick-starting pods inside of *Azure Container Instances* (*ACI*), a service that enables a developer to deploy containers on the Microsoft Azure public cloud without having to provision or manage any underlying infrastructure.

10.3 Rackspace

Rackspace Auto Scale is written in Python and relies on the *Rackspace Cloud Servers*, *Rackspace Cloud Load Balancers* and *Rackspace RackConnect* v3 APIs (https://www.rackspace.com/cloud/auto-scale). Rackspace only supports horizontal scaling. The scaling events can be managed both with scaling rules, which can be defined through the monitoring system, and through a schedule that can be suitably configured. A *scaling group* is a set of identical servers (and optionally a load balancer), characterised by the following components:

- *Scaling group configuration*: the group name, cooldown (configured period of time that must pass between actions) time limit, minimum and maximum number of needed servers.
- *Launch configuration*: if a scaling event is intercepted, the specific server configurations are managed.
- *Scaling policy*: specifies the actions of the policy.
- *Webhook (capability-based URL)*: triggers a scaling policy.

At the time of writing, Rackspace does not provide a container auto-scaling service.

10.4 Google

Auto-scaling in the Google Cloud platform is a feature of the *instance group*. In particular, a group is composed of homogeneous instances, created from a common instance template. The auto-scaling service is offered by Compute Engine, which supports only horizontal scaling (https://cloud.google.com/appengine/docs). Possible scaling policies include:

- *CPU utilisation*: the auto-scaling event is linked to the average CPU utilisation of a group of virtual machines.
- *Load balancing serving capacity*: auto-scaling is based on load balancing serving capacity by monitoring the serving capacity of an instance group. There is a scaling event if the VMs are over or under capacity.
- *Stackdriver monitoring metrics*: auto-scaling is based on a standard metric provided by Google's *Stackdriver Monitoring* (a monitoring service natively integrated with Google Cloud Platform), or on any custom metrics.

Finally, it is worth mentioning that *Google Kubernetes Engine*, a managed environment for deploying, managing, and scaling containerised applications using Google infrastructure, allows for both horizontal and vertical scaling of pods.

11 Research directions

The availability of (automated) elasticity, commonly offered by cloud-based infrastructures, has boosted the interest for new research topics. In the past, the auto-scaling problem was addressed mainly from the data centre management point of view, and it was strictly related to the resource scheduling problem. In particular, researchers focused their interest on the optimal allocation of many-task applications over existing resources, in order to minimise resource usage, reduce energy consumption and so on. According to the state of art we have summarised in the previous sections, currently auto-scaling in infrastructures is no longer a technological issue: existing tools are able to scale both horizontally and vertically in an almost transparent way for the upper layers, thanks to virtualisation, containers and load balancers. In fact, auto-scaling is even offered as a commercial service (as discussed in section 10).

Nowadays the auto-scaling is decoupled from resource scheduling. Auto-scaling is offered as a service to the application, and the requested resources are independently scheduled on the lower physical infrastructural layer, aiming at optimising resource consumption, especially for energy-aware considerations. In practice, the auto-scaling problem moves from the infrastructure level up to the application level: scaling should be done taking into account the single application behaviour, adapting to the workload it is subject to, not considering the load of many-task applications.

At the state of art, infrastructures offer the capability to scale, while it is up to the application to make a decision on when, how and how much to scale. As a consequence, the research topics change and can be summarised as follows:

1) auto-scaling policy definition and design,
2) comparison and benchmarking of auto-scaling tools,
3) definition of auto-scaling metrics and performance figures,
4) trade-off among scaling and costs, and finally
5) protection against Economic Denial of Sustainability or Fraudulent cloud Resource Consumption attacks.

These topics are dealt with below:

Auto-scaling policy definition and design: The first and clear open research point is the need for techniques and languages to design and express scalability policies in a way that is as much as possible vendor- and technology-independent, being at the same time able to catch the application behaviour and to define the criteria and logic needed to scale.

Comparison and benchmarking auto-scaling tools: even if the technologies are nowadays commonly available, at the best of author's knowledge there is not a commonly accepted reference architecture for auto-scaling tools, and no stable benchmarks that enable a comparison among the different techniques and solutions on the market. This is clearly an open issue: being the auto-scaling delegated to the application, together with the associated costs, it is relevant to know how well the auto-scaling tool will react and implement the requested policy. Performance indicators should be correctly identified in order to evaluate well and to implement the scaling policies.

Definition of auto-scaling metrics: the need for benchmarks opens up a research area even for new metrics and indicators, able to express in a clear way the overhead introduced, how it affects the scaling policy and, overall, the scaling quality. In this field, one of the most innovative aspects is to devise metrics that take into account the relationship with costs, as turns out from a pay-per-use resource usage. While at infrastructure level the resources are almost fixed and the costs are mainly considered to optimise their use and/or to reduce energy dissipation (obtained reducing the amount of resources adopted), moving the scaling problem at application level implies a direct effect of scaling on costs: adding/removing a VM or changing the type of machine directly affects the costs, and so it is possible to define policies taking into account the cost as a performance indicator (as illustrated in Section 9).

Trade-off among scaling and costs: in light of the above, auto-scaling policies should accordingly take into account the trade-off between performance-related metrics and cost-related metrics.

Protection against Economic Denial of Sustainability or Fraudulent cloud Resource Consumption attacks: the cloud pay-per-use paradigm has an additional side-effect: a new type of cyber-attacks named Economic Denial of Sustainability (EDoS) or Fraudulent cloud Resource Consumption (FRC), which aim at forcing the cloud applications to consume more resources than usual. The objective of these attacks is not to make service unavailable as in a DoS attack, but to increase the application costs to be payed to the hosting cloud provider. Mitigation techniques for EDoS and FRC exist – interesting surveys on the topic are Somasundaram (2016); Thaper, R. and Verma (2015); Singh et al. (2014) and VivinSandar and Shenai (2012), but they require suitable application design and need to be taken into account in the auto-scaling policy design and/or tools implementation. At the time of writing, no auto-scaling implementation deals with the problem of EDoS and FRCs.

12 Conclusions

To be exploited at best, the great opportunity offered by the elasticity of cloud environments calls for clever auto-scaling techniques and tools. Our snapshot of the state of the art shows clearly that auto-scaling is not only a technological issue, as its adoption opens up to a lot of new problems and opportunities to be addressed from research point of view.

In this paper we have provided a reasoned and commented view of the key factors to be considered in the design of an auto-scaler. We hope that this work could be useful not only for understanding the pros and cons of existing solutions, but also for the development of a new generation of auto-scalers. In fact, the analysis of the literature on auto-scaling and the discussion on commercial solutions shows all the limits of present-day solutions, and the necessity to move in the directions pointed out in our suggestions for further research.

References

Al-Dhuraibi, Y., Zalila, F., Djarallah, N.B. and Merle, P. (2018) 'Coordinating vertical elasticity of both containers and virtual machines', *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, pp.1–9.

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

Alipour, H. and Liu, Y. (2017) 'Online machine learning for cloud resource provisioning of microservice backend systems', *Proceedings of the IEEE International Conference on Big Data (Big Data)*, pp.2433–2441.

Alzahrani, E.J., Tari, Z., Zeephongsekul, P., Lee, Y.C., Alsadie, D. and Zomaya, A.Y. (2016) 'SLA-aware resource scaling for energy efficiency', *Proceedings of the IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp.852–859.

Amiri, M. and Mohammad-Khanli, L. (2017) 'Survey on prediction models of applications for resources provisioning in cloud', *Journal of Network and Computer Applications*, Vol. 82, No. C, pp.93–11.

Assuncao, M., Cardonha, C., Netto, M. and Cunha, R.. (2016) 'Impact of user patience on auto-scaling resource capacity for cloud services', *Future Generation Computer Systems*, Vol. 55, pp.41–50.

AWS Auto Scaling (n.d.) *Application scaling to optimize performance and costs*. Available online at: https://aws.amazon.com/autoscaling.

Beltrán, M. (2015) 'Automatic provisioning of multi-tier applications in cloud computing environments', *Journal of Supercomputing*, Vol. 71, No. 6, pp.2221–2250.

Biswas, A., Majumdar, S., Nandy, B. and El-Haraki, A. (2015) 'An auto-scaling framework for controlling enterprise resources on clouds', *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp.971–980.

Bunch, C., Arora, V., Chohan, N., Krintz, C., Hegde, S. and Srivastava, A. (2012) 'A pluggable autoscaling service for open cloud paas systems', *Proceedings of the IEEE 5th International Conference on Utility and Cloud Computing*, pp.191–194.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F. and Buyya, R. (2011) 'Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms', *Software: Practice and Experience*, Vol. 41, No. 1, pp.23–50.

Casalicchio, E. and Silvestri, L. (2013) 'Autonomic management of cloud-based systems: the service provider perspective', in Gelenbe, E. and Lent, R. (Eds): *Computer and Information Sciences III*, Springer, London, pp.39–47.

Catillo, M., Ocone, L., Rak, M. and Villano, U. (2021) 'Black-box load testing to support auto scaling web applications in the cloud', *International Journal of Grid and Utility Computing*, Vol. 12, No. 2, pp.139–148.

Chieu, T.C., Mohindra, A., Karve, A.A. and Segal, A. (2009) 'Dynamic scaling of web applications in a virtualized cloud computing environment', *Proceedings of the IEEE International Conference on e-Business Engineering*, pp.281–286.

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

Dutta, S., Gera, S., Verma, A. and Viswanathan, B. (2012) 'Smartscale: automatic application scaling in enterprise clouds', *Proceedings of the IEEE 5th International Conference on Cloud Computing*, pp.221–228.

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

Garcia, A., Blanquer, I. and Garcia, V.H. (2014) 'SLA-driven dynamic cloud resource management', *Future Generation Computer Systems*, Vol. 31, pp.1–11.

Garí, Y., Monge, D.A., Pacini, E., Mateos, C. and García Garino, C. (2020) 'Reinforcement learning-based autoscaling of workflows in the cloud: a survey', *arXiv:2001.09957v1 [cs.DC]*, pp.1–41.

Ghanbari, H., Simmons, B., Litoiu, M. and Iszlai, G. (2011) 'Exploring alternative approaches to implement an elasticity policy', *Proceedings of the IEEE 4th International Conference on Cloud Computing*, pp.716–723.

Gujarati, A., Elnikety, S., He, Y., McKinley, K.S. and Brandenburg, B.B. (2017) 'Swayam: distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency', *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, ACM, New York, NY, USA, pp.109–120.

Han, R., Ghanem, M.M., Guo, L., Guo, Y. and Osmond, M. (2014) 'Enabling cost-aware and adaptive elasticity of multi-tier cloud applications', *Future Generation Computer Systems*, Vol. 32, No. C, pp.82–98.

Hasan, M.Z., Magana, E., Clemm, A., Tucker, L. and Gudreddi, S.L.D. (2012) 'Integrated and autonomic cloud resource scaling', *Proceedings of the IEEE Network Operations and Management Symposium*, pp.1327–1334.

Herbst, N.R., Huber, N., Kounev, S. and Amrehn, E. (2013) 'Self-adaptive workload classification and forecasting for proactive resource provisioning', *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ACM, New York, NY, USA, pp.187–198.

Horn, G. and Skrzypek, P. (2018) 'MELODIC: utility based cross cloud deployment optimisation', *Proceedings of the 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, IEEE, pp.360–367.

Horovitz, S. and Arian, Y. (2018) 'Efficient cloud auto-scaling with sla objective using q-learning', *Proceedings of the IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp.85–92.

Hu, Y., Deng, B. and Peng, F. (2016) 'Autoscaling prediction models for cloud resource provisioning', *Proceedings of the 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp.1364–1369.

Huang, J., Li, C. and Yu, J. (2012) 'Resource prediction based on double exponential smoothing in cloud computing', *Proceedings of the 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pp.2056–2060.

Ilyushkin, A., Ali-Eldin, A., Herbst, N., Papadopoulos, A.V., Ghit, B., Dick, E. and Iosup, A. (2017) 'An experimental performance evaluation of autoscaling policies for complex workflows', *Proceedings of the Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ACM, New York, NY, USA, pp.75–86.

Incerto, E., Tribastone, M. and Trubiani, C. (2018) 'Combined vertical and horizontal autoscaling through model predictive control', in Aldinucci, M., Padovani, L. and Torquati, M. (Eds): *Euro-Par 2018: Parallel Processing*, Springer International Publishing, Cham, pp.147–159.

Islam, S., Keung, J., Lee, K. and Liu, A. (2012) 'Empirical prediction models for adaptive resource provisioning in the cloud', *Future Generation Computer Systems*, Vol. 28, No. 1, pp.155–162.

John, I., Sreekantan, A. and Bhatnagar, S. (2019) 'Auto-scaling resources for cloud applications using reinforcement learning', *Grace Hopper Celebration India (GHCI)*, pp.1–5.

Klinaku, F., Frank, M. and Becker, S. (2018) 'Caus: An elasticity controller for a containerized microservice', *Proceedings of the Companion of the ACM/SPEC International Conference on Performance Engineering*, ACM, New York, NY, USA, pp.93–98.

Kriushanth, M. and Arockiam, L. (2014) 'Article: cost aware dynamic rule based auto-scaling of infrastructure as a service in cloud environment', *Proceedings on International Conference on Advanced Computing and Communication Techniques for High Performance Applications*, pp.35–40.

Kumar, D. and Gondhi, N.K. (2017) 'A QoS-based reactive auto scaler for cloud environment', *Proceedings of the International Conference on Next Generation Computing and Information Systems (ICNGCIS)*, pp.19–23.

Lesch, V., Bauer, A., Herbst, N. and Kounev, S. (2018) 'Fox: cost-awareness for autonomic resource management in public clouds', *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ACM, New York, NY, USA, pp.4–15.

Li, C., Tang, J. and Luo, Y. (2020) 'Elastic edge cloud resource management based on horizontal and vertical scaling', *The Journal of Supercomputing*, Vol. 76, pp.7707–7732.

Li, T., Wang, J., Li, W., Xu, T. and Qi, Q. (2016) 'Load prediction-based automatic scaling cloud computing', *Proceedings of the International Conference on Networking and Network Applications (NaNA)*, pp.330–335.

Liao, W., Kuai, S. and Leau, Y. (2015) 'Auto-scaling strategy for amazon web services in cloud computing', *Proceedings of the IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pp.1059–1064.

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

Liu, X., Yuan, S., Luo, G. and Huang, H. (2016) 'Auto-scaling mechanism for cloud resource management based on client-side turnaround time', in Thi Zin, T., Chun-Wei Lin, J., Pan, J-S., Tin, P. and Yokota, M. (Eds): *Genetic and Evolutionary Computing*, Springer International Publishing, Cham, pp.209–219.

Liu, X., Yuan, S., Luo, G., Huang, H. and Bellavista, P. (2017) 'Cloud resource management with turnaround time driven auto-scaling', *IEEE Access*, Vol. 5, pp.9831–9841.

Lorido-Botran, T., Miguel-Alonso, J. and Lozano, J. (2014) 'A review of auto-scaling techniques for elastic applications in cloud environments', *Journal of Grid Computing*, Vol. 12, pp.559–592.

Lu, L., Yu, J., Zhu, Y., Xue, G., Qian, S. and Li, M. (2017) 'Cost-efficient VM configuration algorithm in the cloud using mix scaling strategy', *Proceedings of the IEEE International Conference on Communications (ICC)*, pp.1–6.

Lu, L., Yu, J., Zhu, Y., Xue, G., Qian, S. and Li, M. (2017) 'Cost-efficient VM configuration algorithm in the cloud using mix scaling strategy', *Proceedings of the IEEE International Conference on Communications (ICC)*, pp.1–6.

Mao, M., Li, J. and Humphrey, M. (2010) 'Cloud auto-scaling with deadline and budget constraints', *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, pp.41–48.

Maurer, M., Breskovic, I., Emeakaroha, V.C. and Brandic, I. (2011) 'Revealing the MAPE loop for the autonomic management of cloud infrastructures', *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, IEEE, pp.147–152.

Mell, P. and Grance, T. (2011) 'The NIST definition of cloud computing', *NIST Special Publication*, Vol. 800, p.145.

Merkel, D. (2014) 'Docker: lightweight linux containers for consistent development and deployment', *Linux Journal*.

Moldovan, D., Truong, H. and Dustdar, S. (2016) 'Cost-aware scalability of applications in public clouds', *IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, Berlin, Germany.

Moltó, G., Caballer, M. and De Alfonso, C. (2016) 'Automatic memory-based vertical elasticity and oversubscription on cloud platforms', *Future Generation Computer Systems*, Vol. 56, No. C, pp.1–10.

Moore, L.R. Bean, K. and Ellahi, T. (2013a) 'Transforming reactive auto-scaling into proactive auto-scaling', *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*, ACM, New York, NY, USA, pp.7–12.

Moore, L.R., Bean, K. and Ellahi, T. (2013b) 'A coordinated reactive and predictive approach to cloud elasticity', *Proceedings of the 4th International Conference on Cloud Computing, GRIDs, and Virtualization*, pp.87–92..

Novak, J.H., Kasera, S.K. and Stutsman, R. (2019) 'Cloud functions for fast and robust resource auto-scaling', *Proceedings of the 11th International Conference on Communication Systems Networks (COMSNETS)*, pp.133–140.

Qavami, H.R., Jamali, S., Akbari, M.K. and Javadi, B. (2014) 'Dynamic resource provisioning in cloud computing: a heuristic markovian approach', in Leung, V.C.M. and Chen, M. (Eds): *Cloud Computing*, Springer International Publishing, Cham, pp.102–111.

Qu, C., Calheiros, R.N. and Buyya, R. (2018) 'Auto-scaling web applications in clouds: a taxonomy and survey', *ACM Computing Surveys*, Vol. 51, No. 4, pp.1–33.

Roy, N., Dubey, A. and Gokhale, A. (2011) 'Efficient autoscaling in the cloud using predictive models for workload forecasting', *Proceedings of the IEEE 4th International Conference on Cloud Computing*, pp.500–507.

Sedaghat, M., Hernandez-Rodriguez, F. and Elmroth, E. (2013) 'A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling', *Proceedings of the 3 ACM Cloud and Autonomic Computing Conference*, ACM, New York, NY, USA.

Shariffdeen, R.S., Munasinghe, D.T.S.P., Bhathiya, H.S., Bandara, U.K.J.U. and Bandara, H.M.N.D. (2016) 'Adaptive workload prediction for proactive auto scaling in PaaS systems', *Proceedings of the 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pp.22–29.

Singh, P., Gupta, P., Jyoti, K. and Nayyar, A. (2019) 'Research on auto-scaling of web applications in cloud: Survey, trends and future directions', *Scalable Computing: Practice and Experience*, Vol. 20, No. 2, pp.399–432.

Singh, P., Manickam, S. and Rehman, S.U. (2014) 'A survey of mitigation techniques against economic denial of sustainability (EDos) attack on cloud computing architecture, pp.1–4.

Somasundaram, A. (2016) 'Economic denial of sustainability attack on cloud – a survey', *ICTACT Journal on Communication Technology*, Vol. 7, No. 4, pp.1402–1407.

Souza, A.A.D.P. and Netto, M.A.S. (2015) 'Using application data for SLA-aware auto-scaling in cloud environments', *Proceedings of the IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp.252–255.

Sutton, R.S. and Barto, A.G. (2018) *Reinforcement Learning: An Introduction*, 2nd ed., The MIT Press.

Taherizadeh, S., Stankovski, V. and Cho, J. (2019) 'Dynamic multi-level auto-scaling rules for containerized applications', *The Computer Journal*, Vol. 62, No. 2, pp.174–197.

Thaper, R. and Verma, A. (2015) 'A survey on economic denial of sustainability attack mitigation techniques', *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 3, No. 3, pp.2312–2316.

Tighe, M. and Bauer, M. (2017) 'Topology and application aware dynamic VM management in the cloud', *Journal of Grid Computing*, Vol. 15, No. 2, pp.273–294.

Tran, D., Tran, N., Nguyen, G. and Nguyen, B.M. (2017) 'A proactive cloud scaling model based on fuzzy time series and SLA awareness', *Procedia Computer Science*, Vol. 108, pp.365–374.

Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P. and Wood, T. (2008) 'Agile dynamic provisioning of multi-tier internet applications', *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 3, No. 1, pp.1–39.

VivinSandar, S. and Shenai, S. (2012) 'Economic denial of sustainability (EDoS) in cloud services using HTTP and XML based DDoS attacks', *International Journal of Computer Applications*, Vol. 41, No. 20, pp.11–16.

Yaqub, E., Yahyapour, R., Wieder, P., Jehangiri, A.I., Lu, K. and Kotsokalis, C. (2014) 'Metaheuristics-based planning and optimization for SLA-aware resource management in PaaS clouds', *Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp.288–297.

Ye, T., Guangtao, X., Shiyou, Q. and Minglu, L. (2017) 'An auto-scaling framework for containerized elastic applications', *Proceedings of the 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pp.422–430.

Ye, T., Guangtao, X., Shiyou, Q. and Minglu, L. (2017) 'An auto-scaling framework for containerized elastic applications', *Proceedings of the 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pp.422–430.

Zhang, Q., Cherkasova, L. and Smirni, E. (2007) 'A regression-based analytic model for dynamic resource provisioning of multi-tier applications', *Proceedings of the 4th International Conference on Autonomic Computing*, pp.27–27.