Engineering Applications of Artificial Intelligence 102 (2021) 104288

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Survey paper

Reinforcement learning-based application Autoscaling in the Cloud: A survey

Yisel Garí [a,c,*], David A. Monge [a,*], Elina Pacini [a,b,c], Cristian Mateos [d,c], Carlos García Garino [a,b]

[a] ITIC - Universidad Nacional de Cuyo. Mendoza, Argentina
[b] Facultad de Ingeniería - Universidad Nacional de Cuyo. Mendoza, Argentina
[c] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
[d] ISISTAN-UNICEN-CONICET. Tandil, Buenos Aires, Argentina

ARTICLE INFO

Keywords:
Cloud Computing
Cloud application
Autoscaling
Reinforcement learning

ABSTRACT

Reinforcement Learning (RL) has demonstrated a great potential for automatically solving decision-making problems in complex, uncertain environments. RL proposes a computational approach that allows learning through interaction in an environment with stochastic behavior, where agents take actions to maximize some cumulative short-term and long-term rewards. Some of the most impressive results have been shown in Game Theory where agents exhibited superhuman performance in games like Go or Starcraft 2, which led to its gradual adoption in many other domains, including Cloud Computing. Therefore, RL appears as a promising approach for Autoscaling in Cloud since it is possible to learn transparent (with no human intervention), dynamic (no static plans), and adaptable (constantly updated) resource management policies to execute applications. These are three important distinctive aspects to consider in comparison with other widely used autoscaling policies that are defined in an ad-hoc way or statically computed as in solutions based on meta-heuristics. Autoscaling exploits the Cloud elasticity to optimize the execution of applications according to given optimization criteria, which demands deciding when and how to scale up/down computational resources and how to assign them to the upcoming processing workload. Such actions have to be taken considering that the Cloud is a dynamic and uncertain environment. Motivated by this, many works apply RL to the autoscaling problem in the Cloud. In this work, we exhaustively survey those proposals from major venues, and uniformly compare them based on a set of proposed taxonomies. We also discuss open problems and prospective research in the area.

1. Introduction

Cloud computing (Mauch et al., 2013) brings a technological solution for the execution of different types of applications due to its reliability, availability, and resource scalability. Particularly, the IaaS Cloud service model allows users to create and destroy different types of Virtual Machine (VM) instances, optionally under a pay-per-use scheme. In this way, it is possible to dynamically adjust the infrastructure according to the variations in resource demands during the execution of applications. This feature of Clouds and the need to achieve efficient execution of applications encourage the study and development of autoscaling strategies in the Cloud (Monge et al., 2017; Garí et al., 2019). These strategies are aimed to optimize the application execution based on different objectives such as execution time and economic cost, as well as compliance with restrictions or the Service Level Agreements (SLA), if specified. An SLA represents an agreement between a service provider and its users to define quality aspects of the service offered by the provider based on user requirements (Kearney and Torelli, 2011).

Autoscaling strategies periodically solve two interrelated optimization problems, i.e. scaling and scheduling. The scaling stage consists of adjusting the number and type of Cloud resources acquired (e.g., VMs) according to the application demand. On the other hand, the scheduling stage consists of assigning each application task to the acquired resources. Both subproblems are NP-hard, so they are usually approached with heuristics. Considering that the variability in Cloud performance represents an important factor of uncertainty in the execution of applications, several recent investigations propose solutions based on Reinforcement Learning (RL) to solve some of the involved subproblems, the scaling stage (Dutreilh and Kirgizov, 2011; Barrett et al., 2012; Veni and Bhanu, 2016; Arabnejad et al., 2017; Ghobaei-Arani et al., 2018; Dezhabad and Sharifian, 2018; Bibal Benifa and Dejey, 2018; Garí et al., 2019) or the scheduling stage (Barrett et al., 2011; Peng et al., 2015; Xiao et al., 2017; Duggan et al., 2017; Liu et al., 2017; Soualhia et al., 2018; Mingxi Cheng and Nazarian, 2018).

RL is one of the three basic machine learning paradigms together with supervised learning and unsupervised learning. Specifically, RL

* Corresponding author at: ITIC - Universidad Nacional de Cuyo. Mendoza, Argentina.
E-mail addresses: ygari@uncu.edu.ar (Y. Garí), dmonge@uncu.edu.ar (D.A. Monge), epacini@uncu.edu.ar (E. Pacini), cristian.mateos@isistan.unicen.edu.ar (C. Mateos), cgarcia@itu.uncu.edu.ar (C. García Garino).

https://doi.org/10.1016/j.engappai.2021.104288
Received 12 May 2020; Received in revised form 30 April 2021; Accepted 1 May 2021
Available online 11 May 2021
0952-1976/

proposes a computational approach that allows an agent to learn the appropriate behavior to achieve its objective by interacting with a stochastic environment (Sutton and Barto, 2018). The agent periodically takes an action that modifies the state of the environment and observes a reward signal that allows the agent to evaluate the immediate effect of the action taken. Actions also have long-term consequences that are not immediately perceptible. Therefore, RL purpose is to let the agent learn appropriate policies – i.e., mapping the states to actions – to generate the greatest long-term benefit following the agent objective. RL-based strategies are being widely used, and very encouraging results have been obtained in areas such as Game Theory (Silver et al., 2016; Mnih et al., 2015), which has motivated its study and application in other areas, concretely autoscaling in Clouds.

It is also important to highlight that the main motivation for addressing the autoscaling of applications in Clouds from the perspective of RL are the following:

1. Policies are *transparent*, i.e., they are not dependent on human intervention or deep domain knowledge, since the scaling and scheduling policies are learned through interaction with the environment;
2. Policies are *dynamic*, i.e., a learned policy determines an adequate action based on the current state of the environment and the application execution, instead of a static plan previously computed as in solutions based on meta-heuristics; and
3. Policies are *adaptable*, i.e., *online* policy learning facilitates policy improvement and constant updates. Thus, learned policies can adapt to the changes that occur in the dynamics of the Cloud environment, unlike policies learned in *offline* mode (Barrett et al., 2011; Garí et al., 2019) that are prone to become obsolete in time.

Therefore, in a Cloud setting – i.e., an environment with uncertainty – where, for example, the variability in the performance of VMs constantly changes, it is necessary to make appropriate decisions on the fly. The use of RL enables feasible decision-making regarding the type and number of VM to use (scaling) as well as which resources should be assigned at any moment (scheduling). Motivated by this, several recent researches propose RL-based approaches to solve the Cloud autoscaling problem.

Last years have witnessed an astonishing amount of research in Cloud resource management from both a theoretical and a practical perspective. As a consequence, several works have been proposed and many of those have been summarized in a number of surveys. On the one hand, we can mention those surveys that point to approaches based on intelligent techniques for efficient resource management (scheduling, load balancing, throughput, energy consumption, etc.), but they do not emphasize the elasticity property of Clouds. For example, Kumar et al. (2019) have discussed fundamental concepts and advantages of the existing resource provisioning techniques; they have categorized scheduling algorithms in terms of: static and dynamic, offline and online operation mode, and preemptive/non-preemptive scheduling. Moreover, several scheduling algorithms based upon heuristics, meta-heuristics and hybrid techniques are reviewed. Then, the survey by Singh and Chana (2016) conduct a broad literature analysis of Cloud resource provisioning. The survey is mainly focused on nature-inspired and bio-inspired techniques. Finally, Arunarani et al. (2018)classify algorithms as bio-inspired and Fuzzy-based, target applications, and utilized performance measures.

On the other hand, there are those surveys that focus on intelligent techniques for resource management and also emphasize the elasticity property of Cloud. For example, in Oliveira de Carvalho et al. (2018), the authors present and discuss works based on evolutionary computation facing the problem of resource management in multiple Clouds from the user's perspective. Particularly, the surveyed works exploit Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Simulated Annealing (SA) and hybrid heuristic approaches. The survey is not simply focused on the employed evolutionary computation techniques, but also in the definition and classification of multiple Cloud resources. Note that our work is located within this second category of surveys, but having in mind RL as the intelligent technique.

Hence, we have conducted a literature review of relevant works that address the Cloud autoscaling problem via solutions based on RL, being to the best of our knowledge, the first survey on this topic. In particular, we have surveyed Cloud autoscaling approaches for three types of applications, namely workflows, independent tasks, and Cloud services. We have classified the surveyed works based on a taxonomy according to the type of RL-based technique used. From the analysis of the state of the art in the application of RL strategies for autoscaling in Cloud, noticeable findings are that none of the works jointly solves the scaling and scheduling subproblems. Besides, although workflow is a mature technology driving many of the Cloud applications nowadays, very few works have been proposed for workflows, and those that do consider workflows only focus on scheduling without taking into account the scaling problem. These facts altogether evidence not only the promissory nature of RL-based application autoscaling in Clouds but also the fertile characteristic of the area in terms of prospective future improvements.

This article is organized as follows. In Section 2, the background is presented, explaining underpinning concepts such as Cloud application types, the Cloud Computing paradigm and related provisioning models, the autoscaling problem, and the RL basics that will be referred throughout this work. Section 3 discusses the relevant works in the area in detail. Then, in Section 4 the limitations of the surveyed works and open problems are highlighted. Section 5 concludes the survey. Finally, in Appendices A.1 through A.4 we overview different relevant techniques under the umbrella of RL that are exploited by the surveyed approaches.

2. Background

In this section, the theoretical and technical foundations underpinning the paper are discussed, namely those related to Cloud applications, Clouds as an execution environment, the concept of autoscaling, and the RL basics. For this, the concepts and fundamental characteristics of Cloud applications are presented in Section 2.1. Then, the Cloud Computing paradigm and its different service models are analyzed in Section 2.2. Later, in Section 2.3 we describe the Cloud autoscaling problem by integrating the two previous topics. Finally, we introduce the RL basics.

2.1. Cloud applications

Applications in the Cloud can fall into one of three categories:

1. *Workflows*. A workflow describes a complex objective through the composition of a set of tasks, usually readily-available via software components, through dependencies. This approach allows people without experience in programming languages but with solid knowledge of the problem domain to contribute to the development of new applications. Workflows have been and are widely used in the modeling of complex research experiments in several disciplines such as Geoscience (Liu et al., 2016), Astronomy (Brown et al., 2007; Meade and Fluke, 2018), and Bioinformatics (Vandenbrouck et al., 2019), among many others. Hence, workflows built in this context are termed *scientific workflows*.
2. *Independent tasks* (a.k.a. *bag of tasks*). Applications of this type are constructed as a set of tasks logically related but without dependencies among them. Examples of these applications are different runs of a Monte Carlo simulation (Wang et al., 2011; Hu et al., 2019), parameter sweep experiments (Monge et al., 2018), and any type of embarrassingly parallel jobs. Note that independent-task applications can be considered a particular

case of workflow applications, where all tasks depend on a single fictitious start task and precede a single fictitious end task. And, just like in the case of workflows, tasks are in general data-intensive (Fortino et al., 2014a) and/or CPU-intensive and hence require a large amount of computational hardware and software resources that include computing power, high-speed networks, storage capacity, and sophisticated administration tools, among others.

3. *Cloud services*. Applications of this type are the most dissimilar to the previous ones. In general, they serve multiple *user requests* at a time. User requests (from now on tasks, to homogenize nomenclature with the other two types of applications) are independent of each other and arrive unpredictably. Examples are Facebook, Twitter, or any Cloud-hosted Web application. Note that contrary to the previous two kinds of applications, the number of tasks here can dramatically vary over time and usually have a shorter duration.

A detailed explanation regarding the particularities of each application type and its implications is given in Section 3.4.2. However, we highlight that although every type of application has its own particularities, they share some characteristics. First, in all cases, there is a considerable number of tasks that can be executed in parallel. Second, workloads can heavily vary over time (e.g. either because of the workflow structure itself or just because of the number of requests changes). Such variability determines instants of time where an infrastructure with greater or lesser capacity is required. The provisioning and elasticity capabilities of the Cloud computing paradigm make it an excellent candidate to meet the varying computational requirements of these applications.

2.2. Cloud computing

The National Institute of Standards and Technology (NIST) defines Cloud Computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell et al., 2011)".

The tendency to expose Everything as a Service (XaaS) when it comes to Cloud capabilities describes a widely adopted scenario in which service-oriented architecture and design principles underpin the development and implementation of software services in Clouds (Fortino et al., 2014b; Duan et al., 2015; Gravina et al., 2017). In this sense, NIST defines the three following base service models in the Cloud:

- Infrastructure as a Service (IaaS), where "service" means resource: An IaaS Cloud enables on-demand provisioning of computational resources in the form of VM deployed in a datacenter, minimizing or even eliminating associated capital costs for users, and letting those users adding or removing capacity from their IT infrastructure to meet peak or fluctuating resource demands. Examples of IaaS providers include Amazon EC2,[1] Windows Azure Services Platform,[2] and Google Compute Engine.[3]
- Platform as a Service (PaaS), where "service" means platform-level functionality: The user can create their own software using tools and/or libraries from the provider, including operating systems, programming languages, databases, and Web servers. Some examples are Google App Engine and Windows Azure Cloud Services.
- Software as a Service (SaaS), where "service" means application: Providers install and operate application-level software in the Cloud, which is transparently accessed by users through the browser, a mobile application, or a Web Service API. Examples of SaaS are Google Apps, Microsoft Office 365, and Dropbox.
- Function as a Service (FaaS), where "service" is referred to as a "runtime software component in a serverless architecture": It is based on lightweight functions that can be triggered by a given event. Building an application following this model means writing functions without pondering about concerns such as deployment, server resources, scalability, etc. Examples are AWS Lambda, Google Cloud Functions, Microsoft Azure Functions, and Webtask.io.

This work is placed in the context of the service model IaaS offered by public Cloud providers because the surveyed works mainly focus on how the virtual infrastructure is scaled when running resource-intensive Cloud applications.

One of the main features of the IaaS model is the elasticity at the infrastructure level, which allows users to dynamically acquire and adjust the computing infrastructure according to their needs. Elasticity is supported from a technical perspective through the use of virtualization technologies (Buyya et al., 2009). Virtualization technologies allow us to share the resources of a single physical machine (PM) among several independent VM instances. Several VMs might co-exist in the same PM and have no visibility or control over the configuration of the PM that hosts them or over the neighboring VMs. Each VM has assigned a portion of the physical resources available in the PM (CPU, RAM, storage, and network bandwidth). Moreover, a VM monitor isolates individual VMs from its environment for security reasons and possible failures, but not to improve performance (Koh et al., 2007; Pu et al., 2010). Consequently, the unpredictable performance of the VMs (Koh et al., 2007; Pu et al., 2010) becomes one of the main obstacles facing the Cloud computing model (Armbrust et al., 2009).

Another key feature of the IaaS model is the eradication of costs associated with the maintenance of the infrastructure since users only pay for the resources/VMs they use. Typically, prices differ according to the computing capabilities of available VMs, but prices may also differ depending on the *price model* under which a VM is rented. Two of the most common pricing models are:

On-demand/Non-preemptible instances. This price model is suitable for users with sporadic and bursting demands since the on-demand option allows users to rent resources right away without a fixed time limit, as opposed to *reserved* instances, where the user rents resources for a fixed time duration while obtaining important price discounts. However, on-demand instances generally have higher prices than reserved instances, considering the same computing capabilities for acquired VMs.

Spot/Preemptible instances. Cloud providers have unsold computing capacity during certain periods. To encourage users to buy additional capacity, they offer *spot instances*. The prices of spot instances fluctuate over time, but they are usually much lower (up to 90% in some cases) than the prices of on-demand instances, considering the same computing capabilities. Then, the user makes an offer with the maximum value that he/she is willing to pay for the instance and will be charged only with the (much cheaper) spot price at any time. When the spot price exceeds the user's bid, the VM instances are terminated, interrupting any running task on them.

Some IaaS providers such as Amazon have recently decided to no longer rely on the fluctuation of VM prices based on bid schemes (Fabra et al., 2019). In the new model, the spot prices are more predictable, updated less frequently, and are determined by supply and demand for Amazon EC2 spare capacity, not bid prices. However, factors that condition the interruption of a spot instance are completely internal to the provider and cannot be known without a deep understanding and analysis of the AWS infrastructure at runtime. Then, it is important to

[1] Amazon Elastic Compute Cloud: https://aws.amazon.com/ec2.
[2] Windows Azure Services: https://azure.microsoft.com.
[3] Google Cloud Platform: https://cloud.google.com/.

consider, together with the economic advantages of spot instances, the fact that spot instances are less reliable since the tasks they execute are subject to sudden, unpredictable terminations.

For simplicity, from now on, we will refer to the different instance types according to the terminology used by Amazon (reserved, on-demand, and spot). These pricing models offer both a wide range of options to shape the infrastructure they need for the execution of their applications and the elasticity that enables the dynamic reconfiguration of such infrastructure.

2.3. Autoscaling in Clouds

The workload associated with Cloud applications presents a lot of variabilities over time. Such characteristics stress the necessity for resource elasticity. On top of that, these applications can be compute-intensive and/or require the processing of large volumes of data (e.g. workflows and independent tasks), or simply need to execute a massive number of lightweight tasks that, when aggregated, leading to high demand for resources (e.g. services). All these applications usually involve hundreds, thousands, or even a larger number of tasks with varying durations that can range from a few minutes to several days or weeks, optionally processing from MiB to TiB of input data. This adds a new complexity dimension as the duration of the tasks may further differ depending on the type of VM acquired to run them. Besides, since tasks usually have different computational requirements, some types of instances may be more suitable for certain application types. For example, Amazon offers general-purpose instances, which provide a balance of computing, memory, and networking resources, and are suitable for applications that use these resources in equal proportions such as web servers and code repositories. Moreover, compute-optimized instances are ideal for compute-bound applications that benefit from high-performance processors, for example, media trans-coding, high-performance computing (HPC), scientific modeling, dedicated gaming servers, and machine learning inference.

Even more, in public Clouds, the use of resources has an economic cost, and the different types of instances differ in their price. When having in mind maximizing the execution efficiency (e.g. reducing the execution cost, time, etc.), acquiring the appropriate infrastructure (i.e. the number of VMs of each type and their price models) becomes a complex problem.

All in all, autoscaling strategies have to deal with the dynamic scaling of the infrastructure according to the application needs and the online scheduling of such tasks on the running infrastructure. These are two interdependent problems that must be solved in tandem (Monge et al., 2017):

1. *Scaling*, which consists of dynamically adjusting the allocated virtualized resources. Scaling (Krzywda et al., 2018) can take two forms: horizontal scaling, when the number of assigned VMs of any type to an application is adjusted, and vertical scaling, when the capabilities of individual VMs (CPU, memory, I/O) are varied.
2. *Scheduling*, which consists of assigning tasks for execution in the acquired VM instances.

Both subproblems are NP-hard and, therefore, the solutions proposed to date are mainly based on heuristics (Mao and Humphrey, 2013; Monge et al., 2017, 2018, 2020).

In a Cloud infrastructure, the demand for resources fluctuates over time, and the VMs located in the same PM constantly compete for the resources they share. Also, the potential variations in network traffic impact the communication speed between physically separated VMs. All these elements make the performance of the Cloud infrastructure vary, which represents an important uncertainty factor in the decision-making processes behind scaling and scheduling since the real duration of tasks usually differs from the estimated values. Then, autoscaling strategies need to frequently update the information they have available for taking those decisions, considering that:

- applications present variable workload patterns at different stages,
- models that estimate task durations are imperfect, and
- as said, Cloud infrastructures are characterized by variable performance.

Then, autoscaling strategies need to monitor the state of the environment (infrastructure and applications) to mitigate the effects of discrepancies between the available information about tasks (estimations) and the real progress of the execution. Then, autoscaling strategies are periodically executed as shown in Fig. 1.

In each update interval, the autoscaling strategy addresses an optimization problem based on certain objectives of interest such as load balancing, throughput, flowtime, energy consumption, makespan, monetary cost, and so on. It is important to mention that both the makespan and cost, or a combination of both, are the optimization objectives most addressed by researchers in the area of Cloud scheduling (Pacini et al., 2014). Formally, given the set I^{type} of instance types (in terms of hardware and software capabilities) considered for autoscaling, the set I^{scheme} of price models (reserved, spot, and on-demand instances), the set T of tasks to process, and the set I of available instances in the current infrastructure, then, in each update interval the autoscaling strategies generate:

- $X^{\text{sca}} = \{I^{\text{type}} \times I^{\text{scheme}} \to \mathbb{N}_0\}$, a *scaling* plan that indicates the required number of reserved, on-demand, and spot instances of each type, and
- $X^{\text{plan}} = \{T \to I\}$, a set of *scheduling* decisions that map each task $t \in T$ to one of the $i \in I$ instances.

Furthermore, this optimization problem can be addressed from different perspectives: the execution of individual applications from different users, the execution of multiple applications from the same user, or many applications from different users. In all cases, it is important to consider that when exploiting non-dedicated IaaS Clouds, the performance can be affected by workloads external to the applications being executed themselves.

Thus, although most of the proposed solutions for the autoscaling problem are based on heuristics or meta-heuristics, recent researches aim to apply reinforcement-learning approaches to solve any of the subproblems involved, i.e. the scaling (Dutreilh and Kirgizov, 2011; Barrett et al., 2012; Veni and Bhanu, 2016; Arabnejad et al., 2017; Ghobaei-Arani et al., 2018; Dezhabad and Sharifian, 2018; Bibal Benifa and Dejey, 2018; Garí et al., 2019) or scheduling (Barrett et al., 2011; Peng et al., 2015; Xiao et al., 2017; Duggan et al., 2017; Liu et al., 2017; Soualhia et al., 2018; Mingxi Cheng and Nazarian, 2018).

2.4. Reinforcement learning

Reinforcement learning (RL) (Sutton and Barto, 2018) is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. RL is concerned with how a software agent ought to take actions in an environment with uncertainty to maximize some notion of cumulative reward. In the beginning, the agent does not know what actions to take, and as time passes, it must discover which actions produce the greatest long-term benefit. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Making the most appropriate decision requires taking into account the indirect consequences of the actions, therefore some kind of foresight or planning is necessary. These two characteristics, *trial-and-error search* and *delayed reward*, are the two distinguishing features of RL (Sutton and Barto, 2018).

Markov decision processes (MDP) (Bellman, 1957) provide a formal framework widely used to define the interaction between an RL agent and its environment in terms of states, actions, and rewards (see Fig. 2). MDPs have become the de facto standard formalism for learning

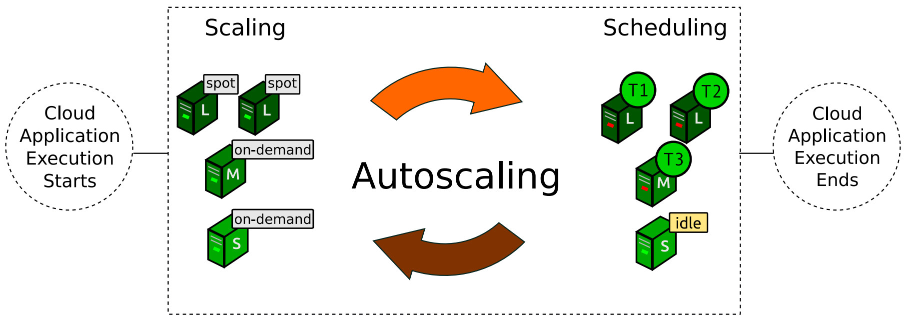

Fig. 1. Cyclic autoscaling process that includes the scaling and scheduling subprocesses. L, M and S represent different VM capabilities. T_i represent tasks to execute.

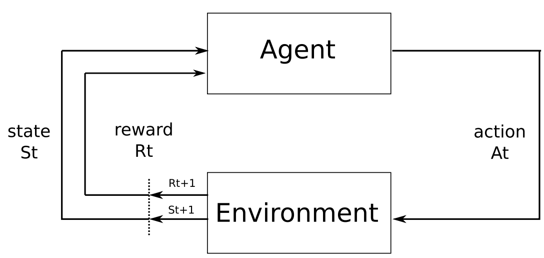

Fig. 2. MDP interaction process between an agent and the environment.
Source: Figure adapted from Sutton and Barto (2018).

sequential decision-making (Otterlo, 2009) and have been applied to autoscaling problems in Cloud (Tang et al., 2012; Barrett et al., 2011, 2012). A classical MDP is defined as a 5-tuple $(S, A, P_{\cdot}(\cdot,\cdot), R_{\cdot}(\cdot,\cdot), \gamma)$, where:

- S represents the environmental state space;
- A represents the whole action space;
- $P_a(s,s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ represents the probability that action a in state s at time t will lead to state s' at time $t+1$;
- $R_a(s,s')$ represents the (expected) immediate reward received after transitioning from state s to state s' due to action a;
- $\gamma \in [0,1]$ (or discount factor) is the difference in importance between future and immediate rewards. When γ is close to 0, rewards in the distant future are viewed as insignificant. When γ is 1 all rewards are equally important.

As shown in Fig. 2, the agent and the environment continuously interact in a constant exchange process. At each time t, the agent receives a representation of the environment state $s_t \in S$, and then selects an action $a_t \in A$ to execute. In the next step, the agent receives a numerical reward signal $r_{t+1} \in R \subset \mathbb{R}$ and goes to a new state s_{t+1}. The boundaries between the agent and the environment are determined by everything that the agent may or may not control, and not by those things the agent knows or does not know. For example, the agent can have knowledge of the current status or how the reward is computed, but only has control over the actions it takes. In general, the agent attempts to maximize the expected gain G_t that is defined as a specific function of the reward sequence. In the simplest case, the gain is the sum of the rewards: $G_t = r_{t+1} + r_{t+2} + \cdots + r_T$ where T is a final time step. Then, considering the discount factor γ, which determines the degree of importance of future rewards compared to the immediate rewards, the agent will attempt to select the action a_t so that the sum of the discounted rewards is maximized. In this case, the expected gain is calculated as: $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$.

Moreover, there are four fundamental elements in the RL learning process: the policy, the reward signal, the value function, and optionally, the model of the environment:

1. A *policy* defines how the learning agent behaves at a given time. A policy $\Pi : S \longrightarrow A$ is a mapping from perceived states of the environment to actions to be taken when in those states.
2. A *reward signal* evaluates the immediate effect of the taken actions considering the goal of the RL problem faced, and it is the primary basis for altering the policy.
3. A *value function* specifies what is good in the long term and it is fundamental for policy improvement. The state-value function $V(s)$ is the expected gain (i.e. cumulative reward over the future) to be obtained starting from this state. Value functions predict rewards into the future following a specific policy, and the purpose of estimating values is obtaining an improved policy to achieve more reward. Rewards are given directly by the environment, but values must be (re-)estimated from the sequences of observations an agent makes over time. In some cases, the state value function is not sufficient in suggesting a policy, and it is required to estimate the values related to each action. The action-value function $Q(s,a)$ represents the expected gain considering the state–action pair.
4. The *model of the environment* mimics the dynamic that determines how the environment behaves. Given a state and an action, the model might predict the next state and the next reward. Models are used for deciding on a course of action by considering possible future situations before they are experienced (in *offline* mode). However, an accurate model of the environment is not always available or the dynamic of the environment is prone to change over time. In these cases, learning is based on current situations being experienced (in *online* mode). Methods for solving RL problems that use models are called *model-based* methods, as opposed to simpler *model-free* methods that are explicitly trial and error learners.

When solving an MDP – i.e. obtaining an appropriate policy – two fundamental processes arise:

- The process of *predicting* the policy, where the values of the states or state–action pairs are estimated (i.e. the function $V(s)$ or $Q(s,a)$ is updated), generally based on the current estimated policy and the information of the environment (either from the model or from experience, according to the technique).
- The process of *control* or improvement of the estimated policy, where $\Pi(s)$ is computed based on the current estimated values.

Both processes determine a continuous interaction between the different approximations of the value function and the policy. This interaction in time converges to the optimal values for both functions (Sutton and Barto, 2018). Two commonly used methods to solve MDPs are Dynamic Programming methods (DP, see Appendix A.1) and Temporal Difference methods (TD, see Appendix A.2). In the context of Autoscaling in Cloud, some proposals also combine RL with Neural Networks (see Appendix A.3) and Fuzzy Logic (see Appendix A.4).

3. Review of Cloud autoscaling based on RL techniques

The two subproblems of Cloud autoscaling – i.e. scaling and scheduling – have been particularly addressed in the literature as decision-making problems in stochastic environments. The actions related to scaling might consist of e.g. increasing or reducing the number of VMs in the virtual infrastructure, while the actions related to scheduling consist of assigning each task to a specific acquired VM. Due to the uncertainty in these subproblems, proposals that model the autoscaling problem as an MDP have appeared, and essentially they use different RL techniques to learn adequate scaling (Dutreilh and Kirgizov, 2011; Barrett et al., 2012; Veni and Bhanu, 2016; Arabnejad et al., 2017; Ghobaei-Arani et al., 2018; Dezhabad and Sharifian, 2018; Bibal Benifa and Dejey, 2018; Garí et al., 2019) or scheduling (Barrett et al., 2011; Peng et al., 2015; Xiao et al., 2017; Duggan et al., 2017; Liu et al., 2017; Soualhia et al., 2018; Mingxi Cheng and Nazarian, 2018) policies. These policies allow an autoscaler to determine which action is more convenient at any time to optimize a long-term objective.

As we describe in detail in Appendices A.1 through A.4, there are different techniques for obtaining adequate policies. On the one hand, *Model-based* techniques require a perfect model of the environment to compute an appropriate policy in *offline* mode. The surveyed proposals which fall in the Model-based category use *Value Iteration* (see Appendix A.1), a well-known algorithm based on Dynamic Programming. On the other hand, there are the so-called *Model-free* techniques, which allow an agent to obtain a proper policy in *online* mode without requiring a perfect model of the environment. In other words, the policy is learned and improved over time via continuous interaction with the environment. The surveyed papers found in the Model-free category use *Q-learning* and *SARSA* (see Appendix A.2), two reference algorithms for Temporal Difference learning. As we mentioned earlier, RL techniques are usually affected by large state spaces, which directly impacts the performance of the aforementioned algorithms in terms of the time to compute a solution and memory usage. In this sense, the use of non-linear functions to approximate $Q(s, a)$ has been proposed, and solutions that combine RL with deep neural networks, i.e. Deep Reinforcement Learning (DRL) (see Appendix A.3) have appeared. Some proposals use Fuzzy Logic (FL) to represent rules capable of *fuzzily* encompassing multiple states in the context of RL, i.e. Fuzzy Reinforcement Learning (FRL) (see Appendix A.4).

In the next subsections, relevant works addressing the Cloud autoscaling problem via solutions based on RL are described and analyzed. The works are first organized according to the type of technique used as defined in the taxonomy depicted in Fig. 3. On the first level of the taxonomy, proposals in Model-based and Model-free categories are presented. Then, on a second level, proposals on the Model-free category are classified into three groups. First are those proposals that apply the technique in its original or pure formulation. These techniques are further subdivided into sequential or parallel since the variant of RL given by (multi-thread or multi-process) parallel learning is distinctive. Second, we present the proposals that combine RL with neural networks, and finally, the proposals that combine RL with FL.

To perform the literature selection process, we focused our search on the main citation databases (Google Scholar, Scopus, ACM Digital Library, IEEE Explore). We generated a corpus of papers whose abstract and title matched combinations of the following keywords: "reinforcement learning", "Markov decision process", "autoscaling", "scheduling", "scaling", "workflow", "scientific application", "parallel computing", "distributed computing". Articles were first filtered based on relevance considering the purpose of this survey. Then, a second selection process was carried out based on the quality of the journal/conference in which it was published. We kept articles published in journals in the first two quartiles (Q1 and Q2) of the SCImago Journal Rank, international conference papers (category A according to ERA ranking and categories A*/B* according to the Qualis ranking), and few papers not included in the aforementioned categories but very relevant to the scope of this work. The result was a corpus of 22 articles.

Furthermore, to better structure the discussion of the selected works, the following characteristics have been taken into account when describing each work:

- The type of problem that is solved (i.e., scaling, scheduling, or both if applicable),
- The targeted optimization objectives,
- The context of the problem (for example, migrations of VMs, scheduling of firewalls, etc.),
- The considered variables in the definition of the states and reward function,
- The RL-based algorithm used,
- Baselines used for comparisons and experimental results (if reported),
- Identified limitations.

Next, each of the works is classified by the type of RL technique implemented (Model-based approaches in Section 3.1 and Model-free approaches in Section 3.2), and then, in Section 3.4, three taxonomies based on the considered characteristics in the surveyed works are proposed.

3.1. Model-based approaches

There are only two proposals (Barrett et al., 2011; Garí et al., 2019) in the Model-based category. Both proposals have in common, on the one hand, the estimation of the probability distribution of the transition between states, due to the requirement of having a complete model of the environment in Model-based methods. On the other hand, both works share the limitation of learning the policies in an offline mode while operating in a dynamic environment (Cloud infrastructure).

Barrett et al. (2011) propose an approach for the efficient scheduling of workflows in the Cloud to minimize the makespan and monetary cost under deadline constraints. First, a genetic algorithm (GA) allows the approach to evolve different execution plans, where each workflow task is assigned to one of the available VMs. Then, through an MDP formulation and *Value Iteration,* a policy that dynamically chooses among the evolved plans the most suitable one for the moment is obtained. For the states definition, the authors consider three variables: a timestamps within the workload period (0–23 h), the resource load (light, moderate or heavy), and the execution result (workflow failed or executed successfully). The actions represent the selection of a specific scheduling plan. Then, the number of actions depends on the number of GA solvers. The reward is computed considering the cost of the workflow execution and a penalty if the schedule results in a violation of the specified deadline. In Barrett et al. (2011), since there is no perfect model of the environment, the probability distribution of the transitions between states is estimated from the information obtained from multiple previous workflow executions. In this sense, there is a limitation by which the quality of the obtained policy will depend directly on the quality of the estimate of $P(s, a)$.

Also, Garí et al. (2018, 2019) study the learning of budget allocation policies for the autoscaling of workflows in Clouds. In such works, through the outputs of multiple workflow executions, an MDP model is built, and then through the use of *Value Iteration* appropriate policies are derived. The derived policies are instead used by a workflow autoscaling strategy called SIAA (Monge et al., 2017) to determine in each autoscaling cycle, the adequate proportion of spot versus on-demand instances that must be maintained. For the states, the authors considered two features related to workload (i.e. the proportion of long duration tasks and the maximum parallel degree for the next period), the current budget limit, and the probability of out-of-bid error (i.e. failures of the acquired spot instances due to low bid). The values of each feature were discretized resulting in 192 possible states. The actions represent the possible budget assignment ratio between spot and on-demand instances (11 possible values). The reward is computed as the ratio between the progress and the cost of the workflow execution in

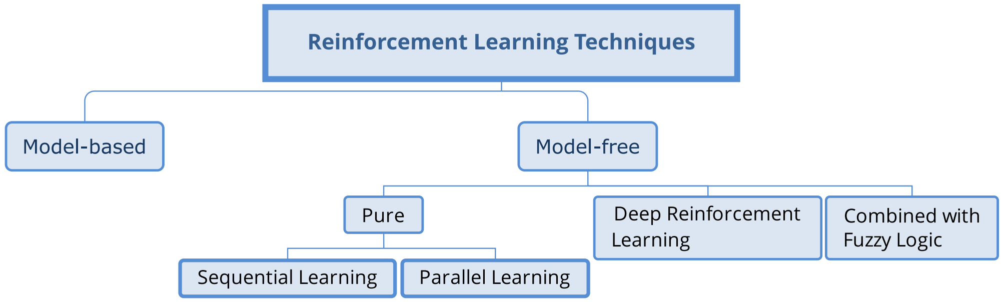

Fig. 3. Classification of RL based techniques applied to the Cloud autoscaling problem.

the last cycle. Both in Garí et al. (2019) and Barrett et al. (2011) there is a limitation given by the quality of the obtained policy depends on the quality of the estimate for $P(s,a)$. Besides, the fact that the policy is learned in an *offline* mode, gives the autoscaler partial ability to smoothly adapt to changes in the environment at runtime. For example, the prices of the spot instances and/or the probability of their failures could be subject to variations. Therefore, if the policy was learned in online mode, it could incorporate the experience of new executions and better adapt to changes accordingly.

3.2. Model-free approaches

Unlike Model-based methods, Model-free methods adopt an *online* learning strategy and do not require a perfect model of the environment. In this group is the largest number of related works in the area of this survey and we will categorize them according to the classification depicted in Fig. 3: pure proposals with sequential learning (Section 3.2.1), pure proposals with parallel learning (Section 3.2.2), proposals combined with neural networks (Section 3.2.3) and proposals combined with fuzzy logic (Section 3.2.4).

3.2.1. Pure proposals with sequential learning

In this section, we describe the works that use Model-free techniques in their original formulation, as opposed to the proposals based on DRL or FRL, and with a sequential learning process. At each decision time, the value of a *single* state–action pair is updated in the table of values $Q(s,a)$. In this sense, these proposals are more likely to have long training times since the speed of convergence of the RL algorithms depends directly on the dimension of the state space and actions.

Peng et al. (2015) propose an approach to optimize task scheduling in Cloud. The proposal is based on RL and queuing theory. The states reflect the remainder of the buffer capacity of each VM. The authors use a state aggregation technique to accelerate the learning process with *Q-learning*. Thus, the capacity of a VM is divided into 5 possible categories (i.e. full, less, middling, more, vain). The actions represent the selection of one of the available VMs for scheduling the current task. The reward is 1, 0, or −1, considering the mean waiting time of recent user requests and if the VM selected was the one with the maximum capacity. Then, for experimentation, two types of methods for task submission were defined: *individual* and *grouped scheduling* methods. In *individual scheduling*, user requests arrive continuously (as in regular Web requests or database queries) and an immediate response is required. In this context, the proposal outperformed in terms of response time to strategies such as FIFO, *fair-scheduling* (Jang et al., 2019), *greedy-scheduling* (Dong et al., 2015), and *random-scheduling*. On the other hand, in *grouped scheduling*, user requests arrive in groups (e.g. as in scientific calculations or business statistics) and it is required that the scheduler optimizes the arrangement of tasks according to their resource requirements and the current state of the infrastructure. This approach outperforms two competitors: *genetic-algorithm* and *modified-genetic-algorithm* (Kaur and Verma, 2012) in terms of makespan. This work is limited in terms of achieved algorithm scalability since both the dimensions of the state space and the number of actions depend on the number of used VMs, which could make the problem difficult to solve in the context of tens of VMs. In fact, the authors perform the experiments with a maximum of 10 VMs and only considered homogeneous VMs. A heterogeneous infrastructure would make it possible to use VMs that best fit the resource requirements of different types of tasks, which helps to achieve a higher execution efficiency.

Xiao et al. (2017) propose a distributed mechanism for scheduling independent tasks in the context of hybrid Clouds, i.e. an infrastructure combining public Clouds and third-party Clouds. The authors aim to maximize the capacity of the available processing entities (PE) (physical or virtual machines) by considering a cooperation scheme between the schedulers of the different Clouds. The approach guides the scheduling decisions based on experience, and therefore, the problem of each scheduler is modeled as an MDP and *Q-learning* is used to obtain the appropriate scheduling policy. For the definition of the states, the authors consider the task type and the workload of all the possible units – i.e., PEs or schedulers at different layers – to which tasks can be assigned. The authors use an aggregation strategy to reduce the state space based on a predefined granularity parameter. The actions correspond to the selection of the unit to which a task will be assigned. In this sense, the proposal could have algorithm scalability problems since the space of actions depends on the number of units, which might be large in a real Cloud. Then, the reward is defined as an inversely proportional function to the response time associated with the scheduling, i.e. the reward corresponds to the objective of the optimization problem. The results show that the proposal improves the response time compared to five state-of-the-art scheduling algorithms (Xhafa and Abraham, 2009): *opportunistic-load-balancing*, *minimum-execution-time*, *minimum-completion-time*, *switching-algorithm*, and *k-percent-best*.

Duggan et al. (2017) propose an RL-based strategy to schedule migrations of VMs, taking into account the current use of network resources in a Cloud. The idea is to learn to determine the most appropriate time to migrate a group of VMs from an overloaded physical machine to an underloaded one. The proposal aims to reduce the saturation of network resources during rush hours, as well as to reduce the migration time of the involved VMs. The authors define a state space based on (*i*) the bandwidth level, determined by current usage concerning a threshold (ranging from 0 to 100 in percentages) and (*ii*) the current direction of the network traffic considering the previous two time steps of bandwidth utilization (i.e increasing, decreasing, stable). The actions to be carried out are to perform or delay the migration of a scheduled group of VMs. Also, it is used as negative reward based on the total migration delay. The RL-based algorithm used is *Q-learning*. For comparison purposes, the authors used an algorithm called *minimum-migration-time* (Beloglazov and Buyya, 2012), which migrates VMs based on the amount of RAM used. Besides, the authors define a cost function to evaluate the migrations based on the network saturation level at the time of migration, the migration duration, and a penalty in case of waiting. The proposal (Duggan et al., 2017), in comparison with the minimum-migration-time algorithm, was able to reduce migration cost, as well as the use of network resources measured as the extra amount of Gb consumed from the link. The results also show that the RL-based strategy learns to perform the migrations when there is less network traffic. Moreover, this strategy contributes to reducing network saturation during rush hours and reduces the duration of the migrations.

Soualhia et al. (2018) propose ATLAS+, a MapReduce-based (Lee et al., 2012) task scheduler for Hadoop (Glushkova et al., 2019). ATLAS+ is dynamic, adaptable and its goal is to minimize task failures, defined as unforeseen events in the Cloud environment such as data loss in storage systems, hard-drive failures, and so on. The proposed framework is based on 3 components: (i) a machine learning algorithm (Random Forest) to predict the probability of task failures, (ii) a dynamic predictor of possible infrastructure failures and, (iii) a scheduler based on policies generated by an MDP. For scheduling purposes, the stages in the life cycle of tasks is modeled (submitted, scheduled, waiting, running, completed, failed). Then, the possible actions to change the status of a task are: process, reschedule, or kill the task. The objective of this proposal is to reduce the task failures to have a minimum impact on their execution time. To learn the policy a variant of RL that starts with the *SARSA* algorithm (for further exploration of the policies) in the first 30 min is proposed, and then, *Q-learning* is used for further exploitation of acquired knowledge. Experiments show that this proposal outperforms other Hadoop schedulers as *FIFO* and *Fairy Capacity*. Reductions of 59%, 40%, and 47% in the number of failed tasks, the total execution time, and the task execution time, respectively, were observed. The approach also reduces the use of CPU and memory by 22% and 20%, respectively.

Dutreilh and Kirgizov (2011) present VirtRL, an autonomous solution to the problem of dynamic adaptation of the number of resources allocated to Cloud applications. VirtRL is based on the *Q-learning* algorithm. In VirtRL, the number of user requests per second, the number of VMs assigned to the application, and the average response time of requests are considered for the definition of the states. Then, the actions represent the number of VMs to acquire or release (experimental setup comprises actions bounded between −1 and 10) while the reward considers the cost of acquiring or maintaining the VMs and a penalty for Service-Level Agreement (SLA) violations. An SLA is a commitment between a service provider and a client. Particular aspects of the service such as quality, availability, and responsibilities are agreed between the service provider and the service user. Concretely, the main contribution of this work (Dutreilh and Kirgizov, 2011) is to include and integrate into an automatic Cloud autoscaler the following elements, (i) initialization of the function $Q(s,a)$, (ii) acceleration in the convergence at regular intervals of observations and, (iii) a mechanism for detecting changes in the performance model.

Moreover, Ghobaei-Arani et al. (2018) propose an RL-based resource provisioning approach for Cloud service applications. The *Q-learning* algorithm is used for a decision-making agent to learn when to add or remove VMs to find a satisfactory compensation between SLA and costs. The authors define 3 possible states based on the CPU utilization degree of the infrastructure, i.e., when the CPU utilization is less than the lower threshold, the state is labeled as under-utilization. Moreover, when the CPU utilization is greater than the upper threshold, the state is labeled as over-utilization. Otherwise, the state is normal-utilization. Then, based on the current state one out of 3 possible actions is selected: scale in, scale-out or no-action, which means increasing, reducing, or maintaining the infrastructure. The reward function R assigns a fixed value for each state–action pair, while prioritizing Scale-out if the state is under-utilization, Scale-in if the state is over-utilization, and No-action if the state is normal-utilization. The performance was evaluated with real workloads and the approach was compared with 3 state-of-the-art strategies: Cost-aware-LRM (Yang et al., 2014) and Cost-aware-ARMA (Roy et al., 2011), which are both based on workload predictions, and DRPM (Al-Ayyoub et al., 2015), a multi-agent system to monitor and provision Cloud resources. The results showed that this approach was able to reduce by 50% the total cost and increase the use of resources by 12%.

Dezhabad and Sharifian (2018) address the automatic autoscaling of virtualized firewalls in a Cloud. The authors propose GARLAS, a solution that combines RL with a genetic algorithm and queue theory. The idea of this work is to determine the number of firewalls that must be active at all times according to the intensity of the input load and the proportion of requests that each one handles. This approach aims to optimize the balance between the firewalls use degree and compliance with SLA related to system performance (for example, response time). On the one hand, an automatic autoscaler based on RL that decides when it is convenient to increase or reduce the number of active firewalls by dynamically adjusting the system to avoid overloading or wasting resources, is proposed. On the other hand, a genetic algorithm is responsible for deciding the appropriate proportion of requests that each of the firewalls must handle, thus balancing the load to minimize the system response time. For the RL-based autoscaling problem, states that consider the current request rate and the number of active firewalls are defined. The actions consist of increasing, reducing, or maintaining the number of active firewalls, and the reward is responsible for penalizing overload or low load states, as well as SLA violations. Then, through the Q-learning algorithm, it is possible to converge to an appropriate scaling policy. The proposal was compared with static strategies (number of fixed firewalls) and rule-based strategies (number of firewalls varies according to load levels). The results show that GARLAS was able to significantly reduce the response time of the system (by more than 80%) and also offers improvements in the use of resources (more than 9%). This is due to a better load balance and a more precise automatic scaling algorithm.

Horovitz and Arian (2018) present a Q-learning solution for horizontal scaling that adds initialization and smoothing rules combined with a utilization rate of states change. In this approach, a state space reduction method is used by exploiting the monotonic behavior of the actions taken as a function of the state space (e.g. utilization). Then, an action space reduction method is used for those actions that are continuous for a given state. Lastly, an innovative approach to Q-Learning based auto-scaling is applied, the Q-Threshold algorithm. The authors define the state space as the current number of resources allocated to the application. The actions consist of selecting a utilization threshold from N possible values. The threshold value serves as bound for a given resource (e.g. CPU utilization, load, response time, etc.). The authors maintain two Q-Tables, one for the upper threshold and another for the lower threshold. Regarding the reward, it balances the response time and the average resource utilization, i.e., the reward tries to meet the SLA while making the resource utilization as high as possible. Therefore, instead of the traditional action space where machine addition and removal actions are used, the thresholds values drive the actions, and the traditional thresholds are dynamically controlled by the algorithm. Q-Threshold learns the best resource utilization thresholds in the horizontal autoscaling problem to derive the optimal policy while abiding by the SLA and maximizing resource utilization.

Finally, Wei et al. (2019) propose an approach based on the Q-learning adjustment algorithm (QAA) to help SaaS providers make optimal resource allocation decisions in a dynamic and stochastic Cloud environment. The goal of this work is to reduce renting expenses as much as possible while providing sufficient processing capacity to meet customer demands. For this, the authors have considered different VM pricing models, including on-demand and reserved instances. The authors consider the following features for the states: the average customer workload, the number of VMs of each type, and a reference to the specific time within the workload period. Each action comprises the number of VMs of each type that will be acquired for the next execution period. Also, the reward function is calculated based on the profit that SaaS provider earned by providing service to his end-users, and on the performance (the gain of application performance), which depends on the resource utilization levels. If SaaS provider owns sufficient VM instances to execute customer workloads, a positive reward will be received. In contrast, a penalty will be obtained if application processing capacity is lower than the customer's demands. The value of reward or penalty is related to the distance between the offered processing capacity and the real customer workload. SaaS provider keeps learning from previous renting experiences and enriching its knowledge. This accumulated information can help the provider know the best choices in different situations and then generate an efficient renting policy for each decision period. Through a series of experiments and simulations, the authors evaluate QAA under different pricing models (on-demand and reserved instances) and compare it with two other resource allocation strategies: empirically-based adjustment algorithm (EAA) and threshold-based adjustment algorithm (TAA). EAA adopts a simple strategy to generate a new renting policy. Since SaaS provider does not know the upcoming customer workload when making decisions, the provider adjusts the number of rental VM instances according to the last workload. TAA is similar to EAA but does not change the renting policy each time. Only when the difference between customer workload and processing capacity offered by the SaaS provider exceeds a specific threshold, a new renting policy will be generated.

3.2.2. Pure proposals with parallel learning

In this section we describe those works from the literature that use Model-free techniques in their original formulation (as opposed to proposals based on DRL or FRL), but with a *parallel* learning process. The main advantage of parallel learning is that training time is reduced due to multiple agents simultaneously sharing the acquired knowledge, i.e. the agents periodically exchange information thus accelerating convergence towards the optimal policy. In this way, the table $Q(s,a)$ is updated at a faster pace, so it is possible to obtain a higher quality policy faster, at the expense of higher approach design complexity.

Barrett et al. (2012) propose CloudRL, a method based on MDP and *Q-learning* for dynamic scaling in IaaS infrastructures, in response to changes in workload and infrastructure performance. The states are defined based on the number of user requests, the number of VMs of each type and region, and the *Coordinated Universal Time* (CUT). Actions are either requesting, maintaining, or removing instances, while the reward includes the cost and a penalty in case of SLA violations. Particularly, the authors introduce a strategy with multiple agents learning in parallel to mitigate the problem of long convergence time of *Q-learning*. The long convergence time of *Q-learning* is due to it does not have a good initial approximation of π and a good initialization $Q(s,a)$. The authors also suggest that parallel learning is scalable in terms of resource growth because the number of learning agents can be determined based on the number of available computational resources. In this sense, the proposal should also be scalable in terms of the number of user requests. If we take into account the algorithm scalability in both dimensions, the state space would grow considerably. As a consequence, this would have an impact not only on the parallel computing capacity of the agent but also on the storage capacity to keep accessible information shared between them and the mechanisms for sharing such information.

Bibal Benifa and Dejey (2018) present RLPAS, an RL-based approach for automatically scaling virtualized resources in a Cloud. The objective of the proposal is to dynamically configure resources to minimize response time while maximizing resource utilization and performance. The states are defined based on the number of user requests, the infrastructure utilization degree (the relationship between acquired and used VMs), as well as the response time and performance observed for each task during a pre-determined period. Then, the *scale-up* (or *scale-down*) actions comprise the number of VMs of each type that will be acquired (or released) in the next execution period and *no-action* specify that changes are no required. Moreover, the reward is based on the relationship between performance (related to response time, throughput, and SLA violations) and the VMs utilization degree. This approach, based on the *SARSA* algorithm, reduces convergence time, and combines parallel learning with an approximation of the function $Q(s,a)$. This approximation is performed by the gradient descent method. For experimentation, reference applications with dynamic workloads were used and RLPAS was compared with the pure variants of *Q-learning* and *SARSA*, as well as with the approach proposed in Barrett et al. (2012) and discussed in the previous paragraph. RLPAS outperformed its competitors in terms of CPU utilization, response time, performance (number of requests processed per second), and convergence time.

Asghari et al. (2021) combine SARSA with GA for resource management in Clouds during the execution of workflow applications. The objectives comprise the minimization of makespan, cost and resource utilization, as well as maximizing load balancing. The proposal includes a first RL stage to decide the order for executing the workflow tasks (named scheduling phase). The states correspond to the tasks in the workflow. The actions represent the selection of one of the successor tasks to determine the execution order. The reward includes the processing time and the communication cost involved in the current transition. Moreover, there is a second RL stage (named resource provisioning) where multiple agents assign each ready task to a specific resource for execution, aiming to minimize resource utilization. In this case, the actions correspond with the selection of a ready task for execution in a predefined resource and the reward measures the resource utilization. Then, a GA is utilized for convergence of the agents to achieve global optimization. Experiments were conducted in a simulated environment using CloudSim. Finally, the proposal shows an improved performance in comparison with two workflow scheduling algorithms from the literature (i.e., MOHEFT — Multi-objective Heterogeneous Earliest Finish Time and MCP — Modified Critical Path).

Nouri et al. (2019) present a decentralized RL-based technique for responding to volatile and complex arrival of tasks through a set of simple states and actions. The technique is implemented within a distributed architecture that cannot only scale up quickly to meet rising demand but also scale down by shutting down excess servers to save costs. The states consist of two types of attributes: system state and application state. System state reflects the level of utilization of resources of a server such as CPU, and the application state represents the performance of each application hosted on the server in terms of metrics such as its response time. To make the state–space discrete, the system states and application states are classified into three categories: normal, warning, and critical. On the other hand, the actions are categorized into two groups: scale-down and scale-up. Scale-up actions would be suitable when the system is not able to meet the SLA and needs more computing resources. In contrast, scale-down actions suit situations in which the system is in normal condition, and idle resources can be released to minimize cost. The application actions involve either duplicating, or creating extra instances of an application, or move, wherein an application is shifted to a different server with more available resources. The reward for reaching a state is determined by the summation of all VM utilities. The utility is based on performance, in terms of the response time of requests, and cost. In this approach, it is feasible to share the states, take actions, and receive rewards among the servers to speed up the learning process. Hence, if a server reaches a state which has not been observed by itself, it tries to find the knowledge of the state from the shared knowledge base. In case that the look-up procedure produces no result, it will take the best possible action using the learning policy. Furthermore, this procedure allows new servers to initialize their knowledge database using the existing shared knowledge. The authors evaluate the decentralized control technique using workloads from real-world use cases and demonstrate that it reduces SLA violations while minimizing the cost of infrastructure provisioning.

3.2.3. Proposals combined with neural networks

In this section we describe the works that propose solutions exploiting neural networks (see Appendix A.3), combining techniques of RL with Deep Neural Networks (DNN) to mitigate the problem of the state dimensionality associated with RL-based techniques in its purest variant. This combination is called Deep Reinforcement Learning (DRL). When including the use of DNN in the decision-making process, it is important to consider that the solution becomes more complex, in addition to the problems inherent to DNN (Marcus, 2018). On the one hand, deep learning models usually include a high number of hyper-parameters (for example, *learning rate*, *batch size*, *momentum*, and *weight decay*) (Smith, 2018) and finding the best configuration for these parameters in a large dimensional space is not trivial. On the other hand, DNNs require a large volume of data and consequently a lot of training time. Nevertheless, DRL has proven to be a promissory technique since it allows working with very complex state spaces and actions. In this sense, the following works show a first approach to applying DRL to the area of autoscaling in Clouds.

Liu et al. (2017) propose a hierarchical framework to solve two important problems in the context of Cloud Computing: task scheduling and the management of energy consumption of the infrastructure. The framework consists of a global decision layer for the scheduling problem and a local decision layer for distributed energy management in local PM. At the global framework level, a DRL-based strategy capable of handling the complex state space and actions that characterize the problem is proposed. In the definition of the states, infrastructure information (utilization degree of each PM) and task information (resource requirements and estimated duration) are represented. The actions correspond to the assignment of the tasks to some of the existing PM. Then, the reward is composed of three terms with values negatively weighted of instantaneous total power consumption, the number of VMs in the system, and reliability objective function value. The DRL-based strategy consists of an *offline* construction stage of the neural network, which represents the correlation between the estimated values of Q and the proposed state–action pairs. Then, the strategy continues to operate in an *online* stage of decision-making and learning-based both on *Q-learning* and the update of the neural network previously trained. On the other hand, the local level of the framework is responsible for energy management, with a distributed mechanism to selectively turn on and off the PMs. This level includes a workload predictor based on a Long Short-Term Memory (LSTM)[4] neural network (Hochreiter and Schmidhuber, 1997). Then, an RL-based adaptive energy manager controls the status of any PM based on the workload predictions made by the LSTM. The authors use Google server logs for experiments and the Round-Robin scheduling method for comparisons. The results show that the RL-based proposal achieves significant energy savings, as well as the best relationship between latency (defined as the time between the arrival of a task and its completion) and energy consumption.

Mingxi Cheng and Nazarian (2018) present DRL-Cloud, an approach based on DRL for workflow scheduling in Clouds. The objective of this strategy is to minimize the energy cost from the perspective of public Cloud providers. For them, the authors propose a two-stage (i.e. resource provisioning and task scheduling) process based on DRL, which is highly scalable and adaptable. The definition of the states includes infrastructure information (CPU and RAM availability) and workflow task information (deadlines, and CPU/RAM requirements). In the first stage, the actions correspond with the selection of one of the servers farm for allocating the task and the possible start time. In the second stage, the actions represent the selection of a specific server to run the task. The reward on each stage is computed based on the energy cost increase relative to the farm or the server, respectively. DLR-Cloud is fully parallelizable and uses training techniques (Mnih et al., 2015) (*target-network*,[5] *experience-replay*[6]) to accelerate convergence. The proposal was compared with two methods: Fast and Energy-Aware Resource Provisioning and Task Scheduling (FERPTS) (Li et al., 2017) and Round-Robin. Results show significant improvements in the reduction of energy costs, the number of rejected applications (due to deadline violations), and the execution time.

Wang et al. (2017) explore the use of RL techniques for horizontal scaling in the Cloud. The idea of this work is to learn policies capable of adjusting the infrastructure, achieving a balance between performance and costs. The states comprise the number of VMs, two instance-level CloudWatch[7] metrics (CPU Utilization, Network Packets In), and two elastic load balancer-level metrics (Latency, Number of Requests). The actions include increasing and reducing the infrastructure in one or two VMs, as well as maintaining the current resources. A negative reward is computed considering the cost of the provisioned resources and a penalization relative to the CPU utilization (depending on the SLA). The authors show a preliminary study of 3 strategies of RL: *tabular-Q-learning* (QL), *deep-Q-network* (DQN) y *double-dueling-Q-network* (D3QN), first in the CloudSim simulator (Humane and Varshapriya, 2015) and then in the Amazon Cloud. QL corresponds to a classic variant of *Q-learning* where the function Q is represented in tabular form. DQN uses a deep neural network to estimate the Q function. D3QN is another variant that uses a second neural network (Double Q-Network (Wang et al., 2015)) to stabilize the training process and to update the value of the states in a more robust and decoupled form from the specific actions (*Dueling Q-network* (Wang et al., 2015)). The study compares a dynamic scaling method based on predefined thresholds of CPU usage (threshold-based method) with the three above mentioned methods based on RL. The results show the superiority of the proposed DRL-based methods. Besides, D3QN significantly outperformed DQN in terms of the accumulated reward and learning speed.

Tong et al. (2020) propose a deep Q-learning task scheduling (DQTS) that combines the advantages of the Q-learning algorithm and a deep neural network. This approach is aimed at solving the problem of scheduling workflow tasks in a Cloud and uses the popular deep Q-learning (DQL) method (Alam et al., 2019). The goal is to learn policies capable of adjusting resource utilization while minimizing the makespan. The approach comprises three layers: task submission layer, deep Q-learning algorithm layer, and workflow management system layer. The states comprise a normalized and weighted combination of the current task processing time and the current accumulated processing time on each VM. The actions represent the selection of one VM for scheduling the task and the reward is computed as a positive or negative value considering the condition of the selected VM (i.e. idle or busy).

Du et al. (2019) propose a DRL approach to make Cloud resource pricing and allocation decisions for reducing cost. For this, the authors consider both the VM pricing and placement of VMs in the DRL model, and time-variant user dynamics, rather than simply assuming user arrivals are independent and identically distributed. The approach combines long short-term memory (LSTM) networks with the Deep Deterministic Policy Gradient (DDPG) method to deal with online user arrivals, and use a new update method to allow them to work together and to learn optimal decisions directly from input states. In this approach, for each VM request, the provider decides the server with available capacity to allocate the VM and the cost for running the VM on the selected server. The states are composed of two parts. The first part includes the current resource availability on all servers and information about the new VM requests, and the second part includes the states history encoded by the LSTM. Then, the action space of the DRL agent includes both discrete actions for server selection and continuous actions for pricing. The reward, when a user accepts the posted price, is the payment of the user minus the increased cost of the server due to running the user VM; otherwise, the reward is 0. The

[4] This kind of neural networks is widely used for predicting time series sequences.

[5] *Target Network*: strategy that proposes the use of a second <<objective>> network, during the training of a DQN, to calculate the updated values of Q. In this way, a more stable training is achieved since the weights of this second network are updated less frequently than those of the original network.

[6] *Experience Replay:* a strategy that proposes to store the agent's experiences and then use random data for the training of the DQN. In this way, correlations in the observation sequences are eliminated and changes in data distribution are smoothed out.

[7] Amazon CloudWatch: https://aws.amazon.com/cloudwatch/.

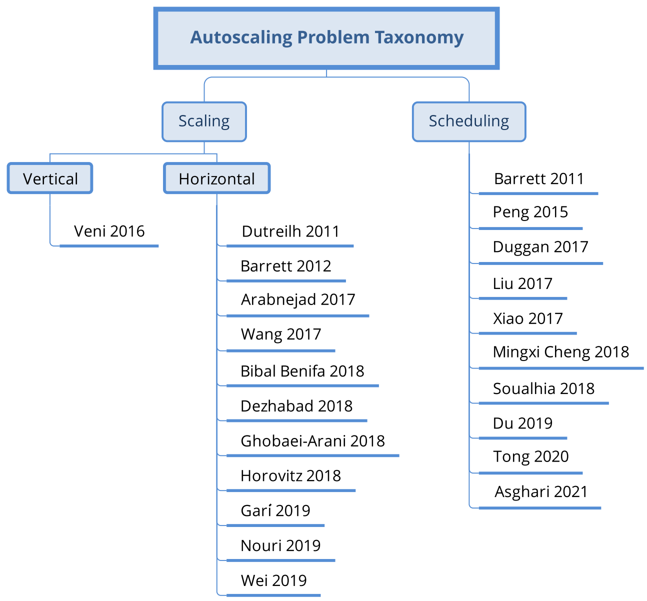

Fig. 4. Taxonomy of autoscaling problems in Cloud addressed with RL techniques.

output of the neural network is divided into two parts: one gives the probability distribution for choosing among different servers and the other produces the corresponding unit-time-usage cost on the servers if the requested VM is to be allocated there. Therefore, the goal of this approach is to learn a policy that minimizes the costs of the Cloud provider.

3.2.4. Proposals combined with fuzzy logic

This section discusses the works that combine RL-based techniques with elements of Fuzzy Logic (FL) as another alternative to the dimensionality problem that arises in the purest variants of RL.

Arabnejad et al. (2017) address the problem of horizontal resource scaling in the Cloud to reduce application costs and ensure SLA compliance. The states are defined based on the workload, the response time, and the number of VMs. The actions are either increasing (one or two VMs), reducing (one or two VMs), and maintaining the infrastructure. The reward depends on the number of resources acquired and SLA violations. In this work, the authors propose and compare two strategies based on RL and an FL system. The modified versions (*Fuzzy Q-learning* and *Fuzzy SARSA*) of the classic RL algorithms can learn and modify fuzzy scaling rules during execution. Both proposals are implemented and compared on the OpenStack[8] Cloud platform. The results show that both proposals can handle different workload patterns, reduce operational costs, and prevent SLA violations, with acceptable performance in terms of response time and the number of used VMs. It is important to note that the authors use an environment limited to a maximum of 5 VMs to evaluate the strategies together with high workloads.

Veni and Bhanu (2016) present an approach for vertical scaling of virtualized resources in Clouds. The proposal is based on neuro-fuzzy reinforcement learning, combining RL with neural networks and the approximate reasoning of fuzzy logic. This combination aims to mitigate the limitations of the most basic variants of RL in a space of large states. The main objective of this work is to dynamically configure the resources of each VM based on the current workload to achieve the maximum overall system performance with minimum resource utilization. The states are defined according to three VM characteristics (CPU time, CPU number, and memory size). The action set describes actions such as increase, decrease, or maintain each configurable resource of a VM. The reward considers the relationship between the overall performance of the system and the resource utilization degree. The proposal was compared with a basic variant of RL (Basic-RL) and with a strategy that combines RL and neural networks (VCONF (Rao et al., 2009)). The results show that the proposal achieves significant improvements in system performance and scalability. Recall that vertical scaling is limited by the underlying hardware. This means that the characteristics of the VMs can be improved only as far as resources are available in the PMs that allocate them. For the scaling problem of applications in the Cloud, it would be interesting that this proposal combines vertical scaling with some variant of horizontal scaling to better mitigate hardware limitations.

[8] OpenStack: Open Source Platform for Cloud Computing (https://www.openstack.org/).

3.3. RL elements summary

This section summarizes the RL elements (states, actions, reward, and time between actions) considered in each work. Table 1 present (*i*) the features considered for defining states, (*ii*) the action space, (*iii*) the model of the environment, (iv) the discount factor, (v) an informal notion of the reward function, and (*vi*) the time between actions. Please note that the table does not include the RL element "model of the environment" for all the surveyed works. The entry is only present for the two works discussing *model-based* methods. In the context of autoscaling in Cloud, the states usually reflect information about the infrastructure (number of VMs, workload, utilization level, etc.) and the application conditions (performance, requirements, etc.). Notice that, some works are focused on scaling actions related to the adjustment of the available resources (e.g. increase, reduce or maintain the number of VMs). Notice also that other works are focused on scheduling actions that reflect the selection of a specific resource where the current task will run. Moreover, different and complex reward (or penalization) functions have been defined to optimize multiple objectives (e.g. execution time, economic cost, resource utilization, SLA, etc.). The time between actions can be fixed (i.e., 5 min or 1 hour) or variable, e.g. when a new user request (or a new task or task batch) arrives.

3.4. Classification of the reviewed approaches

For the sake of organizing the surveyed related works, we present three taxonomies that describe the autoscaling problem in Cloud (see Section 3.4.1), the types of executed applications (see Section 3.4.2), and the optimization objectives that are addressed in each work (Section 3.4.3), respectively. Then, in Section 3.4.4, an in-depth comparative analysis based on the surveyed works and defined taxonomies is presented.

3.4.1. Taxonomy of autoscaling problems

Fig. 4 presents a taxonomy of the main addressed problems in the context of autoscaling in Cloud, where the scaling and the scheduling component are dealt with as two different complex optimization problems. Works make the most out of the available resources to achieve the best performance regarding the objectives to be optimized. The optimization objectives (for example, execution time and cost) guide the search of possible policies, determining when one policy is more convenient than the other. On the other hand, the constraints (for example, response time less than 5 s) reduce the search space to include only acceptable policies, leaving out those that do not comply with the constraints defined for the problem. Besides, there are also SLAs, which represent an agreement between the user and the service provider. This agreement defines which aspects must be respected in the quality of the provided service and represents problem's constraints. Although optimization objectives and constraints may have a non-empty intersection, it is important to consider that they have different roles within the optimization problem.

Table 1
Unified view of the main RL elements in each surveyed work.

Reference	RL elements
Garí et al. (2019)	**States:** The proportion of long tasks, the maximum parallelism degree, the budget limit, and the probability of out-of-bid error **Actions:** The possible budget assignment ratio between spot and on-demand instances. Values: $\{0.0, 0.1, \dots, 1.0\}$ **Model of the environment:** Is an estimation obtained through the execution of a set of heuristic-based autoscaling methods. **Discount factor:** $\gamma \in \{0.1, 0.5, 0.9\}$ **Reward:** The ratio between the execution progress and the cost in the last period **Time between actions:** 1 h
Barrett et al. (2011)	**States:** A time-stamp within the workload period (0–23 h), the resource load (light, moderate or heavy) and the execution result (failed or successful) **Actions:** The possible execution plans generated by the GA solvers. Values: $\{1, \dots, N\}$ **Model of the environment:** Is an estimation obtained through experience using a Bayesian Model Learning approach. **Reward:** The cost and a penalization based on violated SLA **Discount factor:** Unspecified **Time between actions:** 1 h
Peng et al. (2015)	**States:** The remainder of the buffer capacity of each VM (5 discrete levels) **Actions:** The available VMs for allocating the current task. Values: $\{1, \dots, N\}$ **Reward:** 1, 0, or −1, considering the mean waiting time of recent user request and if the VM selected was the one with the maximum capacity **Discount factor:** Unspecified **Time between actions:** Time until the new user request
Xiao et al. (2017)	**States:** The task type and the workload of all the possible units (i.e PMs/VMs or schedulers at different layers) to which the tasks can be assigned **Actions:** The available units for allocating the current task. Values: $\{1, \dots, N\}$ **Reward:** An inversely proportional function to the response time associated with the scheduling **Discount factor:** $\gamma = 0.9$ **Time between actions:** Time until new task arrival
Duggan et al. (2017)	**States:** The bandwidth level (ranging from 0 to 100 in percentages) and the direction of the network traffic (increasing, decreasing, stable) **Actions:** Wait or migrate a scheduled group of VMs **Reward:** A penalization based on the VM migrations delays **Discount factor:** $\gamma = 0.5$ **Time between actions:** 5 min
Soualhia et al. (2018)	**States:** The task status. Values: $\{Submitted, Scheduled, Waiting, Executed, Finished, Failed\}$ **Actions:** Process, reschedule, or kill the task **Reward:** Unspecified **Discount factor:** Unspecified **Time between actions:** Unspecified
Dutreilh and Kirgizov (2011)	**States:** The number of user requests, the number of VMs allocated to the application, and the average response time **Actions:** The number of VMs to acquire or release. Values: $\{-1, 0, +1, \dots, +10\}$ **Reward:** The cost and a penalization based on violated SLA **Discount factor:** $\gamma = 0.45$ **Time between actions:** Unspecified
Ghobaei-Arani et al. (2018)	**States:** The CPU utilization level. Values: $\{UnderUtilization, NormalUtilization, OverUtilization\}$ **Actions:** Reduce (−1), maintain, increase (+1) the number of VMs **Reward:** A set of precomputed reward values, prioritizing for each state, the action that leads to a balanced CPU utilization **Discount factor:** Unspecified **Time between actions:** 5 min
Dezhabad and Sharifian (2018)	**States:** The current request rate and the number of active firewalls **Actions:** Reduce (−1), maintain, increase (+1) the number of active firewalls **Reward:** A penalization based on violated SLA and disadvantageous firewall loads (over-load/under-load) **Discount factor:** $\gamma = 0.8$ **Time between actions:** 5 min
Horovitz and Arian (2018)	**States:** The number of VMs allocated to the application **Actions:** Predefined utilization thresholds. Values: $\{1, \dots, N\}$ **Reward:** A balance between the response time and the average resource utilization **Discount factor:** $\gamma = 0.5$ **Time between actions:** 1 min
Wei et al. (2019)	**States:** The average customer workload, the number of VMs of each type and a time-stamp within the workload period **Actions:** The number of VMs of each type to acquire **Reward:** The profit earned by SaaS provider and the gain of application performance **Discount factor:** $\gamma = 0.5$ **Time between actions:** 1 h

(*continued on next page*)

The *scaling problem* consists of determining the appropriate decisions behind provisioning and releasing virtualized resources, according to the fluctuations in their demands, and achieving more efficient use of them. In other words, an automatic mechanism is required to increase or reduce the infrastructures based on the current resource needs and taking advantage of the elasticity inherent to Clouds.

Scaling can be classified as horizontal scaling or vertical scaling (Spinner et al., 2014). In *horizontal scaling*, the number of VMs

Table 1 (*continued*).

Reference	RL elements
Bibal Benifa and Dejey (2018)	**States:** The number of user requests, the infrastructure utilization degree, and (for each task) the response time and throughput **Actions:** The number of VMs of each type to acquire or release **Reward:** The ratio between Performance (i.e response time, throughput, and SLA violations) and VMs utilization degree **Discount factor:** $\gamma = 0.9$ **Time between actions:** 1 min
Barrett et al. (2012)	**States:** The number of user requests, number of VMs of each type and region, a time-stamp **Actions:** Reduce (−1), maintain, increase (+1) the number of VMs **Reward:** The cost and a penalization based on violated SLA **Discount factor:** $\gamma = 0.85$ **Time between actions:** Unspecified
Asghari et al. (2021)	**States:** The workflow tasks **Actions:** Stage 1: Selection of a successor task to define the execution order, Stage 2: Selection of a ready task for executing in a specific resource **Reward:** Stage 1: processing time and communication cost, Stage 2: resource utilization **Discount factor:** Unspecified **Time between actions:** Unspecified
Nouri et al. (2019)	**States:** The system state (i.e., CPU utilization) and the state of each application (i.e., average response time) **Actions:** Server-related actions are $\{Start, Terminate, Find\}$; Application-related actions are $\{Duplicate, Move, Merge\}$ **Reward:** The summation of VMs utilities (i.e., response time and cost) **Discount factor:** $\gamma = 0.98$ **Time between actions:** 30 min
Arabnejad et al. (2017)	**States:** The workload, the response time, and the number of VMs **Actions:** The number of VMs to acquire or release. Values: {−2,−1,0,+1,+2} **Reward:** A penalization based on violated SLA and the number of VMs **Discount factor:** $\gamma = 0.8$ **Time between actions:** Unspecified
Veni and Bhanu (2016)	**States:** The configuration of each VM (i.e. CPU time, number of CPUs, and memory size) **Actions:** Reduce, maintain or increase for each configurable resources of a VM **Reward:** The ratio between Performance (i.e., response time, throughput, and SLA violations) and VM utilization degree **Discount factor:** $\gamma = 0.1$ **Time between actions:** 1 min
Wang et al. (2017)	**States:** The number of VMs, two instance-level Amazon CloudWatch metrics (i.e., CPU Utilization, NetworkPacketsIn), and two elastic load balancer-level metrics (Latency, Number of Requests) **Actions:** The number of VMs to acquire or release. Values: {−2,−1,0,+1,+2} **Reward:** A penalization based on cost and CPU utilization **Discount factor:** $\gamma = 0.99$ **Time between actions:** 5 min
Liu et al. (2017)	**States:** The infrastructure state (i.e., utilization degree of each PM) and the task state (i.e., resource requirements and estimated duration) **Actions:** The available PMs for allocating the current task. Values: $\{1,\ldots,N\}$ **Reward:** Penalization based on power consumption, number of VMs, and reliability issues **Discount factor:** $\gamma = 0.5$ **Time between actions:** Time until new task arrival
Mingxi Cheng and Nazarian (2018)	**States:** The server state (CPU/RAM availability) and the task state (deadline and CPU/RAM requirements) **Actions:** Stage 1: The available server farms for allocating the current task and the possible start time. Values: $\{F_{1,1},\ldots,F_{N,T}\}$ Stage 2: The available servers within a specific farm for allocating the current task. Values: $\{S_1,\ldots,S_M\}$ **Reward:** Stage 1: energy cost increase in the Farm, Stage 2: energy cost increase in the server **Discount factor:** $\gamma = 0.9$ **Time between actions:** Unspecified
Tong et al. (2020)	**States:** Normalized and linear combination of current task-processing time and current accumulated processing time for each VM **Actions:** The available VMs for allocating the current task. Values: $\{1,\ldots,N\}$ **Reward:** 1 if the selected VM is idle and −1 if busy **Discount factor:** $\gamma = 0.5$ **Time between actions:** Unspecified
Du et al. (2019)	**States:** The current resource availability of each server, information of the new VM request, and states history encoded by the LSTM network **Actions:** The probabilities and prices of selecting the respective servers **Reward:** The profit (i.e. user payment-cost) if the user accepts the posted price or zero in other case **Discount factor:** $\gamma = 0.99$ **Time between actions:** Time until the new user request

assigned to an application across the infrastructure is increased or reduced. The idea is to provide the application with the appropriate number of VMs according to the needs at a given time, to fully exploit the capabilities of parallelization in high demand periods, and reduce the number of resources in low demand periods. This type of scaling requires spending time for initialization of the VMs and it may not be appropriate for all types of applications, for example, database-oriented applications where splitting and distributing data is not trivial. In *vertical scaling*, the resource settings (CPU, memory, I/O) are dynamically updated, increasing or reducing the VMs capabilities, without hindering the execution units where VMs are running. The idea is to constantly provide the VMs with the necessary capabilities and update them again when they are no longer required. This type of scaling usually requires less time for resource configuration (less than 0.5 s) than the time required for horizontal scaling (5 min) (Spinner et al., 2014).

On the other hand, the *scheduling problem* in the Cloud consists of automatically determining the appropriate decisions about when or where a task should be executed to achieve the greatest possible efficiency within the limits defined by the existing constraints. From a temporal perspective, this problem decides which is the most appropriate time for executing a task. In the context of workflow scheduling, this decision is primarily subject to the order established by the structure of dependencies between tasks. The scheduling problem can be addressed from the perspective of prioritizing the execution of critical tasks (Monge et al., 2017) because of the impact they have on the workflow makespan. On the other hand, from a spatial perspective, scheduling aims to make the connection between each workflow task and the most appropriate resources to perform their execution. For this, the scheduler usually uses information related to the characteristics of the tasks and the resources available to support the decision-making process. When the scheduler does not have accurate task information, estimations are used, usually for task durations.

3.4.2. Taxonomy of application types

In Cloud Computing, we can find different types of applications that need to run efficiently. The reason can be because the applications involve many hours/days of computation, because they require many resources, or because immediate results are needed for multiple user requests. All these distinctive elements are the ones that also determine the most convenient type of strategy for making decisions when performing the autoscaling process. In this sense, it is important to classify the applications and analyze their characteristics. Fig. 5 shows a taxonomy according to the application types (see Section 2.1) targeted by the works analyzed in the previous sections.

Workflow applications are increasingly used for modeling complex scientific experiments. Workflows are usually composed of hundreds or thousands of computing-intensive or data-intensive tasks with different durations and resource requirements. The dependencies between workflow tasks determine the order in which tasks are executed since a task cannot begin its execution until all the tasks on which it depends have been completed. The complex structures of dependencies between workflow tasks determine a variable workload during execution. This variability is evident when many tasks can be executed in parallel (high demand for resources). In other cases, bottlenecks may occur, and it is necessary to wait for the completion of some tasks before starting the execution of other tasks (low demand for resources). In this sense, the dependencies between workflow tasks add an important degree of difficulty to the autoscaling problem. When the workflow structure is known in advance, it is possible to use workload estimates to support proper decision-making in the autoscaling process.

Other applications are composed of *independent tasks*. In some cases, the tasks are randomly initialized by the users, so they arrive individually and continuously, while in other cases the tasks arrive in batches. When tasks arrive individually and a quick response is required, the autoscaling decisions are based only on the characteristics of the current task and on the state of the infrastructure. In this sense, autoscaling strategies are intended to provide immediate responses. On the other hand, when the tasks arrive in batches, the number of options to consider increases exponentially, so strategies that search in the space of possible solutions are preferred. An example of this type of applications is parameter sweep experiments (PSE). PSEs are a very popular way of conducting simulation-based experiments, used by scientists and engineers, through which the same application code is run several times with different input parameters resulting in different output data (Makris, 2003; García Garino et al., 2013). PSEs involve large-scale computer modeling and simulation and often require large amounts of computer resources to satisfy the ever-increasing resource-intensive nature of their experiments. Running PSEs involves managing many independent tasks. Moreover, it is important to mention that an independent task application can be considered a special type of workflow where there are fictitious start and end tasks, and all other tasks are intermediate tasks within the workflow structure (Monge et al., 2018).

The third type of application is *Cloud services*. These applications represent a software product running on Cloud, that can be accessed through the Internet either with a Web browser, a mobile application, or via an API. For example, applications for Big Data Analytics enable data scientists to tap into any organizational data to analyze it for patterns and insights, find correlations, make predictions, forecast future crises, and help in data-backed decision making. Cloud services make mining massive amounts of data possible by providing higher processing power and sophisticated tools. Generally, each Cloud service application is composed of one or more services that together perform the functions of the application. In this type of applications, the response time to user's requests is critical. The applications must be scalable in managing the number of requests, which usually generate high demand peaks. Although anatomically these applications do not always strictly comply with a workflow-like structure, we also consider them in this survey as these are heavily used in practice to execute resource-intensive models and data analyses on Cloud infrastructures.

Considering the different Cloud service models (SaaS, PaaS, and IaaS) described in Section 2.2, it is interesting to highlight that from the surveyed works, Cloud service applications are designed under the SaaS model, while workflows and independent tasks based works are intended for IaaS and PaaS models.

3.4.3. Taxonomy of optimization objectives

When optimizing the execution of applications in the Cloud, different objectives have been addressed. It is very important to identify these objectives because they represent the direction in which the efforts will move in guiding the search among possible solutions. The optimization of multiple objectives is also a recurring issue, and in some cases, the objectives conflict with each other, as in the classical trade-off between time and cost in paid Clouds (Monge et al., 2020). Fig. 6 presents a taxonomy of the objectives targeted in the surveyed works.

First, there are three objectives related to *time*. On the one hand, it is important to optimize the *makespan* in applications consisting of compute-intensive or long-duration tasks, usually the case of workflow applications. Note that optimizing workflow makespan is more complex than optimizing the execution time of each of the tasks that compose a workflow since the makespan also depends on the order in which these tasks are executed. On the other hand, the *waiting time* refers to the time between a task is submitted and actually begins to execute. The waiting time is usually due to overloaded infrastructures with high demand peaks, and it is of special interest in applications that process independent tasks. In the case of workflows, the waiting time is also the delay in starting the execution of a task after the execution of all the tasks on which this task depends has finished. For workflow applications, the waiting time is also indirectly optimized since it impacts the makespan. Finally, in the context of *Cloud services*, the *response time* to user's requests is usually optimized. This delay may be due to the overload of the application that needs to be scaled. Both for workflow applications and independent tasks, the response time is considered as the sum of the waiting and execution times.

Secondly are the objectives associated with the *economic cost*. From the perspective of IaaS users, it is of special interest to reduce the economic cost associated with the use of VMs (referred to *VM cost*). This cost is determined by the pay-per-use schemes defined by public Cloud providers. Each instance type has an assigned price (usually per hour of use) according to its computational performance. In this sense, the optimization can be achieved through more efficient use of the acquired infrastructure, reducing the computation time required to execute the application. It is also possible to reduce the execution cost with a dynamic and adequate selection of the VM types and the most convenient pricing models. For example, the use of spot or reserved instances represents an interesting opportunity. Then, from the perspective of public Cloud providers, it is important to reduce

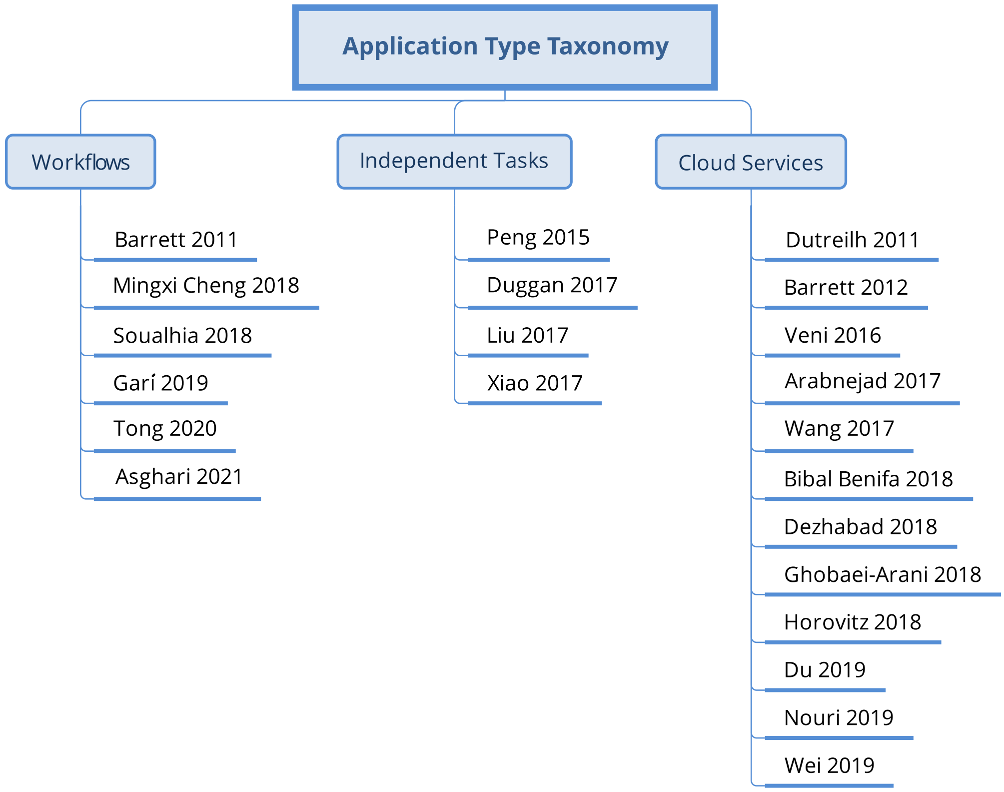

Fig. 5. Taxonomy of application types.

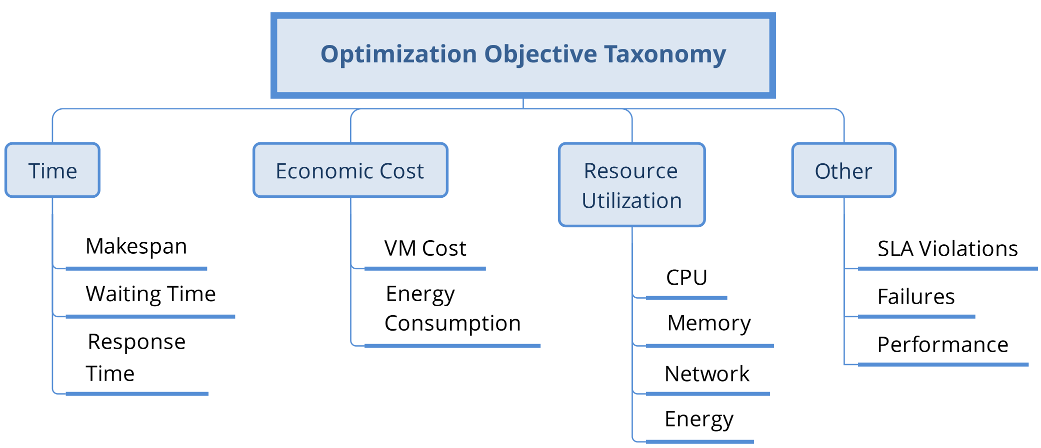

Fig. 6. Taxonomy of optimization objectives.

the costs associated with *energy consumption*, due to their environmental responsibility. The rapid growth of energy consumption and CO2 emission of Cloud infrastructures has become a key environmental concern (Garg et al., 2011; Beloglazov et al., 2011; Beloglazov and Buyya, 2012). Therefore, solutions focused on energy efficiency are required to ensure that the Cloud computing model is sustainable from an environmental perspective. On the other hand, by reducing expenses on electric bills, public Cloud providers can also increase their profit margins.

Third, an objective group common to all types of applications are those that describe the *resource utilization* degree. It is common to optimize the use of *CPU*, *memory,* and *network*. Also, in some cases, it is sought to reduce the *number of used VMs,* and consequently, the computational cost (memory and extra CPU) associated with their administration, and the *energy consumption.*

Another objective that is evidenced in the literature is to maximize the system's *performance*. Performance is usually defined considering the number of user requests served per second, being of special interest in service applications. For independent tasks and workflows applications, it is also important to minimize the *failures* of the tasks. Failures may be due to hardware or software problems in the infrastructure, or failures associated with the use of unreliable VM instances (spots or preemptible ones). Finally, the *SLA Violations* are considered. In many surveyed works, the objective of minimizing the number of SLA violations agreed between the service provider and the users are present.

3.4.4. Comparative analysis

Table 2 shows a summary of the analyzed works concerning the taxonomies defined above. First, the works in terms of the applied

RL technique (RL Technique column, see Fig. 3) are classified. Furthermore, the RL algorithms used in each case (RL Algorithm column) is shown. Then, the works based on the specific addressed problem (Problem column, see Fig. 4), the Optimization Objectives (Objectives column, see Fig. 6), and the Application Type (Application column, see Fig. 5) are classified, respectively. The last column (Experiments) specifies whether the experiments were performed in a simulated or a real Cloud.

From the analysis of the characteristics of the surveyed works in Table 2, the following observations are highlighted.

Regarding the applied *RL techniques*, most of the works (20/22) have proposed solutions in the Model-free category and only two works in the Model-based category (Garí et al., 2019; Barrett et al., 2011), which is convenient in a context where changes in the environment dynamics are likely to occur. In this sense, the proposals for *online* learning seem to be more adequate because they can adapt to these changes. Besides, there are Cloud autoscaling proposals that attempt to address classical problems inherent to RL, such as (a) managing large state spaces (Veni and Bhanu, 2016; Arabnejad et al., 2017; Liu et al., 2017; Wang et al., 2017; Mingxi Cheng and Nazarian, 2018), (b) the reduction of training time (Barrett et al., 2012; Bibal Benifa and Dejey, 2018; Nouri et al., 2019), (c) the poor initial performance (Dutreilh and Kirgizov, 2011) and (d) the slow convergence (Dutreilh and Kirgizov, 2011).

Regarding the *RL algorithms* (particularly in the 20 works in the Model-free category), *Q-learning* (15/20) predominates over *SARSA* (3/20). Note also that two works are using both algorithms, i.e., Q-learning and SARSA. However, in most cases, the selection of *Q-learning* or *SARSA* is not much argued. It is interesting to note that one of the works (Soualhia et al., 2018) proposes a combined solution that begins learning with *SARSA* for further exploration in the policy space and then continues with *Q-learning* to ensure a more direct convergence towards an appropriate policy. On the other hand, in Arabnejad et al. (2017) the authors combine the use of FL with RL. In this work, modified versions of both algorithms (Fuzzy-*Q-learning* and *Fuzzy-SARSA*) are used, but no significant performance differences are obtained.

Regarding the *problem*, it can be seen that both scaling (13/22) and scheduling (9/22) have been the subject of study from the RL perspective. For the scaling problem, most of the works focus on horizontal scaling, which means a niche area to explore, i.e. vertical scaling via RL. No proposals that solve both problems together (scaling and scheduling) from the perspective of RL have been proposed.

Regarding the *optimization objectives*, in most works, the optimization of multiple objectives is pursued. Concretely, most of the surveyed works have proposed reducing time and costs, achieving a more balanced resource usage, improving performance, and reducing failures as well as the violation of restrictions.

Regarding the *application type*, it is observed that service applications are more associated with the scaling problem. This may be related to the nature of these types of applications, which need to be able to scale to the fluctuating demands generated by multiple user requests while maintaining adequate performance and minimizing cost. On the other hand, both independent tasks and workflows applications are more associated with the scheduling problem, assuming a fixed infrastructure. For this type of applications, usually intensive in terms of computation power or data processing, it is intended to distribute the execution of the tasks in the available resources to maximize efficiency in terms of time, cost, and use of resources. In this sense, it should be noted that five proposals consider workflows per se or the above defined special type of workflow composed of independent tasks (four works). Therefore, workflows are a type of application that is still to be further explored in the area.

4. Discussion

This section analyzes the limitations and scope of the previously surveyed related works. First, the limitations regarding the type of addressed problem, the type of application, and the optimization objectives are analyzed. Then, other more theoretical limitations related to the RL techniques used in the proposals are discussed. Finally, open problems and ongoing developments in the area are also discussed.

4.1. Limitations related to the autoscaling problem formulation, application type, and addressed objectives

«*Scaling and scheduling issues are addressed independently*». There are no works that aim to solve both issues together from the RL perspective. Especially for workflows applications and independent tasks, both problems are interrelated and impact the execution efficiency, so it is important to design proposals that address both problems. Since the decisions regarding scaling and scheduling are different (for scaling, the actions are related to the reconfiguration of the infrastructure and for scheduling, the actions respond to where and/or when the tasks will be executed) it could be necessary to define different models and/or the combination of different techniques to address both problems.

«*Mix of works with workflow applications*». Among the 21 analyzed works, we should note that few proposals consider pure workflow applications (five works) or independent task applications (four works). Besides, most of these works focus only on scheduling and not scaling, except for the work in Garí et al. (2019). However, the approach in Garí et al. (2019) does not perform scaling purely with RL since it also uses a heuristic-based autoscaling strategy. Most scaling proposals are still based on service applications. It is important to note that there is a difference in the nature of workflow applications (long-term tasks, data-intensive or compute-intensive tasks, high-parallelism, or bottleneck stages) compared to service applications (generally short tasks under high demand peaks). Then, the strategies designed for service applications are not the most suitable ones for workflows, since they try to optimize processes of a different nature. In this sense, it is necessary to expand the study of RL techniques in the context of scaling for the efficient execution of workflows in Clouds.

«*For the scaling problem, the particular characteristics of the application about the type of required VM are not considered*». Particularly, most of the works define the characteristics of the environment based mainly on the state of the infrastructure, the number of available VMs or the resource utilization degree, and so on. Some works include information regarding the state of the application execution for evaluating the workload. Besides, several authors use homogeneous infrastructures (Dutreilh and Kirgizov, 2011; Mingxi Cheng and Nazarian, 2018). Although using homogeneous infrastructures is a common choice in HPC on the Cloud, in many cases, using heterogeneous infrastructures leads to better time and cost optimizations. From the works where the authors have considered the use of heterogeneous infrastructures (Ghobaei-Arani et al., 2018; Bibal Benifa and Dejey, 2018; Barrett et al., 2012), only one work (Bibal Benifa and Dejey, 2018) represents in the actions of the model the selection of different types of VMs. The surveyed works mostly target service applications, where the characteristics of acquired VMs may not be relevant. Conversely, for workflow and independent-task applications, the characteristics of acquired VMs are very important since they are usually composed of long-duration tasks that are intensive in computation/data. Therefore, it is necessary to consider the requirements of the tasks in terms of CPU, memory and data transfer, to determine the type of VM where task execution is viable and more efficient.

«*Different price models are not considered*». Most of the work in the area focuses on the classical payment scheme, where the provider defines a fixed price for each VM type charged hourly. Cloud flexibility is also found in the different options that providers offer in terms of price models. For example, Amazon spot instances, whose price fluctuates according to existing demand, have significant cost reductions (up to 90% in some cases) compared to the fixed price model. Considering that the economic cost is one of the main objectives of optimization in this type of problem, and it is also usually present in the SLA, it is interesting to exploit the Cloud options in terms of the different price models.

Table 2
Comparative summary of the relevant works in the literature that apply RL techniques for Cloud autoscaling.

Reference	RL technique	RL algorithm	Problem	Objective(s)	Application type	Experiments
Garí et al. (2019)	Model-based	*Value Iteration*	Scaling (H)	*Time, Cost*	Workflow	Simulated
Barrett et al. (2011)	Model-based	*Value Iteration*	Scheduling	*Time, Cost*	Workflow	Simulated
Peng et al. (2015)	Model-free, Pure-Seq.	*Q-learning*	Scheduling	*Time*	Indep. Tasks	Simulated
Xiao et al. (2017)	Model-free, Pure-Seq.	*Q-learning*	Scheduling	*Time*	Indep. Tasks	Simulated
Duggan et al. (2017)	Model-free, Pure-Seq.	*Q-learning*	Scheduling	*Time, Res. Utiliz.*	Indep. Tasks	Simulated
Soualhia et al. (2018)	Model-free, Pure-Seq.	*Q-learning* & *SARSA*	Scheduling	*Failures*	Workflow	Real
Dutreilh and Kirgizov (2011)	Model-free, Pure-Seq.	*Q-learning*	Scaling (H)	*Cost, SLA*	Services	Simulated
Ghobaei-Arani et al. (2018)	Model-free, Pure-Seq.	*Q-learning*	Scaling (H)	*Cost, SLA*	Services	Simulated
Dezhabad and Sharifian (2018)	Model-free, Pure-Seq.	*Q-learning*	Scaling (H)	*Res. Utiliz., SLA*	Services	Simulated
Horovitz and Arian (2018)	Model-free, Pure-Seq.	*Q-learning*	Scaling (H)	*Res. Utiliz., SLA*	Services	Simulated+Real
Wei et al. (2019)	Model-free, Pure-Seq.	*Q-learning*	Scaling (H)	*Time, Perf.*	Services	Simulated
Bibal Benifa and Dejey (2018)	Model-free, Pure-Parallel	*SARSA*	Scaling (H)	*Time, Res. Utiliz., Perf., SLA*	Services	Real
Asghari et al. (2021)	Model-free, Pure-Parallel	*SARSA*	Scheduling	*Time, Res. Utiliz., Cost, Energy*	Workflow	Simulated
Barrett et al. (2012)	Model-free, Pure-Parallel	*Q-learning*	Scaling (H)	*Res. Utiliz., SLA*	Services	Simulated
Nouri et al. (2019)	Model-free, Pure-Parallel	*Q-learning*	Scaling (H)	*Cost, SLA*	Services	Real
Arabnejad et al. (2017)	Model-free, FRL	*Q-learning* & *SARSA*	Scaling (H)	*Time, Res. Utiliz., SLA*	Services	Real
Veni and Bhanu (2016)	Model-free, FRL	*SARSA*	Scaling (V)	*Time, Res. Utiliz., Perf., SLA*	Services	Real
Wang et al. (2017)	Model-free, DRL	*Deep Q-learning*	Scaling (H)	*Cost, Res. Utiliz.*	Services	Simulated+Real
Liu et al. (2017)	Model-free, DRL	*Deep Q-learning*	Scheduling	*Time, Res. Utiliz.*	Indep. Tasks	Simulated
Mingxi Cheng and Nazarian (2018)	Model-free, DRL	*Deep Q-learning*	Scheduling	*Cost*	Workflow	Simulated
Tong et al. (2020)	Model-free, DRL	*Deep Q-learning*	Scheduling	*Time, Res. Utiliz.*	Workflow	Simulated+Real
Du et al. (2019)	Model-free, DRL	*Deep Q-learning*	Scheduling	*Cost*	Services	Real

4.2. Limitations related to the RL techniques

Model-based techniques: «A perfect model of the environment is required». This is one of the main limitations of the Model-based methods since in many problems the actual distribution of the transition probabilities between the states is unknown (Sutton and Barto, 2018). Even in dynamic environments, these probabilities could change over time. Estimates of the function $P(s,a)$ are usually used, but it is necessary to consider that the quality of the obtained policy depends directly on the quality of these estimates. In the works surveyed in this category (Barrett et al., 2011; Garí et al., 2019), to obtain an estimation of $P(s,a)$, the information generated by multiple previous executions of Cloud applications is used. Although major public Cloud providers (Amazon, Microsoft, and Google) have access to a large amount of information regarding executions, and such information could be used to generate this type of estimated models (Garí et al., 2019), complexity must be considered for determining the type and amount of information that should be used to avoid the classic problems of over-fitting and under-fitting when approximating functions.

Model-based techniques: «Offline policies might no longer be adequate due to changes in the dynamics of the environment». The fact that in Model-based approaches the policy is learned *offline* from a predefined fixed model does not allow it to adjust to changes in the dynamics of the environment. In the context of Cloud, a change in instance prices represents a possible cause of variation in the environment dynamics. A clear example of this is the price fluctuations of spot instances. When the execution cost is considered as an optimization objective, the previously computed policy could no longer be adequate since the learning of the environment that the policy represents has become obsolete. If the model continues to be updated and a new policy is recomputed every certain period, the resources (like time or capacity) required for such computation in an online context should also be considered. Also, if possible, it is necessary to determine the periodicity with which to perform such updates. In this line, the approaches in Barrett et al. (2011) and Garí et al. (2019) present this limitation because the policies are learned in *offline* mode.

Model-based and Model-free techniques: «Difficulty to manage large state spaces». This limitation generally affects both Model-based and Model-free methods. In the first case, the computational complexity of the algorithms like Value Iteration and Policy Iteration is polynomial in the number of states and actions defined. From the two analyzed works (Garí et al., 2019; Barrett et al., 2011) in the Model-based category, it can be seen that the state space and actions are limited. On the other hand, in the works based on Model-free methods (Peng et al., 2015; Xiao et al., 2017; Duggan et al., 2017; Soualhia et al., 2018; Dutreilh and Kirgizov, 2011; Ghobaei-Arani et al., 2018; Dezhabad and Sharifian, 2018; Bibal Benifa and Dejey, 2018; Barrett et al., 2012; Horovitz and Arian, 2018; Wei et al., 2019), the problem of managing many states and actions is associated not only with the requirements to store the function $Q(s,a)$ but also with the time and amount of data needed to update it. For example, the use of many features (or dimensions) generates a combinatorial explosion of states that is very difficult to handle. Besides, when scalability is required in some of the defined variables, the number of states increases considerably depending on the possible values of those variables. In this sense, since the problem of the dimension of the state space (also known as the dimensionality problem) is one of the main limitations of RL, different alternatives have been studied to try to mitigate its effects. An alternative to this problem (Peng et al., 2015; Xiao et al., 2017) consists of the *aggregation of states* by defining specific ranges of values for the variables that define them, grouping similar states. A second alternative is to combine RL with *function approximation*, a kind of generalization from supervised learning. An example is non-linear approximations of $Q(s,a)$, as in the proposals that use deep neural networks (Liu et al., 2017; Mingxi Cheng and Nazarian, 2018; Tong et al., 2020; Du et al., 2019). However, RL with function approximation has not yet been fully exploited in the area of Cloud autoscaling. A third option is the use of fuzzy logic combined with RL (Veni and Bhanu, 2016; Arabnejad et al., 2017), for evolving rules that enable approximate reasoning.

Model-free techniques: «Slow convergence». In the basic variants of Model-free methods, the function $Q(s,a)$ is updated when an action is executed, but only the visited state value at that time is updated. Although convergence is guaranteed, this usually involves a long training time, especially when it comes to problems with many states and/or many actions. To reduce training time, there are proposals (Barrett et al., 2012; Bibal Benifa and Dejey, 2018; Nouri et al., 2019) with multiple agents that learn in parallel and share the obtained information. On the other hand, in Dutreilh and Kirgizov (2011) the authors propose accelerating convergence using ideas from Dynamic Programming. For this, frequent phases of updating the function $Q(s,a)$ are defined using estimations of the obtained state values by recording the observations made of the visited states, transitions, and rewards.

Model-free techniques: «Poor initial performance». Considering that there is not an adequate policy at the beginning of the learning process (cold start effect), the initial performance of the strategy is usually poor and it will be improved as it converges to an appropriate policy. From all surveyed works, only in Dutreilh and Kirgizov (2011) we found a proposal to address this problem using an initial approximation of $Q(s,a)$.

4.3. Open possibilities

RL has demonstrated a great potential for automatically solving decision-making problems, particularly because of their ability to consider the long-term consequences of the available actions. Some of the most impressive results have been shown in Game Theory (Silver et al., 2016; Mnih et al., 2015), but RL's potential can be also extended to many other areas. Specifically, in Cloud autoscaling, only the first steps have been taken, and much remains to be done. From the analysis of the State of the Art, it becomes evident that there is a long list of current limitations, which means that there is a wide spectrum of research opportunities regarding RL techniques in the area of Cloud autoscaling. In the general sense, there is a wide number of unexplored combinations derived from the taxonomies outlined earlier in Section 3.4.

It would be interesting to design and develop autoscaling strategies for scientific applications in Cloud that combine a scheduler and scaler, both based on RL. These strategies could be based on the learning of appropriate scheduling and scaling policies, which allow dealing with the inherent uncertainty in executing applications in the Cloud. Besides, when policies are learned in an online mode, they would be able to adapt to changes in the dynamics of the environment. In the context of Cloud application execution, the uncertainty comes from the variability in the performance of the Cloud infrastructure. Also, the changes in the environment may be due to adjustments in the instances prices (as resource prices depend on market-like fluctuations) and even due to the appearance of other types of instances with different performance-cost trade-offs. The scaling policy could try to adjust the infrastructure dynamically according to the variable demand of the application, while the scheduling policy could determine the most appropriate resource for the execution of each task, considering the characteristics of each task and the available infrastructure. Both policies would be learned from experience in the interaction with the Cloud environment, modifying it and observing the effects.

Regarding the learning process, *parallel learning* is a topic that deserves much more attention. Parallel learning schemes update the Q values in parallel, speeding up the process of policy learning. In real Cloud settings, this kind of scheme might have particular importance since multiple autoscaling agents could share the feedback derived from their actions and update the Q-values collectively. From a theoretical point of view, this accelerates policy convergence but also allows that an enormous amount of agents operate as feedback collectors while at the same time are benefiting from the latest Q updates making the information instantly available to all of the agents.

Nowadays, *one-step*, *tabular*, *model-free* TD is the most widely used RL method. This is probably due to their great simplicity. Still, these algorithms can be extended, making them slightly more complicated and significantly more powerful (i.e., multi-step forms, various forms of function approximation rather than tables, etc.) (Sutton and Barto, 2018). In the area of Cloud autoscaling, the majority of approaches still use the simplest variants of RL methods. Therefore, there is still room for further investigating the synergy of different variants of the basic RL strategies and other machine learning methods. In fact, 20% of the surveyed works, which have been published in 2017–2020, have exploited deep neural networks, which shows a trend in this line.

Also, the particular states, actions, and how they are represented can strongly affect the performance of the implemented approach. In RL, as in other areas of Machine Learning, such representational choices are, nowadays, more an art than a science (Sutton and Barto, 2018). In the area of Cloud autoscaling, it is fundamental to study the specific implications of such representational choices (states and actions) and how they impact the performance of autoscalers. For example, interesting questions to answer in this matter are: (1) What information from a real Cloud environment is relevant to learn a policy properly? (b) What could be an adequate representation of this information to accelerate the learning process?

On the other side, the problem of Cloud autoscaling is closely related to Multi-objective Optimization (MOO). The reader might have noticed that in almost all surveyed papers, multiple optimization objectives are present. Even more, conflicting objectives (such as economic cost and execution time) are common in this context. Current proposals usually combine these objectives in the reward function to optimize all of them simultaneously. However, many other possibilities are investigated in the active area of research called Multi-objective Reinforcement Learning (MORL) (Liu et al., 2015), which combines the concepts and strengths of such two important fields: MOO and RL. Needless to say that the study of MORL techniques in Cloud autoscaling is a fundamental future line of research, yet it is incipient.

5. Conclusions

The flexibility and elasticity offered by the Cloud Computing paradigm have opened opportunities to the study of autoscaling strategies for the efficient execution of workflows, independent tasks, and Cloud service applications. However, the variability in Cloud performance generates an important uncertainty factor when making scaling or scheduling decisions during application execution. In this sense, RL-based strategies allow autoscalers to learn appropriate policies through interaction with a stochastic environment. In this context, recent research is focused on the exploitation of RL-based strategies to address the autoscaling subproblems, i.e. scaling and scheduling.

Motivated by these facts, we have surveyed and classified works in this area by deriving a taxonomy according to the type of RL-based technique used. On the first level of the taxonomy, proposals in the Model-based and the Model-free categories are presented. Then, on a second level, proposals in the Model-free category are classified into three groups. First, are those proposals that apply the technique in its original or pure formulation. These techniques are further subdivided into sequential or parallel, since the variant of RL is given by parallel learning. Second, we present the proposals that combine RL with neural networks, and finally, the proposals that combine RL with Fuzzy Logic (FL).

As evidenced in the analysis of the reviewed literature, algorithms based on RL such as *Q-learning* and *SARSA* have shown to be effective in the online learning of scaling and scheduling policies in the Cloud. A 45% of the surveyed works are based on the autoscaling problem in Cloud for workflow and independent tasks applications, which are applications with distinctive features (long-term tasks, data-intensive or computational-intensive tasks, high workload variability with high parallelism and bottleneck stages) mostly used in engineering and scientific settings, while a 55% of the works focus on Cloud service applications, mostly used in e-commerce and business settings. However, a major finding is that neither of the surveyed works proposes a solution covering both the scaling and scheduling problems. Hence, the inception of full-fledged autoscalers purely based on RL techniques for either type of Cloud application remains to be seen in the area.

As a final comment, it is important to note that RL is a key technology for the future development of Distributed Computing systems, even beyond Cloud Computing. In particular, through RL it would be possible to develop autonomous infrastructure management platforms that meet:

- Transparency: Implementation and operation details of the applications would be hidden to the user. The use of these systems would not depend on human intervention, nor would demand to have access to deep domain knowledge since it is expected that scaling and planning policies are learned through the interaction with the environment. Such a scenario would be different from the actual one in which, for example, the scaling approaches that public computing infrastructures (Amazon, 2020) use are based on explicit thresholds for resource use. Such thresholds are usually defined by experts based on the available metrics such as CPU or memory usage.

- Dynamism: At any moment, learned policies would allow the provider to take the necessary actions given the current state of the environment and the state of the applications. In such a scenario, the system would not have to rely on static plans nor rule-based actions defined manually.
- Adaptability: Thanks to online learning, policies can be constantly improved and updated. In such a way, the policies would be able to adapt to the changes that occur in the dynamics of the execution environment. Such a characteristic is fundamental compared with policies learned in offline mode (Garí et al., 2019) that are prone to become obsolete in time.

Although the potential benefits are evident, many efforts are still necessary towards making these goals a reality.

CRediT authorship contribution statement

Yisel Garí: Writing - original draft, Writing - review & editing, Investigation. **David A. Monge:** Writing - review & editing, Investigation, Conceptualization. **Elina Pacini:** Writing - review & editing, Investigation, Funding acquisition. **Cristian Mateos:** Writing - review & editing, Supervision, Funding acquisition, Conceptualization. **Carlos García Garino:** Supervision, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We acknowledge the financial support by CONICET, Argentina, grant number PIP 2017–2019 GI 11220170100490CO, and the SIIP-UNCuyo, Argentina project No. B082 and 06/B369.

Appendix

A.1. MDP resolution via dynamic programming

Dynamic programming (DP) in this context refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as an MDP. Methods based on DP compute the policy based on a complete model of the environment (Model-based). In the process of estimating the state values $V(s)$, the probability distribution of the transitions between the states $P_a(s,s')$ is used. This often becomes a limitation since it is not always possible to derive such a model. In some cases, the probability distribution of the transitions is estimated from data obtained from previous experiences. The DP methods offer an *offline* learning variant, where the policy is obtained by iterating over the model and not based on the dynamics of current experiences. It is important to note that prior estimates of other states are used to estimate the values of the states (a technique known as *bootstrap*). Two widely used DP algorithms are *policyIteration* and *valueIteration*. Both algorithms have polynomial complexity in the number of states and actions, so it is important to consider the dimensions of these spaces when using DP. However, the search performed with DP is much more efficient than an exhaustive exploration in the space of all possible policies.

The *policyIteration* algorithm (see Algorithm 1) is defined based on the iterative repetition of the evaluation and the improvement of the policy until convergence is achieved. In this way, the algorithm generates the following sequence of value functions and policies $v_0 \rightarrow \pi_0 \rightarrow v_1 \rightarrow \pi_1 \ldots \rightarrow \pi^*$ until to reach an appropriate policy. On the other hand, the *valueIteration* algorithm (see Algorithm 2), first includes the search for the appropriate value function and then, the computation of the associated policy. These steps are not repeated because once the value function is adequate, so is the associated policy. The search for the appropriate value function can be understood as a combination of the policy improvement process and a truncated evaluation of the policy (the values are reassigned after a single sweep of the states) without losing convergence (Sutton and Barto, 2018). In this way, the algorithm generates the sequence of value function updates $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v^*$ and then, it computes the suitable policy π^*.

Both algorithms, *policyIteration* and *valueIteration*, formally require an infinite number of iterations to converge exactly to the appropriate policy. In practice, both algorithms stop when the difference between two successive approximations is less than a limit Θ, which usually within a much lesser number of iterations (Sutton and Barto, 2018).

Algorithm 1 The Policy Iteration algorithm.

procedure PolicyIteration($S, A, P, R, \gamma, \Theta$):
  1.Initialize $V(s)$ y $\pi(s)$ arbitrarily $\forall s \in S$
  2.Policy Evaluation
  repeat:
    $\Delta \leftarrow 0$
    for each $s \in S$ **do**
      $v \leftarrow V(s)$
      $a \leftarrow \pi(s)$
      $V(s) \leftarrow \sum_{s'} P_a(s,s')[R_a(s,s') + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  until $\Delta < \Theta$ (a small positive number)
  3.Policy Improvement
  $stablePolicy \leftarrow true$
  for each $s \in S$ **do**
    $oldAction \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_a(s,s')[R_a(s,s') + \gamma V(s')]$
    If $oldAction \neq \pi(s)$ **then** $stablePolicy \leftarrow false$
  If $stablePolicy$ **then** stop and return $V \approx v*$ and $\pi \approx \pi*$

Algorithm 2 The Value Iteration algorithm.

procedure ValueIteration($S, A, P, R, \gamma, \Theta$):
  Initialize array V arbitrarily $\forall s \in S$
  repeat:
    $\Delta \leftarrow 0$
    for each $s \in S$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \max_a \sum_{s'} P_a(s,s')[R_a(s,s') + \gamma V(s')]$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  until $\Delta < \Theta$ (a small positive number)
  Output a deterministic policy $\pi \approx \pi*$ such that:
  $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_a(s,s')[R_a(s,s') + \gamma V(s')]$

A.2. MDP resolution via temporal difference

Methods based on Temporal Difference (TD) do not require a perfect model of the environment (Model-free), since the policy learning process is based on the observed dynamics during its experimentation. In this sense, these methods offer an approach with a greater ability to adapt to changes in the environment since, unlike DP-based methods, learning through TD occurs in an *online* way. Because of the lack of the model, the state value function $V(s)$ is not sufficient in suggesting a policy and it is required to estimate the values related to each action. The action-value function $Q(s,a)$ represents the expected gain considering the state–action pair, and it is usually represented in tabular form. Similarly to DP, to estimate new values, the previously estimated values are used (i.e., bootstrap is performed).

Two widely used algorithms in this area are *Q-learning* (Watkins and Dayan, 1992) and *State-Action-Reward-State-Action* (*SARSA*). It is important to highlight that one of the main limitations in RL is that the convergence time of these algorithms depends directly on the dimension of the state space and actions. Moreover, since these algorithms do not have an adequate initial policy they have a poor initial performance that will have a greater or lesser impact depending on the addressed problem and the time taken for training. Making inappropriate decisions at the beginning of the autoscaling of workflows in Cloud, which is necessary for the exploration process, can directly impact the makespan and the economic cost, so it is convenient to have a good initial policy. This could also reduce the time required to learn the right policy. In any case, it is necessary to consider that obtaining an acceptable initial policy is not always trivial (Dann et al., 2014).

Algorithm 3 The Q-learning algorithm.

procedure Q-LEARNING($S, A, P, R, \gamma, \alpha, \varepsilon$):
  Initialize $Q(s,a)$ arbitrarily $\forall s \in S, a \in A$ y $Q(terminalState, .) = 0$
  for each (episode) **do**
    Initialize S
    repeat:
      Select A from S using the policy derived from de Q (ε-greedy)
      Take action A, observe R,S'
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
      $S \leftarrow S'$
    until S is terminal

Algorithm 4 The SARSA algorithm.

procedure SARSA($S, A, P, R, \gamma, \alpha, \varepsilon$):
  Initialize $Q(s,a)$ arbitrarily $\forall s \in S, a \in A$ y $Q(terminalState, .) = 0$
  for each episode **do**
    Initialize S
    Select A from S using the policy derived from Q (ε-greedy)
    repeat:
      Take action A, observe R,S'
      Select A' from S' using the policy derived from Q (ε-greedy)
      $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
      $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

The distinctive characteristic of *Q-learning* (see Algorithm 3) is that it uses two different policies, one to select the next action and another to update Q. In other words, *Q-learning* tries to evaluate π while following another policy μ. Alternatively, SARSA (see Algorithm 4) uses the same policy all the time. The most important difference between the two above mentioned algorithms is how Q is updated after each action. *Q-learning* updates Q with the action that maximizes the gain for the next step. This makes *Q-learning* follows an *ε-greedy* policy[9] with $\varepsilon = 0$, i.e., there is no exploration. In contrast, SARSA updates Q by following exactly an *ε-greedy* policy, since the action is extracted from it. Both algorithms include the $\alpha \in (0,1]$ parameter relative to the size of the step in the learning process, and the $\varepsilon > 0$ parameter that determines the exploration degree of new policies.

A.3. MDP resolution via neural networks

Large state spaces in an RL problem lead to the need to find non-tabular representations of the Q function, not only for the memory required to store large tables but also for the time it would take to fill it. Algorithms capable of generalizing in more complex and sophisticated state space contexts are then consequently needed. In this sense, non-linear approximations of Q with artificial neural networks appeared. A type of neural network that has proven very successful in RL applications (Mnih et al., 2015; Silver et al., 2016) is Deep Convolutional Neural Networks, which are specialized in the processing of large-scale data organized in spatial matrices. Then, the strategy that combines RL with Deep Neural Networks (DNN) is called Deep Reinforcement Learning (DRL). Finally, the DNN used to approximate the Q function is called *DeepQNetworks* (DQN), and the learning algorithm that uses DQN is referred to as *Deep Q-learning*.

Fig. 7 shows an example of a DQN. The input corresponds to a state s of the environment and the output represents the estimated value of function Q for the state s and all possible actions. In training the network, the objective is to minimize the approximation error between the result of the network and the optimality equation of Bellman (Mnih et al., 2015). Thereby, the same problem as with the classical DP and TD techniques is solved, but now using a non-linear approach based on deep neural networks.

A.4. MDP resolution via fuzzy logic

Fuzzy Logic (FL) appears as another alternative to address the dimensionality problem of RL strategies. The idea is to reduce the state space using a diffuse representation of the information.

Broadly, FL systems attempt to represent knowledge inaccurately, similar to how human beings do, and as opposed to classical numerical forms. In this sense, FL works with fuzzy sets in which the elements have some membership degree. To define the membership degree of these sets, triangles or trapezoid curves are usually used (see Fig. 8). For example, in Fig. 8, a fuzzy membership function is represented for a *Cloud workload* variable with three fuzzy sets (Low, Medium, High) that define the membership degree of the variable to each of them. Thus, in the presence of a workload α, it is possible to affirm that it belongs both to the Low and Medium fuzzy sets with a 50% probability, and hence the diffuse nature of this representation.

These concepts from FL allow reasoning based on rules of the form:

$$\text{if}(antecedent)\text{then}(consequent),$$

where the antecedent and consequent values are expressed in a fuzzy way. Based on the previous example, one possible rule is: if the workload is *high* then *more VMs* must be allocated.

FL has been applied in different fields, from Control Theory to Artificial Intelligence. A control process based on FL consists of the following steps:

- Mapping of input data to fuzzy set labels (Fuzzifier)
- The inference process based on fuzzy rules (Fuzzy Reasoning)
- Fuzzy output mapping to clear values (Defuzzifier)

Fuzzy Reinforcement Learning (FRL) is the strategy that combines the strength of fuzzy reasoning with RL. FRL allows handling problems with large state spaces without affecting the performance of the RL algorithms. For this, a fuzzy representation of the information is used, which considerably reduces the number of states.

Motivated by this benefit, some authors (Arabnejad et al., 2017; Veni and Bhanu, 2016) have proposed approaches based on FRL for autoscaling in Cloud. Fig. 9 shows the interaction between the components involved in these approaches. A monitoring process continuously observes the Cloud platform and the running application that composes the environment. The monitoring process retrieves data of interest in the state of the environment and reports it to the Automatic Controller (AC). One of the main components of the AC is the FL-based control process called Fuzzy Controller (FC). The FC is composed of the Knowledge Base (or rules), the Fuzzifier, the Inference Engine, and the Defuzzifier. In this way, the FC receives the signal of the environment

[9] *ε-greedy*: a policy that with an ε probability selects a random action, but most of the time it selects an action with the maximum estimated value.

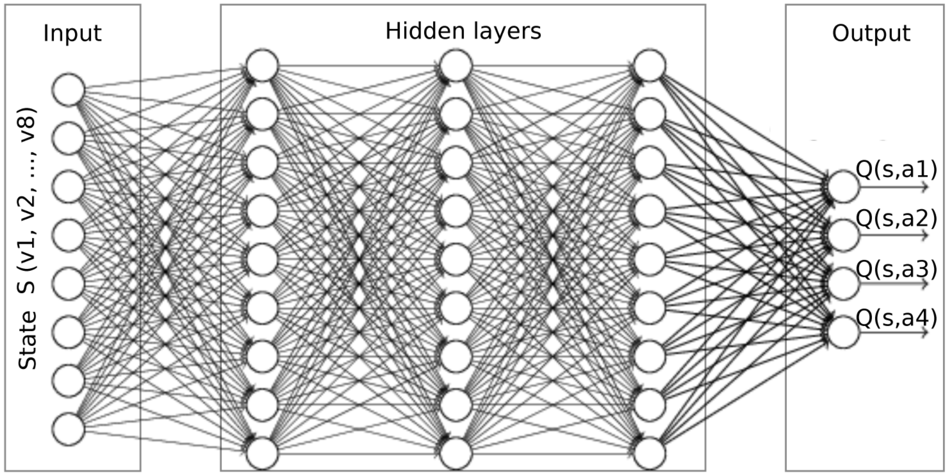

Fig. 7. Example of the structure of a DQN.

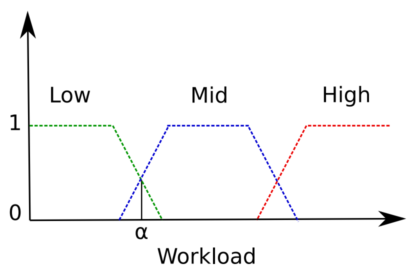

Fig. 8. Example of the fuzzy membership function (Y axis) for the workload variable. The fuzzy sets are defined using a trapezoid.

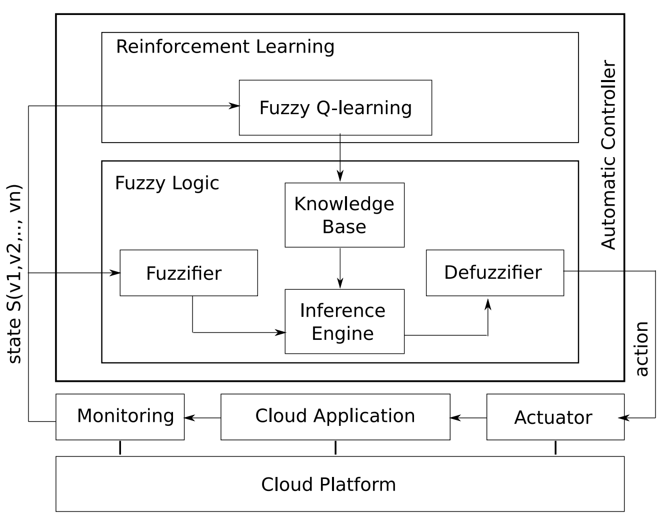

Fig. 9. Example of the architecture of a Cloud autoscaling system based on RL and FL.
Source: Figure adapted from Jamshidi et al. (2016).

state, transforms it to its diffuse representation, reasons based on the rules, and obtains a diffuse output that is finally returned to its clear representation. The *actuator process* uses this output or action to modify the environment. The second component of the FC is precisely the RL process, which also receives the signal of the environment state and, guided by the optimization objectives, is responsible for learning the most appropriate set of rules to update the knowledge base of FC. Each member of the table of values Q is assigned to a specific rule (which describes some action–state pairs). Then, these values are updated during the learning process. In this way, it is possible to take advantage of the strengths of RL and FL strategies to design an automatic controller capable of evolving fuzzy rules that allow making approximate reasoning.

References

Al-Ayyoub, Mahmoud, Jararweh, Yaser, Daraghmeh, Mustafa, Althebyan, Qutaibah, 2015. Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure. Cluster Comput. 18 (2), 919–932.

Alam, Md Golam Rabiul, Hassan, Mohammad, Uddin, Zla, Almogren, Ahmad, Fortino, Giancarlo, 2019. Autonomic computation offloading in mobile edge for IoT applications. Future Gener. Comput. Syst. 90, 149–157.

Amazon, 2020. Autoescaling. http://aws.amazon.com/autoscaling/, [Online; accessed May-2020].

Arabnejad, Hamid, Pahl, Claus, Jamshidi, Pooyan, Estrada, Giovani, 2017. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In: Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017. pp. 64–73.

Armbrust, Michael, Fox, Armando, Griffith, Rean, Joseph, Anthony D., Katz, Randy, Konwinski, Andy, Lee, Gunho, Patterson, David, Rabkin, Ariel, Stoica, Ion, Zaharia, Matei, 2009. Above the Clouds: A Berkeley View of Cloud Computing. Technical. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Arunarani, A.R., Manjula, D., Sugumaran, V., 2018. Task scheduling techniques in cloud computing: A literature survey. Future Gener. Comput. Syst. 91, 407–415.

Asghari, Ali, Sohrabi, Mohammad Karim, Yaghmaee, Farzin, 2021. Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm. J. Supercomput. 77 (3), 2800–2828.

Barrett, Enda, Howley, Enda, Duggan, Jim, 2011. A learning architecture for scheduling workflow applications in the cloud. In: Proceedings – 9th IEEE European Conference on Web Services, ECOWS 2011. pp. 83–90.

Barrett, Enda, Howley, Enda, Duggan, Jim, 2012. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. Concurr. Comput. Pract. Exp..

Bellman, Richard, 1957. Dynamic Programming. Princeton University Press, New Jersey.

Beloglazov, Anton, Buyya, Rajkumar, 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput.: Pract. Exper. 24 (13), 1397–1420.

Beloglazov, Anton, Buyya, Rajkumar, Lee, Young Choon, Zomaya, Albert, 2011. A taxonomy and survey of energy-efficient data centers and cloud computing systems. Advances in Computers, Vol. 82, first ed. Elsevier Inc., pp. 47–111.

Bibal Benifa, J.V., Dejey, D., 2018. RLPAS: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment. Mob. Netw. Appl. 1–16.

Brown, Duncan A., Brady, Patrick R., Dietz, Alexander, Cao, Junwei, Johnson, Ben, McNabb, John, 2007. A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. In: Workflows for E-Science: Scientific Workflows for Grids. Springer London, London, pp. 39–59.

Buyya, Rajkumar, Yeo, Chee Shin, Venugopal, Srikumar, Broberg, James, Brandic, Ivona, 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. 25 (6), 599–616.

Dann, Christoph, Neumann, Gerhard, Peters, Jan, 2014. Policy evaluation with temporal differences: A survey and comparison. J. Mach. Learn. Res. 15, 809–883.
Dezhabad, Naghmeh, Sharifian, Saeed, 2018. Learning-based dynamic scalable load-balanced firewall as a service in network function-virtualized cloud computing environments. J. Supercomput..
Dong, Ziqian, Liu, Ning, Rojas-Cessa, Roberto, 2015. Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers. J. Cloud Comput. 4 (1).
Du, Bingqian, Wu, Chuan, Huang, Zhiyi, 2019. Learning resource allocation and pricing for cloud profit maximization. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33. AAAI Press, Honolulu,Hawaii, USA, pp. 7570–7577.
Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N.C., Hu, B., 2015. Everything as a service (xaas) on the cloud: Origins, current and future trends. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 621–628.
Duggan, Martin, Duggan, Jim, Howley, Enda, Barrett, Enda, 2017. A network aware approach for the scheduling of virtual machine migration during peak loads. Cluster Comput. 20 (3), 2083–2094.
Dutreilh, Xavier, Kirgizov, Sergey, 2011. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In: 7th International Conference on Autonomic and Autonomous Systems. pp. 67–74.
Fabra, Javier, Ezpeleta, Joaquín, Álvarez, Pedro, 2019. Reducing the price of resource provisioning using EC2 spot instances with prediction models. Future Gener. Comput. Syst. 96, 348–367.
Fortino, Giancarlo, Di Fatta, Giuseppe, Pathan, Mukaddim, Vasilakos, Athanasios V., 2014a. Cloud-assisted body area networks: state-of-the-art and future challenges. Wirel. Netw. 20 (7), 1925–1938.
Fortino, G., Parisi, D., Pirrone, V., Di Fatta, G., 2014b. BodyCloud: A SaaS approach for community Body Sensor Networks. Future Gener. Comput. Syst. 35, 62–79.
García Garino, Carlos, Ribero Vairo, Melisa, Andía Fagés, Susana, Mirasso, Anibal, Ponthot, Jean-Phillip, 2013. Numerical simulation of finite strain viscoplastic problems. J. Comput. Appl. Math. 246, 174–184.
Garg, Saurabh Kumar, Yeo, Chee Shin, Buyya, Rajkumar, 2011. Green cloud framework for improving carbon efficiency of clouds. In: Jeannot, Emmanuel, Namyst, Raymond, Roman, Jean (Eds.), Euro-Par 2011 Parallel Processing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 491–502.
Garí, Yisel, Monge, David A., Mateos, Cristian, Garino, Carlos García, 2018. Markov Decision Process to Dynamically Adapt Spots Instances Ratio on the Autoscaling of Scientific Workflows in the Cloud. Springer International Publishing, Cham, pp. 353–369.
Garí, Yisel, Monge, David A., Mateos, Cristian, Garino, Carlos García, 2019. Learning budget assignment policies for autoscaling scientific workflows in the cloud. Cluster Comput..
Ghobaei-Arani, Mostafa, Jabbehdari, Sam, Pourmina, Mohammad Ali, 2018. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. Future Gener. Comput. Syst. 78, 191–210.
Glushkova, Daria, Jovanovic, Petar, Abelló, Alberto, 2019. Mapreduce performance model for hadoop 2.x. Inf. Syst. 79, 32–43, Special issue on DOLAP 2017: Design, Optimization, Languages and Analytical Processing of Big Data.
Gravina, Raffaele, Ma, Congcong, Pace, Pasquale, Aloi, Gianluca, Russo, Wilma, Li, Wenfeng, Fortino, Giancarlo, 2017. Cloud-based activity-aaservice cyber–physical framework for human activity monitoring in mobility. Future Gener. Comput. Syst. 75, 158–171.
Hochreiter, Sepp, Schmidhuber, Jürgen, 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.
Horovitz, Shair, Arian, Yain, 2018. Efficient cloud auto-scaling with SLA objective using Q-learning. In: 6th International Conference on Future Internet of Things and Cloud (FiCloud). IEEE, pp. 85–92.
Hu, Liqin, Long, Pengcheng, Song, Jing, He, Tao, Wang, Fang, Hao, Lijuan, Wu, Bin, Yu, Shengpeng, He, Peng, Wang, Lei, Sun, Guangyao, Zou, Jun, Gan, Quan, Sun, Guomin, Li, Zhongyang, Wu, Yican, 2019. Supermc cloud for nuclear design and safety evaluation. Ann. Nucl. Energy 134, 424–431.
Humane, Pravesh, Varshapriya, J.N., 2015. Simulation of cloud infrastructure using cloudsim simulator: A practical approach for researchers. In: 2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM). IEEE, pp. 207–211.
Jamshidi, Pooyan, Sharifloo, Amir, Pahl, Claus, Arabnejad, Hamid, Metzger, Andreas, Estrada, Giovani, 2016. Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In: Proceedings - 2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA 2016. pp. 70–79.
Jang, Joonhyouk, Jung, Jinman, Hong, Jiman, 2019. K-LZF : An efficient and fair scheduling for Edge computing servers. Future Gener. Comput. Syst. 98, 44–53.
Kaur, Shaminder, Verma, Amandeep, 2012. An efficient approach to genetic algorithm for task scheduling in cloud computing environment. Int. J. Inf. Technol. Comput. Sci. 4 (10), 74–79.
Kearney, Keven T., Torelli, Francesco, 2011. Security in service level agreements for cloud computing. In: Proceedings of the 1st International Conference on Cloud Computingand Services Science, (CLOSER). pp. 636–642.
Koh, Younggyun, Knauerhase, Rob, Brett, Paul, Bowman, Mic, Zhihua, Wen, Pu, Calton, 2007. An analysis of performance interference effects in virtual environments. In: 2007 IEEE International Symposium on Performance Analysis of Systems Software. pp. 200–209.
Krzywda, Jakub, Ali-Eldin, Ahmed, Carlson, Trevor E., Ostberg, Per-Plov, Elmroth, Erik, 2018. Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling. Future Gener. Comput. Syst. 81, 114–128.
Kumar, M., Mohit, S., Sharma, S.C., Goel, A., Singh, S.P., 2019. A comprehensive survey for scheduling techniques in cloud computing. J. Netw. Comput. Appl..
Lee, Kyong-Ha, Lee, Yoon-Joon, Choi, Hyunsik, Chung, Yon Dohn, Moon, Bongki, 2012. Parallel data processing with mapreduce: A survey. SIGMOD Rec. 40 (4), 11–20.
Li, Hongjia, Li, J., Yao, Wang, Nazarian, S., Lin, X., Wang, Y., 2017. Fast and energy-aware resource provisioning and task scheduling for cloud systems. In: 2017 18th International Symposium on Quality Electronic Design (ISQED). pp. 174–179.
Liu, Jia, Feld, Dustin, Xue, Yong, Garcke, Jochen, Soddemann, Thomas, Pan, Peiyuan, 2016. An efficient geosciences workflow on multi-core processors and GPUs: a case study for aerosol optical depth retrieval from MODIS satellite data. Int. J. Digit. Earth 9 (8), 748–765.
Liu, Ning, Li, Zhe, Xu, Zhiyuan, Xu, Jielong, Lin, Sheng, Qiu, Qinru, Tang, Jian, Wang, Yanzhi, 2017. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In: IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 866–876.
Liu, Chunming, Xu, Xin, Hu, Dewen, 2015. Multiobjective reinforcement learning: A comprehensive overview. IEEE Trans. Syst. Man Cybern. Syst. 45 (3), 385–398.
Makris, Nicos, 2003. Plastic torsional buckling of cruciform compression members. J. Eng. Mech. 129 (6), 689–696.
Mao, Ming, Humphrey, Marty, 2013. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. IEEE, pp. 67–78.
Marcus, Gary, 2018. Deep learning: A critical appraisal. arXiv e-prints, pages arXiv:1801.00631.
Mauch, Viktor, Kunze, Marcel, Hillenbrand, Marius, 2013. High performance cloud computing. Future Gener. Comput. Syst. 29 (6), 1408–1416, Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems.
Meade, Bernard F., Fluke, Christopher J., 2018. Evaluating virtual hosted desktops for graphics-intensive astronomy. Astron. Comput. 23, 124–140.
Mell, Peter, Grance, Timothy, Grance, Timothy, 2011. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. National Institute of Standards and Technology Special Publication 800-145 7.
Mingxi Cheng, Ji Li, Nazarian, Shahin, 2018. DRL-cloud : Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In: Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific. pp. 129–134.
Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, Hassabis, Demis, 2015. Human-level control through deep reinforcement learning. Nature 518, 529 EP –.
Monge, David A., Garí, Yisel, Mateos, Cristian, Garino, Carlos García, 2017. Autoscaling scientific workflows on the cloud by combining on-demand and spot instances. Int. J. Comput. Syst. Sci. Eng. 32 (4 Special Issue on Elastic Data Management in Cloud Systems).
Monge, David A., Pacini, Elina, Mateos, Cristian, Alba, Enrique, García Garino, Carlos, 2020. CMI: An online multi-objective genetic autoscaler for scientific and engineering workflows in cloud infrastructures with unreliable virtual machines. J. Netw. Comput. Appl. 149, 102464.
Monge, David A., Pacini, Elina, Mateos, Cristian, Garino, Carlos García, 2018. Meta-heuristic based autoscaling of cloud-based parameter sweep experments with unreliable virtual machines instances. Comput. Electr. Eng. 69, 364–377.
Nouri, Seyed Mohammad Reza, Li, Han, Venugopal, Srikumar, Guo, Wenxia, He, MingYun, Tian, Wenhong, 2019. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. Future Gener. Comput. Syst. 94, 765–780.
Oliveira de Carvalho, J., Trinta, F., Vieira, D., Carmona Cortes, O.A., 2018. Evolutionary solutions for resources management in multiple clouds: State-of-the-art and future directions. Future Gener. Comput. Syst. 88, 284–296.
Otterlo, Martijn Van, 2009. The Logic of Adaptive Behavior. In: Frontiers in Artificial Intelligence and Applications, vol. 192, IOS Press.
Pacini, Elina, Mateos, Cristian, García Garino, Carlos, 2014. Distributed job scheduling based on Swarm intelligence: A survey. Comput. Electr. Eng. 40 (1), 252–269, 40th-year commemorative issue.
Peng, Zhiping, Cui, Delong, Zuo, Jinglong, Li, Qirui, Xu, Bo, Lin, Weiwei, 2015. Random task scheduling scheme based on reinforcement learning in cloud computing. Cluster Comput. 18 (4), 1595–1607.
Pu, Xing, Liu, Ling, Mei, Yiduo, Sivathanu, Sankaran, Koh, Younggyun, Pu, Calton, 2010. Understanding performance interference of I/O workload in virtualized cloud environments. In: Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010. pp. 51–58.

Rao, Jia, Bu, Xiangping, Xu, Cheng-Zhong, Wang, Leyi, Yin, George, 2009. VCONF: A reinforcement learning approach to virtual machines auto-configuration. In: Proceedings of the 6th International Conference on Autonomic Computing. In: ICAC '09, ACM, New York, NY, USA, pp. 137–146.

Roy, Nilabja, Dubey, Abhishek, Gokhale, Aniruddha, 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In: Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011. pp. 500–507.

Silver, David, Huang, Aja, Maddison, Chris J., Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, Dieleman, Sander, Grewe, Dominik, Nham, John, Kalchbrenner, Nal, Sutskever, Ilya, Lillicrap, Timothy, Leach, Madeleine, Kavukcuoglu, Koray, Graepel, Thore, Hassabis, Demis, 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529 (7587), 484–489.

Singh, S., Chana, I., 2016. Cloud resource provisioning: survey, status and future research directions. Knowl. Inf. Syst. 49, 1005–1069.

Smith, Leslie N., 2018. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. arXiv e-prints, pages arXiv:1803.09820.

Soualhia, M., Khomh, F., Tahar, S., 2018. A dynamic and failure-aware task scheduling framework for Hadoop. IEEE Trans. Cloud Comput. 8 (2), 1–16.

Spinner, Simon, Kounev, Samuel, Zhu, Xiaoyun, Lu, Lei, Uysal, Mustafa, Holler, Anne, Griffith, Rean, 2014. Runtime vertical scaling of virtualized applications via online model estimation. In: International Conference on Self-Adaptive and Self-Organizing Systems, SASO, Vol. 2014-December(December). pp. 157–166.

Sutton, Richard S., Barto, Andrew G., 2018. Reinforcement Learning: An Introduction. A Bradford Book, USA.

Tang, Shaojie, Yuan, Jing, Li, Xiang Yang, 2012. Towards optimal bidding strategy for Amazon EC2 cloud spot instance. In: Proceedings – 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012. pp. 91–98.

Tong, Zhao, Chen, Hongjian, Deng, Xiaomei, Li, Kenli, Li, Keqin, 2020. A scheduling scheme in the cloud computing environment using deep Q-learning. Inform. Sci. 512, 1170–1191.

Vandenbrouck, Yves, Christiany, David, Combes, Florence, Loux, Valentin, Brun, Virginie, 2019. Bioinformatics tools and workflow to select blood biomarkers for early cancer diagnosis: an application to pancreatic cancer. Proteomics 19 (21–22), 1800489.

Veni, T., Bhanu, S. Mary Saira, 2016. Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach. Int. J. Big Data Intell. 3 (3).

Wang, Zhiguang, Gwon, Chul, Oates, Tim, Iezzi, Adam, 2017. Automated cloud provisioning on AWS using deep reinforcement learning. arXiv e-prints, pages arXiv:1709.04305.

Wang, Henry, Ma, Yunzhi, Pratx, Guillem, Xing, Lei, 2011. Toward real-time Monte Carlo simulation using a commercial cloud computing infrastructure. Phys. Med. Biol. 56 (17), N175–N181.

Wang, Ziyu, Schaul, Tom, Hessel, Matteo, van Hasselt, Hado, Lanctot, Marc, de Freitas, Nando, 2015. Dueling network architectures for deep reinforcement learning. arXiv e-prints, pages arXiv:1511.06581.

Watkins, Christopher J.C.H., Dayan, Peter, 1992. Q-learning. Mach. Learn. 8 (3–4), 279–292.

Wei, Yi, Kudenko, Daniel, Liu, Shijun, Pan, Li, Wu, Lei, Meng, Xiangxiu, 2019. A reinforcement learning based auto-scaling approach for saas providers in dynamic cloud environment. Math. Probl. Eng. 1–11.

Xhafa, Fatos, Abraham, Ajith, 2009. A compendium of heuristic methods for scheduling in computational grids. In: Corchado, Emilio, Yin, Hujun (Eds.), Intelligent Data Engineering and Automated Learning - IDEAL 2009. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 751–758.

Xiao, Zheng, Liang, Pijun, Tong, Zhao, Li, Kenli, Khan, Samee U., Li, Keqin, 2017. Self-adaptation and mutual adaptation for distributed scheduling in benevolent clouds. Concurr. Comput. 29 (5), 1–12.

Yang, Jingqi, Liu, Chuanchang, Shang, Yanlei, Cheng, Bo, Mao, Zexiang, Liu, Chunhong, Niu, Lisha, Chen, Junliang, 2014. A cost-aware auto-scaling approach using the workload prediction in service clouds. Inf. Syst. Front. 16 (1), 7–18.